ext3 ファイルシステムへのスナップショット機能の設計と実装

NTT コムウェア株式会社　前野真輝　松尾隆利
VA Linux Systems Japan 株式会社　山幡為佐久

概 要

現在, Linux の標準で利用されているファイルシステムとして, ext3 ファイルシステムがある. ext3 ファイルシステムは, jbd と連携したジャーナリング機能を備えているものの, スナップショット機能を備えていない. スナップショット機能とは, ある瞬間のファイルシステムイメージを記憶したもので, その取得が短い時間ででき, ディスク容量も節約できる機能である. スナップショット機能を持つ事で, バックアップ時に業務を止める必要がなくなったり, スナップショット上の書き込みにより重要な変更を行う前のテスト環境になるなど, その有用性は非常に高い.

有力なスナップショットの実装として LVM があるが, ブロックデバイスレイヤで動作する. ext3 ファイルシステムで動作するスナップショット機能ならば, LVM に比べて, 必要なディスク容量の節約と不要なディスクアクセスの防止, 複数のスナップショットが効率的に作成でき, LVM のスナップショット以上に有用性が高いと考える.

そこで我々は, ファイルシステムレイヤで動作する ext3 スナップショットを実装する. 今回は, ext3 スナップショットの設計, 現段階の実装状況, 性能評価, 今後の実装課題について述べる.

1　はじめに

BSD-ffs [1][1] ではディスク書き込みの依存関係を注意深く追跡する soft updates [2] が実装され, それを元にスナップショット機能 [3] が実装された. ext2 ファイルシステム [4] は ffs の設計を元に実装されたファイルシステムであり, ext3 ファイルシステム [5] は ext2 ファイルシステムに対してジャーナル機能 [6] を追加したものである.

Linux 上ではブロックデバイスレイヤで動作する LVM [2] [7][9] を使用したスナップショットが実装されている. しかし, LVM ではスナップショットの取得元のファイルシステムを守るために, スナップショット領域を使い切ってしまうと, スナップショットが突然消滅してしまう.

これに対して ext3 ファイルシステムレイヤでスナップショット機能を実装することにより, 事前に LVM を導入しておく必要はなく, 既存の ext3 ファイルシステムに導入が可能となる. また, ファイルシステムに関する情報を利用する事により LVM に比べて動作の効率や性能が良くできると考えられ, スナップショットの永続化やスナップショット用の領域を使い切った場合の動作の改善といったことが期待できる. そこで我々は, ファイルシステムレイヤで動作する ext3 スナップショットを実装を行うことにした.

本論文では ext3 ファイルシステム上でスナップショットを実装する場合に, 問題となる点とその解決方法を述べる. また, その内の一部を実装し, 性能評価を行った結果について述べる.

2　ext3 の基本動作

ext3 はジャーナルを採用し, システムクラッシュ後にジャーナルをリプレイすることにより高速なクラッシュリカバリを実現している. 複数のブロックを不可分に変更する際に, 一旦変更後の値をジャーナルに書き出しておき, その後実際のブロックへ書き出しを行う. 実際のブロックへの書き出しの完了前にシステムクラッシュが起きた場合は, ジャーナルを使用して完了していない (か

[1] Fast File System の略. 主に FreeBSD で利用されているファイルシステム.

[2] Logical Volume Manager の略. 現在メジャーバージョンが Version2 となっており, LVM2 と略される. 動的にボリュームサイズを変更できる. また, Device-mapper[8] レイヤで実装されており, 暗号化やスナップショット, マルチパスなどの機能も併せ持つ.

もしれない) 変更処理を完了させる．このようにしてファイルシステムの一貫性を保持する．

2.1 ext3 のジャーナルの動作

ext3 のジャーナル処理は jbd [3] として分離されている．ジャーナルとして使用する領域は, ファイルあるいは別のディスクの 2 通りを指定でき, ファイルシステムブロック単位でジャーナルを行う．処理手順は次のようになる．

1. ジャーナル開始通知
   処理が変更する可能性がある最大のブロック数を指定し, jbd は必要とされるジャーナル領域の確保を行う．十分な領域がない場合ジャーナル領域解放処理を行う．
2. 変更前処理
   あるディスクブロックの変更を開始することを jbd に通知し, jbd は必要なメモリの確保を行う．
3. ディスクブロックの変更
   ディスクブロックに書き出される値を作成する (ブロックの値の変更)．
4. 変更完了処理
   あるブロックに対する変更処理を終えたことを jbd に通知する．必要なら前処理, ブロック変更, 変更完了を繰り返す．
5. ジャーナル終了通知
   変更処理が終わりディスクへの書き出しを行って良いことを jbd に通知する．使用しなかったジャーナル領域の解放等の処理を行う．
6. ジャーナルのディスクへの書き出し
7. 実際のブロックへの書き出し
   ジャーナルへの書き出しが完了後, 実際のブロックの書き出しが開始できる．
8. ジャーナルの解放
   ディスクブロックへの変更が全てディスクに書き出された後に, 対応するジャーナル領域を解放し, 次のジャーナルに使用することができる．

[3] journaling block device の略．

2.2 ジャーナルモードについて

ext3 のジャーナル動作には複数のモードがあり, マウントオプションにより指定できる．

- data=journal
  データを含む全ての変更をジャーナルする．
- data=ordered
  通常ファイルのデータ以外に対しジャーナルを行い, ジャーナルを書き出す前に通常ファイルのデータを書き出す．
- data=writeback
  通常ファイルのデータ以外に対しジャーナルを行う．ジャーナルと通常ファイルのデータの書き出し順序関係をつけない．

2.3 ディスク書き出しの依存関係

ext3 のディスク書き出し依存関係の観点からは, ディスク書き出しは次の 3 種類に分類される．

- ジャーナル
  ジャーナル領域に書かれるもの
- データ
  通常ファイルのデータとして書き込まれるもの
- メタデータ
  上記のジャーナル, データ以外の書き込み, ファイルシステムの管理に使用されるもの

ファイルシステムの一貫性を保つためには, メタデータを書き出す前に必ず対応するジャーナルをディスクに書き出しておく必要がある．また, ジャーナル領域は有限であるため解放処理が必要となる．あるジャーナルエントリの占有しているディスク領域解放が出来るのは, 対応するメタデータ (data=journal の場合はデータも) 全てがディスクに書き出された後である．これらの関係を示したものが図 1 である．

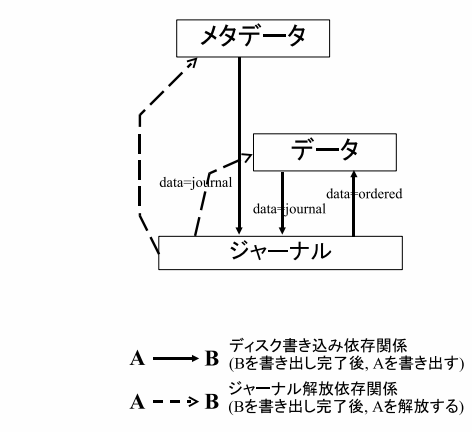

図 1: ディスク書き出し及び解放の依存関係

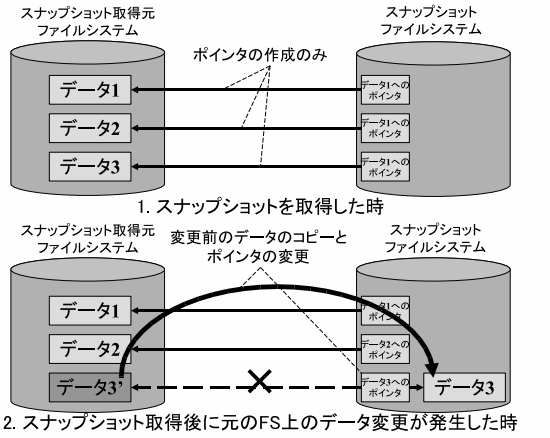

図 2: スナップショットの取得と取得時のデータ保護の機構

ジャーナルを導入することにより, 全ての追加/更新の各操作に対してジャーナルよりも細かい単位で依存関係を追跡する必要がある ffs softupdates に比べて, ディスク書き出しの依存関係が大幅に単純化されている.

図 1 はディスク書き出しの依存関係を示したものである.

この図からジャーナルモードの違いが次のようにまとめられる. data=journal の場合はメタデータとデータの区別を行わない, data=ordered の場合はジャーナルよりもデータを先に書き出す, data=writeback の場合はデータの書き出しには依存関係をもうけない事を示す.

3 スナップショットの基本動作

スナップショットはブロックレイヤで動作するものやファイルシステムレイヤで動作するものがあるが基本的な動作は同じである.

スナップショット取得時からデータが変化する場合には, 変更前のデータを退避する事 (=copy-on-write[4]) によってスナップショット取得時のデータを保護する. スナップショットの取得とその取得後のスナップショット取得時データの保護の機構を示すと図 2 のようになる.

copy-on-write 処理により退避したデータはスナップショットを取得したファイルシステム内の1つの通常ファイル (=スナップショットファイル) として保存される. このスナップショットファイルは見かけ上スナップショット取得元ファイルシステムと同じ大きさの巨大ファイルである. 図 3 にこれを示す.

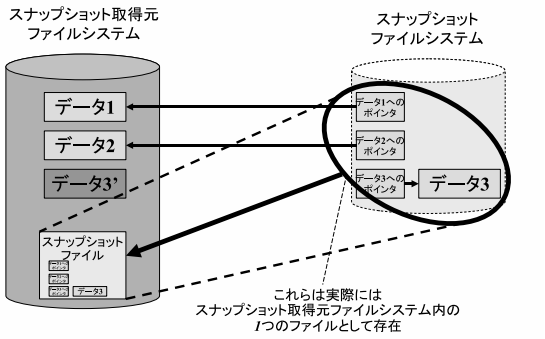

図 3: スナップショットの退避データの格納場所

実際にスナップショットを利用する際には, スナップショットをマウントするためのデバイスを用意する. そのデバイスとスナップショットファイルを関連付けてスナップショットを読み出す. 図 4 にこれを示す.

[4]COW と略される.

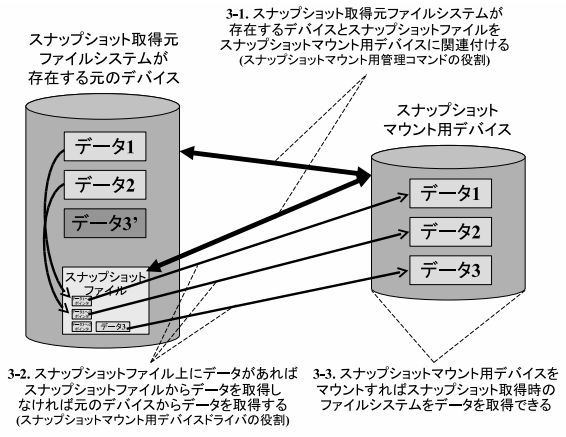

図 4: スナップショットの利用

4 ext3 スナップショットの設計

今回の ext3 ファイルシステムへのスナップショット機能追加のための設計について述べる.

4.1 スナップショットファイル

ext3 ファイルシステム内にファイルシステムと見かけ上同じ大きさのファイルを用意する. ディスクブロックに対する更新が起きた時, そのファイルに対応するブロックの更新前のデータをコピーする (=copy-on-write) ことによりスナップショットを実現する. このファイルのことをスナップショットファイルと呼ぶ. スナップショットファイルを作成するには, 一旦ファイルシステムを停止する必要がある.

4.1.1 ファイルシステムの停止と再開

実行を抑止する必要があるのはブロックの値を変更する可能性のある処理である. つまり, ジャーナル処理を行っているプロセスである. そこで, ジャーナル開始処理及びジャーナル完了処理でジャーナル処理を行っているプロセス数を追跡する. ファイルシステムの停止はジャーナル開始処理部でプロセスをスリープさせ, ジャーナル処理を行っているプロセスがいなくなるまで待つことより実現する. ジャーナル処理は再帰的に開始することができるので, それを考慮する必要がある.

4.1.2 スナップショットファイルの作成

スナップショットファイルにはスーパーブロック, ブロック割り当てを管理する為のビットマップ, i ノード割り当てを管理するビットマップ及びスナップショットファイルに対応する i ノードを含むブロックをコピーする.

4.2 before イメージジャーナル

スナップショット取得中にシステムクラッシュが起き, 再マウントされた場合でもスナップショットは保持される必要がある. これを実現するために before イメージジャーナルと呼ぶもう一つのジャーナルを導入する.

これは更新前のデータ (=スナップショットファイルにコピーされるデータ) を保持しておくジャーナルである. 従来のジャーナルは, 変更後の値を保持するため (before イメージジャーナルに対して)after イメージジャーナルと呼ぶことにする.

これらの依存関係は図 5 のようになる. 特に更新後の値をジャーナル領域に書き出す前に, 更新前の値を before イメージジャーナルに書き出す (=copy-on-write 処理) ところがポイントである.

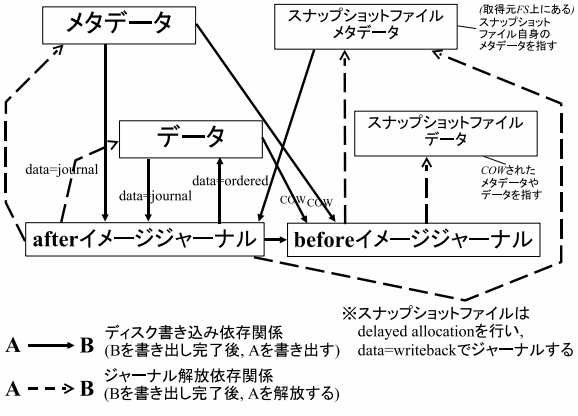

図 5: before イメージジャーナルのディスク書き出し及び解放の依存関係

before イメージジャーナルのフォーマットは従来のジャーナルのものと同じフォーマットを使用できるが, 依存関係の追跡及びクラッシュリカバリは新たな実装の必要がある.

4.3 クラッシュリカバリ

クラッシュリカバリに対してスナップショットの回復処理を追加する.

- 従来ジャーナルのリプレイ
- 従来ジャーナルの有効化
- スナップショットの回復
  before イメージジャーナルからスナップショットファイルに対して更新前のデータをコピーする.
- クラッシュ時に unlink/truncate 中であったファイルの処理

4.4 Copy-On-Write

ffs スナップショットでは copy-on-write をディスクブロックの書き出し部で行っている. しかし, ext3 では一部の処理においてディスクブロックの書き出しが, 既にジャーナル領域に書き出されてしまった後であるので, ディスクブロック書き出し部で copy-on-write した場合, クラッシュリカバリによりスナップショットは失われてしまう.

その為, ext3 スナップショットではデータ更新前に copy-on-write する. copy-on-write 用のフックを次の箇所に追加する.

- メタデータ更新時
  - ジャーナルの変更前処理
  - inode bitmap による inode ブロック内クリア処理
- データ更新時
  - write() 処理
  - mmap() 処理
  - direct_IO() 処理
  - ブロック境界でない truncate down 処理

4.5 ブロック解放抑止

ブロック解放では, データはジャーナルされずそのままブロック解放される. 実際のブロック解放処理の前にスナップショットで参照されているブロックであるかをチェックする. 参照されている場合はスナップショットファイルから該当ブロックを参照するようにする. ブロック解放抑止用のフックは次の箇所に追加する.

- ブロック解放時
  - free_blocks() 処理

4.6 delayed allocation とジャーナル領域

4.6.1 ジャーナル領域

スナップショット取得後, ある処理の途中で copy-on-write 処理やブロック解放抑止処理を行うことは, あるシステムコールを完了する為の変更ブロック数が増えることである. 即ち, ある処理を完了する為に必要となるジャーナル領域が増加することになる.

複数のスナップショットの取得を可能とすると, copy-on-write 処理で必要となるジャーナル量が多くなり過ぎてしまう. この為, 今回の実装ではスナップショット取得可能数を一つに制限している. ブロック解放抑止処理ではある一つのスナップショットがブロックを参照するようにすれば十分なので必要となるジャーナル領域は押さえ込める.

この問題は delayed allocation と呼ばれる技法を使用することで回避できる.

4.6.2 delayed allocation

ブロック割り当て処理を write 時に行うのではなく, ページのディスク書き出し時に行うテクニックである. delayed allocation をスナップショットファイルに適用すると copy-on-write 時にはジャーナルを必要としなくなる.

4.7 スナップショット用デバイス

スナップショットを利用するためには, スナップショットファイルが保持するブロックとスナップショット取得元のブロックを選択的に読み込む必要がある.

ffs スナップショットでは vnode 仮想デバイスへ関連付け, 別のファイルシステムとしてマウントしてスナップショットを読み出している.

今回は, 選択的に読み込みを行う専用のブロックデバイスドライバを作成する. スナップショットファイルをスナップショット用デバイスへ関連付け, そのデバイスを適当なディレクトリにマウントしてスナップショットを読み出す.

5 現時点での実装

現在, 4.1 節と 4.4 節, 4.5 節, 4.7 節で述べた機能が実装できている.

この実装により, パーティション毎に 1 つのスナップショットが取得でき, スナップショット用デバイスを介してスナップショットをマウントして読み出せ, スナップショットを解放できる所まで機能している.

ただし, スナップショット取得元のファイルシステムのアンマウントやシステム再起動やシステムクラッシュにより, スナップショットは消滅してしまう. ディスクフル時には, さらに copy-on-write が発生するとスナップショット取得元ファイルシステムが ReadOnly マウントへ移行するようになっている. その状態でもディスクフルに関与しなかった部分のスナップショットへの読み出しは可能である.

また, before イメージジャーナルや delayed allocation の実装をまだ行っていないため, 1 度に記録するジャーナル量を増やして, copy-on-write されるメタデータやデータをジャーナルへ書き込むように ad hoc な実装で対処している.

6 評価

現時点の実装における ext3 スナップショットを簡単に評価する. 既存のスナップショットとして有力な LVM2 [5] と比較する.

測定項目は、

- スナップショットの取得に掛かる時間とその中でファイルシステムが停止する時間 (後者は ext3 のみ)
- スナップショット取得後の既存ファイルのデータ上書き
- スナップショット取得後の新規ファイルのデータ作成

とする。

表 1: 評価環境

Computer	VALinux2200 (CPU: PentiumIII 1GHz x 2, Memory: 512MB)
OS	Linux (kernel-2.6.8.1-smp をベースにスナップショット機能を付加)
Distribution	Debian 3.1 -sarge-
Disk (スナップショット取得対象)	Quantum Atlas 10K II 36.7GB Ultra160
Small Filesystem	ext3, 1GB (blksize: 1KB, File: 200MB x 2, 4KB x 50,000)
Large Filesystem	ext3, 20GB (blksize: 4KB, File: 4GB x 2, 4KB x 1,000,000)

表 1 の条件だが, LVM の場合は物理 Volume を同一容量 (1GB もしくは 20GB) で作成し, その物理 Volume を 1 つだけ持つ Volume Group を作成した. その同一 Volume Group 上にスナップショット取得元とスナップショット領域用の論理 Volume を以下の容量をそれぞれ作成し, スナップショット取得元となる Volume を ext3 フォーマットした.

[5] Logical Volume Manager Version2 の略.

Small Filesystem の場合にはスナップショット取得元を 550MB, スナップショット領域を 450MB として測定し, Large Filesystem の場合にはスナップショット取得元を 16GB, スナップショット領域を 4GB として測定した[6] .

ちなみに以下に続く節の測定結果表中の ext3 は現時点の実装である ext3 スナップショット機能を追加した ext3 である事を指す.

6.1 測定結果

6.1.1 スナップショットの取得

スナップショットの取得に関しては、I/O が特に発生していない時と I/O に負荷を掛けた時の 2 通りの時間を測定する。

I/O 負荷無し

表 2 は I/O が特に発生していない時のスナップショット取得時間の 3 回の平均を表している. 表 3 はその際のファイルシステム停止時間 (ext3 スナップショットのみ) の 3 回の平均を表している.

ただし, スナップショットを取得した後にすぐ解放し, 再度スナップショットを取得すると, メタデータのキャッシュが効くため, 取得時間が半分近くになる. ファイルシステムの停止時間には変化はない.

表 2: スナップショット取得時間 (I/O 負荷無し)

FS 容量	ext3	LVM
1GB	1.34sec	0.410sec
20GB	2.52sec	0.660sec

I/O 負荷有り

表 4 は新たにブロックを確保してデータを作成し続けて, I/O 負荷を掛けた状態で取得した時のスナップショット取得時間の 3 回の平均を表している. 表 5 はその際のファイルシステム停止時間 (ext3 スナップショットのみ) の 3 回の平均を表している.

ただし, LVM 上で 1GB の場合はスナップショットを取得する前に空き容量がなくなってしまうため, 取得できなかった.

表 3: スナップショット取得時ファイルシステム停止時間 (I/O 負荷無し)

FS 容量	ext3
1GB	0.129sec
20GB	0.0210sec

表 4: スナップショット取得時間 (I/O 負荷有り)

FS 容量	ext3	LVM
1GB	66.7sec	—sec
20GB	85.8sec	137sec

表 5: スナップショット取得時ファイルシステム停止時間 (I/O 負荷有り)

FS 容量	ext3
1GB	38.26sec
20GB	18.44sec

6.1.2 スナップショット取得後の既存データ上書き

既存ファイルのデータの上書きにより, copy-on-write が必ず発生する. ext3 スナップショットと LVM スナップショットの違いは, ext3 スナップショットがディスクブロック単位で copy-on-write するため不必要なブロックをコピーしないのに対して, LVM はブロックよりも大きいエクステント単位で copy-on-write する.

[6] LVM の管理領域に容量が消費されるので, 実際の設定した容量はこれらよりも若干小さい.

表 6 はスナップショット取得後に 2 つのファイルに対してデータを上書きした際に要した時間の 3 回の平均を表している.

括弧内の値は, スナップショットを取得していない状態での同様の操作に要した時間を示す.

表 6: 既存データ上書き時間

FS 容量	上書き容量	ext3	LVM
1GB	200MB x 2	320sec (45.2sec)	83.9sec (40.6sec)
20GB	1GB x 2	1200sec (294sec)	890sec (287sec)

6.1.3 スナップショット取得後の新規データ作成

ファイルのデータの新規作成により, ext3 スナップショットでは copy-on-write が発生しないが, LVM では発生する.

表 7 はスナップショット取得後に 2 つのファイルに対してデータを新規作成した際に要した時間の 3 回の平均を表している.

括弧内の値は, スナップショットを取得していない状態での同様の操作に要した時間を示す.

表 7: 新規データ作成時間

FS 容量	新規作成容量	ext3	LVM
1GB	200MB x 2	50.1sec (49.1sec)	81.0sec (32.7sec)
20GB	1GB x 2	289sec (280sec)	811sec (280sec)

6.2 測定結果考察

スナップショットの取得は, I/O 負荷を掛けた状態だと非常に遅くなってしまう. しかもファイルシステム停止時間も無視できない時間となっており, 大きな課題である. LVM と比較しても遅い.

これは現在のアルゴリズムが性能を考慮した実装になっていないためであり, 改善の余地がある. 7 節でその改善策について述べる.

スナップショット取得後のデータの上書きは, LVM の方が, copy-on-write の単位が大きく効率が良いため, ext3 スナップショットよりも良い結果となっている.

スナップショット取得後のデータの新規作成は, ext3 スナップショットでは copy-on-write が発生しないが, LVM では copy-on-write が発生するため, ext3 スナップショットの方が LVM よりも良い結果となっている.

また, スナップショット機能を付加していない通常の ext3 ファイルシステムに比べて, スナップショット機能を付加した ext3 ファイルシステム上のデータ作成が遅くなっている. 今回, copy-on-write 処理の為に予約するジャーナル量を増やしているため, ジャーナル領域が不足してジャーナルのディスクへの書き出し処理が動作する割合が高くなっていると思われる. その為, スナップショットを取得していない場合でも, 新規データ作成時間が通常の ext3 ファイルシステムに比べて遅くなっていると考えられる.

ただし, いずれにしても実装が完了していないため, この測定結果の比較は現状報告と言った参考程度に留めて欲しい.

7 まとめ

本論文では ext3 スナップショットを実現するための設計とその一部実装と評価について述べた. この設計を実装する事により, ext3 スナップショットが実現できる.

今後の実装上の課題として以下が挙げられる.

- コードの安定化
- ディスクフル時の動作改善

- スナップショット取得の性能向上
- スナップショットの永続化
  before イメージジャーナルの実装及びシステムクラッシュ回復処理の実装
- 複数のスナップショットの取得
  delayed allocation の実装

スナップショット取得の性能向上は, 現在の機構だとスナップショット取得時のメタデータのコピー処理で, その都度ブロックを取得しているので前もって必要なブロックを確保した方が効率が良いという課題である.

スナップショットの永続化させる場合, before イメージジャーナルの実装とシステムクラッシュ回復処理の実装が必要となる. 特に before イメージジャーナルの依存関係追跡は複雑になると思われる.

複数のスナップショットを取得する場合, 現在の実装だと copy-on-write 数が増えるため, delayed allocation の実装を行い, ジャーナル量を増やさないための実装が必要となる. もしくは, copy-on-write 数が増えないような複数のスナップショット間のデータを参照するように実装しなければならない.

これらの実装を完了した後に, LVM と ext3 スナップショットで性能比較を行い, ext3 スナップショットの意義を立証したいと考えている.

また, ある時点でソースコードを広く公開し, 今後も開発を進めていきたいと考えている.

スナップショットのその他の課題としては以下が挙げられる.

- スナップショットへの書き込みを可能にする
- スナップショットからのブートを可能にする

謝辞

今回の ext3 スナップショットの設計と実装は, NTT コムウェア株式会社と VA Linux Systems Japan 株式会社の共同プロジェクトとして実施した.

プロジェクトの実施にあたり, VA Linux Systems Japan の高橋浩和氏, 小田逸郎氏, 箕浦真氏, 山本高志氏, 黒澤崇宏氏, NTT コムウェアの北井敦氏, 若山弘和氏, 荒井晋二氏, 佐々木博正氏, 大塚憲司氏をはじめとした関係者の皆様の御協力に深く感謝すると共に, Linux をはじめとしたオープンソースソフトウェア開発に貢献する皆様に厚く御礼申し上げます.

参考文献

[1] M. McKusick, W. Joy, S. Leffler, and R. Fabry: A Fast File System for UNIX. ACM Transactions on Computer Systems, 2(3):181–197, August 1984

[2] Marshall Kirk McKusick and Gregory R.Ganger: 'Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem', Proceegings of the FREENIX Track. The 1999 USENIX Annual Technical Conference.

[3] Marshall Kirk McKusick: 'Running "FSCK" in the Background'. In USENIX BSDCon 2002 Conference Proceedings, pages 55-64, February 2002.

[4] Remy Card and Theodore Y. Ts'o and Stephen Tweedie: Design and implementation of the second extended file system. The 1994 Amsterdam Linux Conference, 1994.

[5] Theodore Y. Ts'o and Stephen Tweedie: Planned Extensions to the Linux Ext2/Ext3 Filesystem. The 2002 USENIX Annual Technical Conference, Monterey, CA, June 2003.

[6] Stephen C. Tweedie: Journaling the Linux ext2fs File System LinuxExpo'98, 1998.

[7] LVM2 Resource Page
(http://sourceware.org/lvm2/)

[8] Device-mapper Resource Page
(http://sources.redhat.com/dm/)

[9] Heinz Mauelshagen (RedHat Consulting Development Engineer): Logical Volume Manager (LVM2). Webcast LVM2, 21 Sep 2004.