基幹システムの連続稼動のためのLinux LVMミラー機能の拡張

海野 将之　高田 充康　ホルガー・スモリンスキー

Linux LVM Real-time Enhancement for Mission Critical Systems

Masayuki Unno, Michiyasu Takada and Holger Smolinski

企業の生命線を握る基幹システムでは, 長期のシステム・ダウンを排除して連続稼動する必要があり, 全ての構成要素の冗長化が重要である. 鍵となるのがディスク・サブシステム障害に対応する筐体(きょうたい)間ミラーの実装であるが, Linux®は筐体間ミラーを実現する機能を実装していない. 筆者らは, Linuxの Logical Volume Manager（LVM）のミラー機能を拡張して筐体間ミラーを実現し, 実システムで実装して有効性を確認したので報告する.

Mission critical systems require all system components to be redundant and long-time outages have to be eliminated. The key point is to implement disk mirroring to achieve continuous operation even in case of disk subsystem failure. However, Linux® does not have mature mirroring functions across storage subsystems. In this paper, we describe how we were able to meet the extremely high-availability requirement by overcoming a weakness in Linux with an IBM enhancement of the Logical Volume Manager, to which we contributed to the development and deployment through an actual customer project.

Key Words & Phrases：Linux, 連続稼動, LVM, DS8000, System z
Linux, continues availability, LVM, DS8000, System z

1. はじめに

近年, 企業ではTCO（Total Cost of Ownership）削減に加えて, 社会的責任としてエネルギー削減など地球温暖化対応も求められることから, 仮想化機能を備えた信頼性の高いサーバーやディスク・サブシステムに基幹システムを統合する動きが進んでいる［1］. 企業の生命線を握る基幹システムでは, 長時間のシステム・ダウンを排除して連続稼動する必要があるため, 全ての構成要素の冗長化が重要であるが, 特にシステム統合では筐体レベルの障害が発生した場合の影響範囲が広がり, 一層の考慮が必要となる.

サーバーの冗長化は, クラスター化によって考慮されることが多いが, ディスク・サブシステム筐体障害時の連続稼動に備えた対策を行うケースは少ないのが現状である. しかし, データ・ロストの可能性を加味すると, 信頼性要件はサーバーよりも厳しいと言える［2］. ハイエンドのディスク・サブシステム製品では, 筐体内部の主要構成要素を冗長化することで高信頼性を確保しているが, 筐体全体やRAIDグループ（IBM製品ではランクと呼ぶ）の障害の可能性を完全には排除できない. 従って, 基幹システムでは, 筐体をまたいだディスク冗長化機能, 具体的にはデータ・ミラー機能の実装は必須と言える.

データ・ミラーは, 主に以下の4つのレイヤーでの実装が考えられるが, 汎用性・連続稼動性に最も優れるのはOS機能による実現である.

- IBMディスク製品の Peer-to-Peer Remote Copy（PPRC）などディスク・サブシステムの機能
- Oracle ASM：Automatic Storage Managementのミラー機能など［3］ミドルウェア製品の機能
- ユーザー・アプリケーションによる二重書き
- AIX® Logical Volume Manager（以下, LVM）[1]のミラー機能［4］などOSの機能

幅広いハードウェアで稼動するLinuxはサーバー統合を促進するオープンな基盤として基幹システムでの採用が進んでいるが, ディスク・サブシステム筐体間ミ

提出日：2007年9月3日　　再提出日：2007年12月27日

1 複数のハード・ディスクやパーティションにまたがった記憶領域を一つの論理的なディスクとして扱うことのできるディスク管理機能で, パーティションを切り直さずに, ファイルシステムのサイズ変更ができるなどの特徴がある.

ラーを実現する機能を備えていない. Linux®にもAIXと同様にLVMのミラー機能が存在するが, ミラーの状態を保持するログ・ボリュームと呼ばれる領域がミラー化されないため, ログ・ボリュームが存在する筐体に障害があった場合には機能しない[5][6].

著者らは, Linux LVMのミラー機能を, ディスク・サブシステムの筐体障害時にもデータ・ロスなしに業務を継続できるように拡張し, それを, IBMメインフレームSystem z™上で稼動するLinuxとIBM System Storage™製品の最上位機種であるDS8000™ [7]をベースとしたシステムで実装して有効性を確認したので本論文で報告する.

以下, 2章で筐体間ミラーにおける要件と既存機能の対応を整理し, 3章でその要件を満たすためのLVMミラー機能拡張を紹介し, 4章で具体的な実装事例と評価結果を考察する.

2. 筐体間ミラー機能に関する要件

2.1 Linux基幹システムでのディスク冗長化要件

一般的に業務アプリケーションには数十秒から数分のタイムアウト値が設定され, ディスク・サブシステムの筐体障害やビジー状態によりこれに達した場合は, アプリケーションからエラーを返し, ディスクが正常にアクセスできる状態になるまでは処理を再開できない. そのため, タイムアウト値以内に障害箇所を自動検知して切り離し, I/Oを継続することは必須要件である. 以下に, Linux on System zとDS8000の組み合わせを例にとってシステム構築・運用面での考慮点も含めてディスク冗長化要件をまとめた.

A　アプリケーション要件

① DS8000筐体またはランク障害を自動検知し, アプリケーションのタイムアウト値以内にI/Oを再開し, 業務を継続
② DS8000内部のロング・ビジー状態においても, アプリケーションのタイムアウト値以内にI/Oを再開し, 業務を継続

B　システム要件

① ミラー構成になっても, 通常のLVM論理ボリューム（ストライプ, リニア）を使用でき, 構成の柔軟性を確保
② ミラーのステータスを保持することができ, ミラー間の内容に差異が生じた場合も差分のみの同期が可能
③ ミラーの同期, 切り離しなどのオペレーションがコマンドにより可能
④ Linuxとディスク・サブシステムのみで完結するシンプルな基盤
⑤ 障害箇所のみミラーの自動切り替えを実施

次節以降では, 上記の要件をハードウェア層とソフトウェア層における幾つかの実装により, 実現可能かどうかを検討する.

2.2 ハードウェア層による筐体間ミラー

まず, 代表的な実装としてDS8000が提供する同期ミラー機能PPRCメトロミラーを検討する[7]. Linuxからミラー操作は透過的であり, 前節で定義したB-①～④の要件は満たす. 一方, 障害発生時にターゲット・ボリュームへの切り替えが手動となり, タイムアウト値以内の自動復旧はおろか, 短時間での復旧も困難であるため, A-①は満たさない. また, A-②については, ロング・ビジー状態を検知する仕組みがなく, 誤検知を誘発しないための判断の見極めと切り替えの自動化が困難である. さらに, 大量のディスク・ボリュームを持つ場合, 問題範囲の特定が困難であり, B-⑤の要件も満たさない.

次に, PPRCを拡張したGDPS HyperSwap[8]を考えるが, z/OS®前提のソリューションであることからB-④の要件を満たさない点に加え, 1ボリュームのエラーであっても管理下の全ペアの切り替えが発生し, 統合されたサーバー群全体に影響するため, B-⑤にも合致しない.

以上から, ハードウェア層での筐体間ミラーは2.1であげた要件全てを満たすことは困難であると判断できる.

2.3 ソフトウェア層による筐体間ミラー

ソフトウェア層は, アプリケーション, ミドルウェア, OSの3層に細分化できるが, 汎用的に適用できるOS層に絞り, LVMとソフトウェアRAIDを検討する. LVMはdevice-mapper（DM）と呼ばれる論理/物理デバイス間のマッピング機構を管理する. Red Hat Enterprise Linux 4（以下, RHEL4）よりLVMにミラー機能が追加された[5][6][9]が, 前述のとおりミラーの状態を保持するためのログ領域が冗長化されないため, A-①に対応できない. また, ロング・ビジー状態を検知できないため, A-②を満たせない. さらに, 複数の物理ディスクで構成され論理ボリュームをミラー化できないため, B-①を実現できない. 以上から, 標準LVMでの実装は不可能と言える. 一方のソフトウェアRAIDも, A-②, B-②などに対応できないこと, またLVMと比較して実績の少なさや品質面の不安も伴っており, こちらも採用は不可と判断する[2].

他にも商用のボリューム管理製品による対応も考えられるが, Linuxディストリビューションの対応状況や障害発生時のマルチベンダー間の責任範囲などで課題が残り, 結局は案件毎の個別対応になってしまうため汎用性に乏しい.

3．筐体間LVMミラー機能の拡張

2.3で述べたとおり, LVMとDMベースのミラー機能は, 現状では2.1の要件を満たすことができないが, オープンソースの標準ボリューム管理機能という特性から, これを拡張することで要件を満たすことができると判断し, 筆者らは開発を行った. 以下に具体的な内容について説明を行う. なお, 開発した拡張機能は正式にはreal-time enhancements [2] [10] と呼ぶが, 以下「ミラー機能拡張」と言う.

3.1　要件への対応

ミラー機能拡張では, 次の機能を実装することで, 2.1で述べた要件を満たした [2] [10].

・ログ領域の冗長化（A-①, A-②）

従来は単一ボリュームで構成されていたログ領域を冗長化し, 筐体間のミラーを実現

・片系稼動での業務継続（A-①, A-②）

従来はRead-Onlyでのみ可能であったミラー片系での起動を可能とし, 片系障害時でも正常系のみでLinuxの起動・業務開始が可能

・障害の自動検知および業務の継続（A-①）

DMはデバイス・ドライバからのI/Oエラーを検知し, 即座に障害が発生した片系を自動的に切り離して業務を継続

・ロング・ビジー発生時の業務の継続（A-②）

論理ボリューム単位にタイムアウト値を設定し, 片系へのI/Oがタイムアウトとなった際はその片系を自動停止し業務を継続

・論理ボリューム構成の柔軟性（B-①）

ストライプ, リニア論理ボリュームをミラー化するLVMの二層構成が可能

・ミラー状況保持による差分同期（B-②）

タイムアウトによりミラーに差分が発生するケースでは, 管理テーブルで同期情報を保持し, 差分のみの同期が可能

・ミラー操作のコマンド（B-③）

ミラーの切り離し, 再結合, 障害復旧などミラーに特化した操作のためのオプションをLVMコマンドに追加

・Linuxのみで完結するシンプルな基盤（B-④）

ミラー機能拡張はLinux単体の実装のため, 他OSや特別なハードウェア機能は不要

・障害箇所のみがミラー切り替えの対象（B-⑤）

論理ボリューム単位で個別にエラーを管理するため, 障害時の影響範囲の局所化が可能

3.2　ミラー機能拡張の特徴

ミラー機能拡張の基本的なレイヤー構造を**図1**に示す. まず, ディスク・サブシステムの各筐体内部の複数のディスク（Physical Volume：PV）からボリューム・グループ（Volume Group：VG）を構成し, この中から必要なサイズの論理ボリューム（Logical Volume：LV）を作成するまでは標準のLVMにおける定義と同じである. 次に, 各筐体で作成したLVをPVとして定義し, これらをVGとしてまとめる. このVGからLVを作成する際に, ミラーとして定義する.

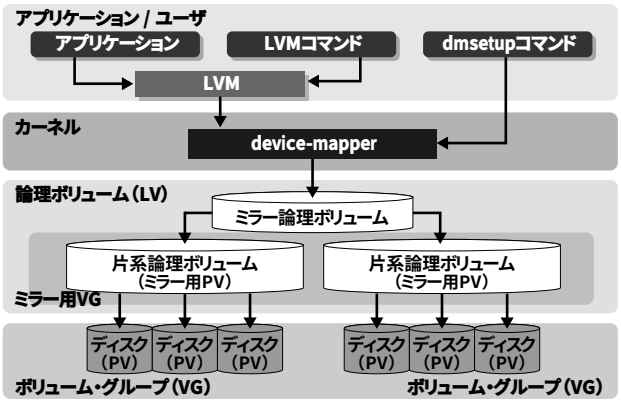


図1.　LVM/device-mapperのレイヤー構造

ミラーLVは**図2**のようにミラー本体, ミラー・ログ, ミラーの実体であるミラー・イメージで構成される. アプリケーションやユーザーが意識するのはミラー本体のみであり, DMによりミラー・イメージ双方にI/Oが発行され, 同期がとられている. ミラー・イメージの同期情報はミラー・ログ上のリージョン単位で管理される. タイムアウトにより左右のミラー・イメージで差分が発生すると, 該当リージョンの状態が"out-of-sync"となり, 再度同期をとることで"sync"状態に戻る.

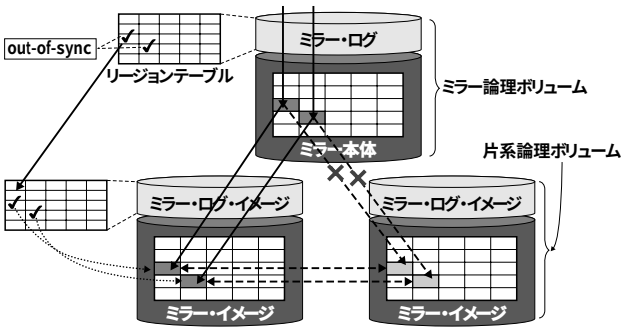


図2.　ミラーLVの構成

3.3　ミラーI/Oの仕組み

A　通常時の動作（図3）

・Write I/O

① ミラー・ログ内のWrite対象の領域に対応するリージョンをout-of-sync状態に変更

② 実データのI/Oを実施
③ 両系へのデータI/Oが完了した後, ①でout-of-syncに書き換えていたリージョンをsync状態に更新しI/Oをコミット

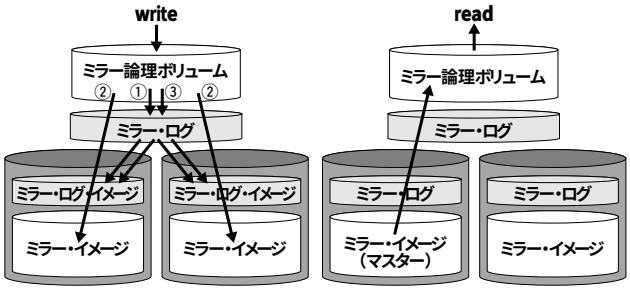


図3. 通常時のミラーI/Oの動作

・Read I/O

① ミラーのマスター側からReadを実施

B I/Oエラー発生時の動作（図4）

・Write I/O

片系からI/Oエラーが返った場合, DMはその片系を即座に切り離し, 正常な片系での稼動に切り替える. I/Oは正常時と同じ3ステップで正常な片系に対してのみ発行される.

・Read I/O

マスター側のI/Oが正常の場合, ミラー・ボリュームはI/Oエラーを検知せずそのままI/Oを実施する. 障害中の場合は, エラーを検知して正常な片系からReadを実施する.

このようにI/Oエラーを検知して障害系ボリュームを切り離して稼動する状態をInactiveと言う.

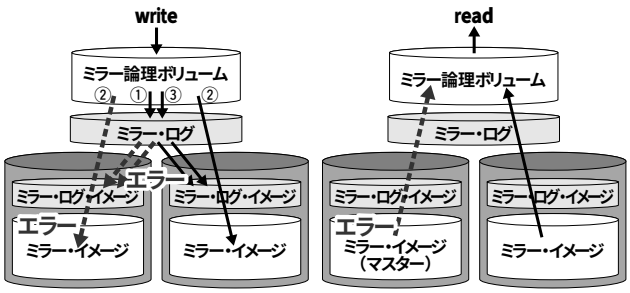


図4. ミラーがInactive状態の場合のI/O動作

C 全パス障害やロング・ビジー時の動作（図5）

・Write I/O

片系へのI/Oでタイムアウトを検知した場合, 正常な片系に切り替えてI/Oを継続する.

① ミラー・ログ内のWrite対象の領域に対応するリージョンをout-of-sync状態に変更
② 実データのI/Oを片系に対してのみ実施

・Read I/O

マスター側のI/Oが正常だった場合, DMはタイムアウトを検知せず, そのままRead処理を実施する. 障害中の場合は, タイムアウトを検知して正常な片系からReadを実施する.

このケースでは, リージョンをsync状態に更新しI/Oをコミットする処理が発生せず, 片系稼動中にWrite I/Oが発生した領域はout-of-syncのまま増加していく. このようにタイムアウトを検知して片系稼動に切り替え, リージョン数の差分を管理する状態をDegradedと言う.

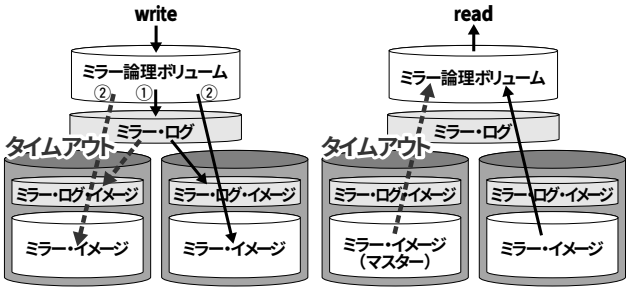


図5. ミラーがDegraded状態の場合のI/O動作

3.4 復旧方法

ミラー機能拡張では, 障害の発生パターンにより以下のような方法を用いて復旧する.

A I/Oエラーでの片系稼動時（図6）

- ディスク障害によりI/Oエラーが発生し, ドライブの交換や初期化が行われた場合
- ランクや筐体障害によるデータ破壊を想定
- LVMを構成するメタ情報が失われるため, ハードウェア側の復旧と合わせてLVMの再構築を実施後, 全量同期を実行して復旧する

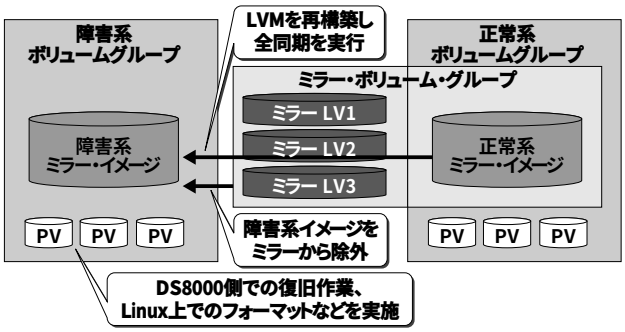


図6. ディスク再構成を伴う全量同期

B オンライン中のI/O停止

- ミラーLVが稼動中にロング・ビジーなどによりタイムアウトとなった場合
- 稼動中マイクロコード保守, RAID再構成や制御装置のリカバリ, 全パス障害を想定
- I/Oが復旧次第, 自動的に差分の再同期が開始

C ロング・ビジー中にミラーが片系で起動

- ロング・ビジー中にミラーLVが起動され, タイムアウトで片系となる場合

・想定事象はAと同様
・LVMの構成情報が失われておらず全量同期の実行のみで復旧

4．適用事例と評価

4.1　事例紹介

筆者らが参画したプロジェクトでは，各店舗に配置される数百台以上のWindows®ベースの営業店システムを複数台のIBM製メインフレームSystem z9®上のLinuxへ統合している．営業店システムは基幹系の一部として窓口業務を担い，長時間の停止は窓口を利用する顧客に対して直接の影響を及ぼすため，極めて高い可用性が要求される．これまで障害時の影響範囲は該当の営業店に限定されていたが，統合によりハードウェア単体の信頼性は大幅に向上する一方で，筐体障害が発生すれば，複数店舗に影響範囲が拡大するリスクがある．従って，全構成要素を冗長化し，停止時間を極小化する設計を行い，最も困難な課題であったディスク・サブシステムの冗長化に対応するため，ミラー機能拡張を採用した．

4.2　論理ボリューム構成

・ブートおよびルート領域

ブート領域はLVM未対応のため，単一ボリュームをペアのDS8000にそれぞれ用意し，ランク障害や筐体障害に備えてどちらからでも起動可能とした．ルート領域は起動時の初期化プロセスにおいてLVMの二層構成に未対応のため，単一ボリュームで構成されるミラーとした．

・スワップ領域

他の領域ほど重要ではないため，独立した領域で構成し，スワップ領域の障害が他の領域に影響を及ぼさないようにした．

・/var，/opt，/usrなどのシステム領域

数百MBから数GBまで大小様々なボリュームがあるため，複数のディスクを束ねて大きな論理ボリュームを各DS8000 上に構成し，これらの上で領域毎に柔軟な容量でミラーLVを構成する二層構成とした．

・共有データ領域

IBM製クラスター製品であるTivoli® System Automation（以下，TSA）を採用し，障害時には共有ディスクがTSA待機系ノードに引き継がれる．この共有ディスク上のデータ領域は，ファイルシステムとして利用するボリュームとrawデバイスとして利用するボリュームでVGを分け，二層構成として大小入り混じったボリュームを効率良く配置した（以上，**図7**）．

図7．ミラーLV構成事例

4.3　運用自動化方法

当プロジェクトでは64クラスター（計128システム）が同時稼動するため，可能な限り運用と監視を自動化する必要があった．筆者らは，TSAでミラーLVの起動や引き継ぎを自動管理するにあたり，以下の点を工夫した．

・ミラー正常起動の確認

ミラーLV起動処理の最後に，dmsetupコマンドを実行してミラーのステータスを確認する．この際に両系障害が発生していた場合，ミラー障害が発生した旨を通知して処理失敗のリターンコードを返す．片系稼動の場合は，ミラーLV自体が起動すれば正常完了のリターンコードを返す．

・各処理の自動リトライ

ミラーLVを構成するディスクや一層目の論理ボリュームの起動処理の結果が想定外だった場合はリトライ・ロジックを含めて，片系起動のリスク低減に対応する．

・論理ボリュームの存在監視

30秒置きに実行されるステータス管理コマンドの中に，ミラーLVのステータスを確認するロジックを追加した．これにより，主系Linuxでの論理ボリュームの停止，待機系Linuxでの想定外の論理ボリュームの起動，あるいは両系障害を検知した際に資源の停止処理を実施して二重マウントなどのリスクを回避する．

4.4　チューニング・ポイント

ミラーLVの運用やアプリケーション処理に特有なチューニング要素があるが，プロジェクトでは以下の3つの対応により，パフォーマンスの最適化を行った．

・リージョン・サイズ

リージョン・サイズは復旧時の同期時間に影響する．テストの結果，256KB以下のリージョン・サイズでは同期時間が大幅に上昇している（**図8**）．これはチャネル・サブシステムの転送レートが1MBのため，小さなリージョ

ン・サイズの場合は効率が悪いためである. この結果を基に論理ボリュームの最小単位であるエクステント・サイズと同じ4MBを設定した.

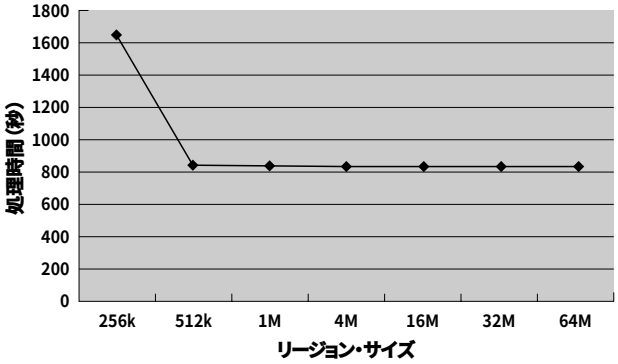


図8. リージョン・サイズと同期時間の関係

・ミラー・タイムアウト値

ミラー・タイムアウト値はLV毎に設定されるため, 複数LVに対してI/Oを発行する処理の場合はその数だけタイムアウトが発生し, アプリケーション・タイムアウトにつながりやすい. 従って, ミラー・タイムアウト値について以下の3つの観点で検討を行った.

① 基幹系ホストのタイムアウト値(15秒)を参考とする.
② 過去の負荷テスト結果の分析により, 100%ピーク時のI/Oレスポンス最大値が2秒以内であり, ミラー・タイムアウトも発生していないことから, ①の15秒は余裕のある値と言える.
③ DS8000内部でのロング・ビジー発生により, タイムアウトを検知して片系稼動になる可能性はあるが, 数十秒程度の片系稼動であれば自動復旧時の同期処理の影響も軽微である.

以上から15秒をデータ領域のタイムアウト値として設定した. 結果として100%ピーク処理時にDS8000筐体へのI/Oを切断する検証においても, アプリケーションのタイムアウトは一切発生せず全トランザクションが継続処理された.

・LVMコマンド発行時の対象デバイス数

オンライン中の業務停止は最大5分とされており, この間にTSAで管理資源の引き継ぎを完了し, 業務を再開する必要がある. 当初はLVの起動だけで1分半近くを要し, アプリケーションの起動も含めると目標を達成できなかったが, 特にLVMメタ情報を読み込むvgscan, LV起動のvgchange処理に時間を要した. これらは認識する全デバイスを走査してLVMメタ情報を読み込むため, 大量ディスクの環境では実行時間が増加傾向にあった. また, 今回は二層のミラー構成のため, 上記コマンドが2回発行され, 引き継ぎ時間の増大につながっていた. さらに, 一部のディスクでI/Oが停止した場合, 無関係な論理ボリュームの起動もI/O停止の影響を受け, 起動時間が大幅に遅延する問題も発生していた. これらに対応するため, ミラー機能拡張ではLVM_EXTRACONFIG環境変数が実装されており, この環境変数を用いて, コマンド実行時に走査対象デバイスを絞ることができる. 今回は1回目のLVMコマンド実行時には対象となるVGを構成するPVのみを対象とし, 2回目は一段目のLVのみを対象とした. この結果, 約90秒だった起動時間を約20秒削減し, アプリケーションのチューニングと合わせて目標引き継ぎ時間5分をクリアした(**表1**).

表1. EXTRA_CONFIG対応による起動短縮時間

対応前	対応後	短縮時間	引き継ぎ時間
86秒	68秒	18秒	4分40秒

4.5 ストレージ構成の考慮点

ミラー機能拡張では, 通常I/Oの前後にログI/Oが発生するため, これを考慮したキャパシティ計画, データ配置が必要である. また, アプリケーションからI/Oが発行される領域の種類によって, 影響度が異なることも考慮が必要である. 今回, DB領域はRAWデバイス, 業務データ領域はExt3ファイルシステム(同期オプションでマウント), 問題判別のためのアプリケーション・トレース領域はExt3ファイルシステム(デフォルトの非同期オプションでマウント)を使用しており, I/O増加率は**表2**のようになった.

表2. 領域種別のミラー化に伴うI/O増加率

	片系あたりのI/O増加率
RAWデバイス	約3倍
EXT3(SYNC)	約2倍
EXT3(ASYNC)	約3-5倍(ファイル・キャッシュ書き出し時のブロックサイズに依存)

4.6 大規模障害テスト結果

プロジェクトでは, 過負荷状態でのSystem z9筐体障害を想定した大量ミラーLVのTSAによる引き継ぎ(①)と片系DS8000に対する全パス切断時のI/O継続性(②)を確認する検証を実施した.

①の結果は, 全LPAR(計14)の引き継ぎは5分以内で成功裏に完了し, 引き継ぎ後も安定したスループットが得られた. ②では, ミラー・タイムアウト(15秒)が正常に動作することで, 該当の全ミラーLVは片系稼動に切り替わり, 全トランザクションがタイムアウトやエラーが発生することなく継続できる結果となった.

上記検証結果により, ミラー機能拡張の性能, 連続稼動検証は完了し, ミッションクリティカルな基幹システムにおいても十分耐え得る性能と信頼性を保持していると判断している.

5. おわりに

LinuxのLVMミラー機能を筐体間で機能するように拡張することにより, 従来は対応が困難であったディスク・サブシステム障害時の連続稼動を実現できた. また, ミラー機能拡張はディスク・サブシステムのマシンタイプに依存せず, 本論文で取り上げたDS8000以外の機種でも稼動することに加え, OS単体での実装でありミドルウェアやアプリケーションからは透過的であることから, 汎用性は非常に高い.

ミラー拡張機能は現状では標準のLinuxカーネルとLVM/DMに取り込まれていないため, Red HatやSUSEが提供する主要ディストリビューションには含まれていない. そこで, IBMでは, これをSystem z上のRHEL4をベースとしたLinux保守を含むサービス・ソリューションとして提供しており, 他のディストリビューションでの拡張も検討中である. 適用事例で取り上げたプロジェクトのように連続稼動性が最重要課題であり, サービス開始後は必要最小限のパッチ適用を除いて今後数年間にわたってバージョンを凍結するようなシステムでは, 追加コストを相殺するメリットがあり, 検討に値する.

ミラー機能拡張は基幹システムにて実証済みの機能であり, Linuxディストリビューションにおいて採用されることで, 基幹システムにおけるLinuxの地位向上に寄与すると確信している. ボブリンゲン研究所と協業して, 標準機能として取り込まれるよう活動を継続して行きたい.

参考文献

[1] 日本アイ・ビー・エム: "Project Big Green, "http://www.ibm.com/systems/jp/saiteki/cost_efficiency/energy_efficiency/approach/.

[2] Horst Hummel:"Real-time enhancements for SW-RAID1: Securing applications against Storage controller failures with Linux, " IBM System z Expo Session L97, San Antonio (2007).

[3] Oracle Cooperation:"AUTOMATIC STORAGE MANAGEMENT _ THE NEW BEST PRACTICE, "Paper # 40288 (2003).

[4] IBM Corporation:"AIX Logical Volume Manager from A to Z: Introduction and Concepts, " IBM Redbooks, SG24-5432-00, pp.101-136 (2000).

[5] Jonathan Brassow: "LVM Mirroring, "Linux Cluster Summit, Walldorf (2005).

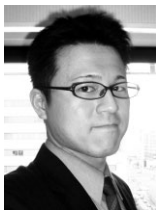
[6] Matthew T. O'Keefe: "New availability features in Red Hat Enterprise Linux 4, "Red Hat Magazine, Issue #7 (2005).

[7] IBM Corporation: "IBM System Storage DS8000 Series: Copy Services with IBM System z, " IBM Redbooks,SG24-6787-02 (2006).

[8] IBM Corporation: "GDPS/PPRC V3R3 Installation and Customization Guide, " ZG24-6703-08 (2006).

[9] Klaus Heinrich Kiwi:"論理ボリュームの管理, "IBM developerWorks投稿記事 (2007).

[10] IBM Corporation: "How to operate the disk mirroring target with real-time enhancements, "LNUX-MIRR-00 (2006).



日本アイ・ビー・エム
システムズ・エンジニアリング株式会社
エンタープライズ・サーバー
ITスペシャリスト

海野 将之 Masayuki Unno

[プロフィール]

2005年に日本アイ・ビー・エム システムズ・エンジニアリング入社. z/VMのフィールド・サポートを中心に, Linux on System zのITスペシャリストとして技術支援に従事. 2006年より本論文の事例である大規模統合案件をサポート.

日本アイ・ビー・エム株式会社
システム製品テクニカルセールス
ACPシニアITスペシャリスト

髙田 充康 Michiyasu Takada

[プロフィール]

1995年に日本アイ・ビー・エム入社. 製造業のお客様担当SE, メインフレーム上のTCP/IPやセキュリティ製品の担当を経て2000年よりLinux on System zの技術支援に従事. このエリアに特化したITスペシャリストとして国内の代表的なサーバー統合プロジェクトにおけるシステム設計をリード.

アイ・ビー・エム
システムズ・アンド・テクノロジー・グループ
Linuxテクノロジー・センター

ホルガー・スモリンスキー Holger Smolinski

[プロフィール]

ボブリンゲン研究所開発部門のアドバイザリー・エンジニアで, Linux on System zプロジェクトに貢献してきた. 現在は製品開発部隊のストレージ・アーキテクトとしてデータ管理に関する各開発チームを支援. 本論文のミラー拡張機能の開発をリードした.

smolinski@de.ibm.com