チュートリアル: device-mapper 解説

日本電気株式会社 野村淳一

device-mapper について、2.6.31 で取り込まれる予定の request-based
device-mapper の機能も含め、動作原理や使用方法を解説する。

1. device-mapper とは
2. device-mapper の背景と利用例
3. Linux の storage stack と request-based device-mapper

---

1. device-mapper とは

device-mapper は、2.6 カーネルから取り込まれた Linux のデバイスドライバで、
一つまたは複数のブロックデバイスへの I/O をとりまとめ変換を加える機能を
提供する。
具体的には、Linear（ブロック位置のオフセット変更)、Mirror（I/O 要求を複数の
デバイスへ発行)、Stripe（I/O 要求を一定サイズで分割して複数のデバイスへ発行)、
Crypt（ブロック単位での暗号化・復号)、Multipath（I/O 要求をいずれか一つの
デバイスに発行）などの機能が実装されており、
LVM2（論理ボリュームマネージャ)、dmraid（ソフトウェア RAID 管理ツール)、
cryptsetup（暗号化デバイス管理ツール)、multipath-tools（冗長パス管理ツール）
といったツールから利用されている。

2. device-mapper の背景と利用例

device-mapper が 2.6 カーネルに取り込まれるきっかけとなったのは、
LVM 機能の開発である。
2.4 カーネルでは、LVM 機能をメタデータの管理も含めてデバイスドライバ側で
行なっていたため、多機能化していく際のコードの肥大化などが問題であった。
2.6 カーネルでは、I/O のマッピングなど最低限の処理をドライバ側に残し、
メタデータの管理・認識などはユーザスペースのプログラムに移すことで、
カーネル側のコードを軽量化し、また異なるボリューム管理機能間で共有可能にした。
同様の議論は、ソフトウェア RAID、マルチパスといったストレージ機能についても
起こり、それぞれ dmraid、cryptsetup、multipath-tools などのユーザスペース
プログラムと対応する device-mapper のドライバが開発された。

2.1. LVM2

LVM2 は、論理ボリュームへのアクセスを物理ボリュームにマップするために

device-mapper の linear, stripe, mirror および snapshot の機能を利用している。
snapshot については数を増やした場合の性能的な問題が、mirror については
耐障害性の面での課題が、それぞれありコミュニティでの開発が続いている。

2.2. dmraid

dmraid は、実際の RAID 処理をソフトウェア側で行なうような安価な RAID デバイス
をサポートするためのプログラムである。ディスク上のメタデータを読み取り、
device-mapper を使って対応するブロックデバイスを作成する。
RAID 向けの device-mapper 機能としては、現状 stripe（RAID0）、mirror（RAID1）
が 2.6 カーネルに含まれるほか、RAID5 機能も開発途上にある。
実用的な役割を果たすには、障害発生を検出してメタデータを更新する必要があるが、
この機能はまだコミュニティでの開発途上である。

2.3. cryptsetup

cryptsetup は、ブロック単位での暗号化デバイスを作成するためのプログラムである。
LUKS 形式による鍵管理をサポートしており、デバイスの初期化や鍵の追加・削除など
を行なうことができる。

2.4. multipath-tools

multipath-tools は、Fibre channel 接続のディスクアレイなど、複数のパスを
もつディスク装置で、パス障害時の切り替えや負荷分散などのマルチパス機能を
有効にするためのプログラムである。
負荷分散機能については、従来の device-mapper では、I/O スケジューラより
上位の層でパスを振り分けるため、I/O 要求のマージやソートなどの効能を
実質的に無効化してしまう問題があった。
この問題は、2.6.31 カーネルで取り込まれる予定の request-based device-mapper
によって解消されており、Linux のブロックデバイスの動作についての一般的な説明
も含め次章で説明する。

3. Linux の storage stack と request-based device-mapper

Linux のブロックデバイスでは、以下のように I/O が処理される:

1）ファイルシステムが struct bio の形で I/O 要求発行
2）struct bio から reqeust 作成しキューに追加
3）I/O スケジューラがキュー中の request のソートとマージ
4）低レベルデバイスドライバが I/O スケジューラから request 受け取り処理

従来の device-mapper は、この階層の中で 2）に位置する。

通常、2）では、I/O 要求がデバイスの制約を違反していないかどうかのチェックや、
パーティションに対する I/O をディスクに対する I/O に再マップ
するなどの処理を行なった後、struct request に bio をまとめる。

従来の device-mapper では、2）で bio を struct request にまとめる処理に
代わって、bio の複製を作り、ブロック位置や I/O 発行先デバイスの変更、
バッファ内容の暗号化などの処理を行なって低位のデバイス（一般的なブロック
デバイスもしくは他の device-mapper デバイス）の 2）の処理に複製 bio を引き渡す。
I/O 発行先を選択した後で 3）のソートやマージの処理が行なわれるため、
マルチパスでの負荷分散のように I/O 発行先を I/O 要求単位で切り替えるケースでは、
本来であればマージできた I/O を細切れにしてしまう副作用がある。

request-based device-mapper は、上述の問題点を解決するために新規に追加された
機能で、3）と 4）の中間に位置し、マージ・ソート後に I/O スケジューラが
ピックアップした request に対して device-mapper の処理を行なう。