# drbd使いになろう

Satoru Yamashita <sat@sengawa.jp>

# Agenda

- drbd って何？
- drbd 動かしてみよう
- LVM を使おう
- heartbeat に drbd を任せよう

# Profile

- 山下 悟 (sat@sengawa.jp)
  - 福岡県北九州市出身
  - 東京に22年
  - 大阪在住 3 年目
- ネットワーク業務歴 19年
- UNIX歴 20年
  - 4.2BSD (VAX11/750) 〜
  - CentOS 歴 3年
- PHP歴 7年

# drbdとは？

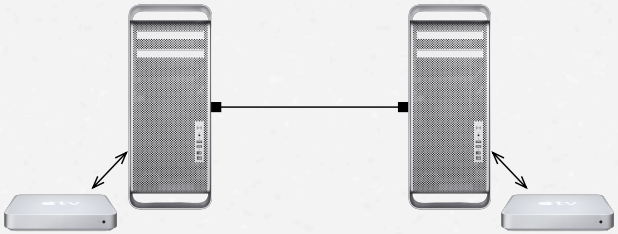
- Distributed Replicated Block Device の略
  - 日本語で言うと「分配されて複製されたブロックデバイス」。
  - DRBD って書くと DRDB に空目しやすい。
- ネットワーク越しに2台のディスクの同期を行う。
  - 商用版のDRBD Plus を使うと4台まで複製が可能。
- Block Device として動作するので、ファイルシステムを選ばない。
  - LVMでもZFSでもOCFSでも使える。

# よく聞く話

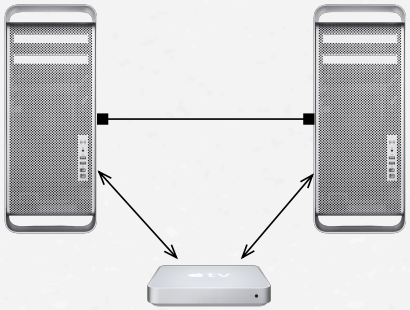
- drbd ってなんか不安定で。
  - 0.7時代は色々ありました。
  - 8系も最初の頃は色々発生しました。
- とらぶった時にどうしたらいいのか…
  - 検索しても使ってる人少ないから情報が無い。
  - 古い情報しかなくて役に立たない。

# ディスク冗長色々

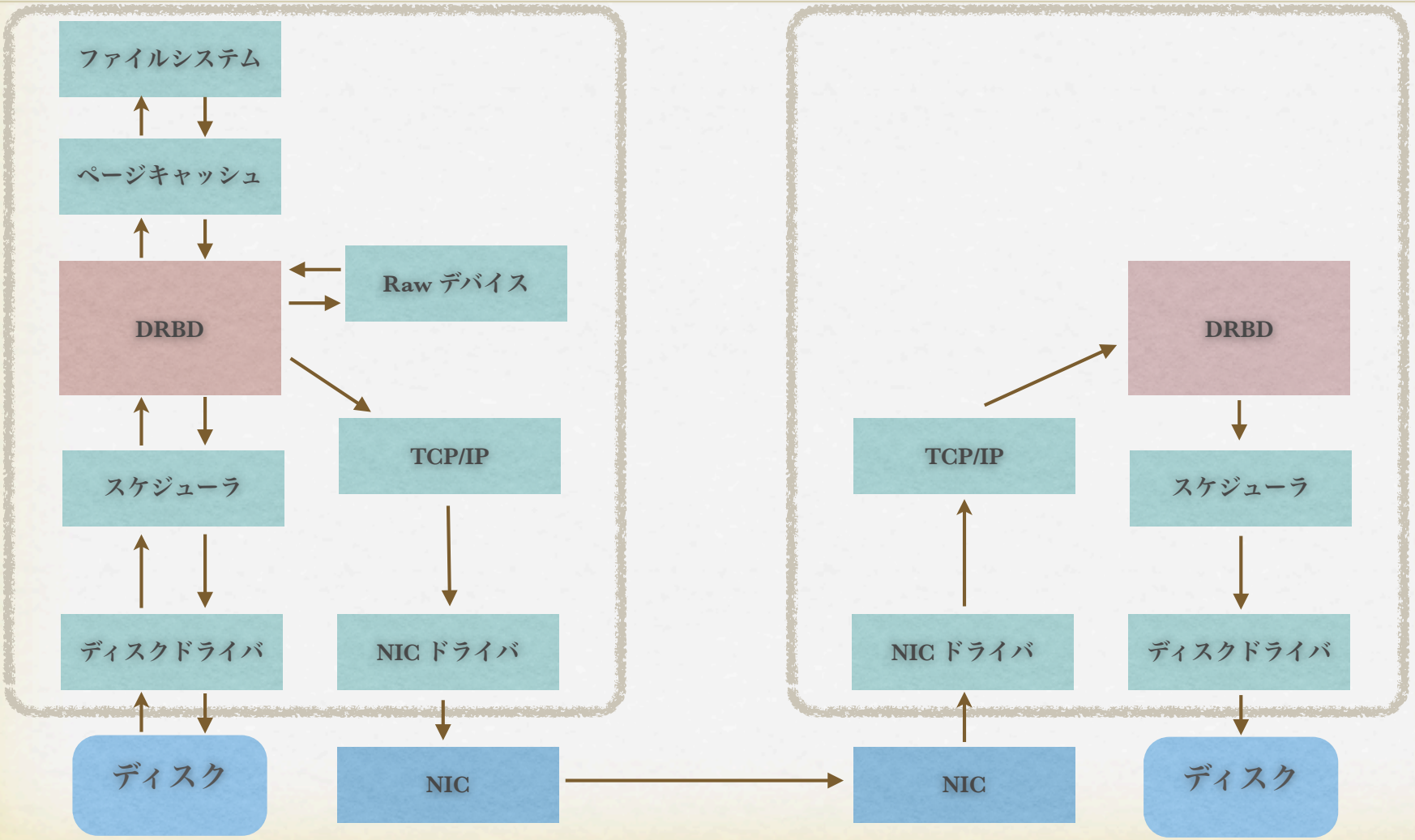
- ミラーディスク方式



- 共有ディスク方式



# drbdの仕組み(1)


ファイルシステム
ページキャッシュ
DRBD
Raw デバイス
スケジューラ
TCP/IP
ディスクドライバ
NIC ドライバ
ディスク
NIC
NIC
NIC ドライバ
TCP/IP
DRBD
スケジューラ
ディスクドライバ
ディスク

# drbdの仕組み(2)

- アプリケーションがファイルにデータを書き込む。
- ファイルシステムはメモリ上のページキャッシュにデータを送る。
- pdflush kernel thread がデータをディスクに書き出す。
  - ここでDRBDがデータを受け取る。
- DRBDはスケジューラにデータを送ると同時に、TCP/IP経由でSecondary DRBDに write を送る。
- Secondary DRBD はデータをスケジューラに送る。
- Primary DRBD は書き込み完了を上位レイヤに通知する。

# インストールの前に

- メタデータ領域(128MB固定)が必要。
- 事前にmkfsされたdeviceには新規メタデータが作成出来ない。
  - 外部meta-diskを利用する
  - dd で破壊する

# インストール

- DRBDはkernel moduleとユーティリティプログラムで構成されている。
- binary package は２つ。(CentOS の場合)
  - drbd82-#.#.#-*.centos.i386.rpm
  - kmod-drbd82-#.#.#-*.i686.rpm
  - yum で install 可能
- package がない場合は source から make。
  - http://oss.linbit.com/drbd/
  - kernel の source が必要。

# 設定の流れ

- drbd を install する。
- drbd に使用する device を決める。
- drbd の同期に使用するI/Fを決める。
- config をがしがし書く。
- meta-disk 領域を作成する。
- drbd を起動。
- primary 側を決めて、同期開始。
- ファイルシステムを作成。

# config(1)

- config の例

```
# /etc/drbd.conf
global {
        usage-count no;
}

common {
        syncer { rate 30M; }
}

resource r0 {
        protocol C;
        startup {
                wfc-timeout 120;
        }
        disk {
                on-io-error pass_on;
        }
        on drbd0.example.jp {
                device /dev/drbd0;
                disk /dev/vg01/lv02;
                address 192.168.0.1:7788;
                meta-disk internal;
        }
        on drbd1.example.jp {
                device /dev/drbd0;
                disk /dev/vg01/lv02;
                address 192.168.0.2:7788;
                meta-disk internal;
        }
}
```

# config(2)

- config は複数ブロックで構成
  - global ブロック
    - global な設定パラメータ
  - common ブロック
    - 全ての resource に共通名設定
  - resource ブロック
    - レプリケーションデータ領域ごとの設定
    - handlers, startup, disk, net, syncer, on のブロックはこの中に書く

# config(3)

- global ブロック
  - usage-count [ask|yes|no]
    - 開発元の利用カウントに情報を送信するかどうかの指定。
    - default は yes
    - 普通は no にしておく
- common ブロック
  - syncer ブロック
    - 通信速度の制限
    - rate 通信速度 (Bytes/sec)

# config(4)

- resource ブロック
  - protocol [A|B|C]
    - A: データを local TCP send buffer に送った時点
    - B: データを local disk と remote TCP に送った時点
    - C: データを local disk と remote disk に送った時点
    - 通常はCを使いパフォーマンスに問題があるようだったらBを使う
    - Aは遠隔地用。
  - startup ブロック
    - drbd 起動時の挙動を指定
    - wfc-timeout 秒
      - 対向 drbd と接続が出来るまでの待ち時間
      - デフォルトは 0 (無期限)
      - 設定しない片側だけで boot 出来ない事態になる

# config(5)

- resource ブロック(続き)
  - disk ブロック
    - on-io-error detach|pass_on|call-local-io-error
      - detach: エラーを起こしたデバイスを切り離して、ディスクレスモードで動作
      - pass_on: 上位レイヤにエラーをそのまま渡す
      - call-local-io-error: local-io-error ハンドラに指定したスクリプトを実行する

# config(6)

- resource ブロック(続き)
  - on ブロック
    - on ホスト名
      - ホスト名は hostname と一致させると楽
    - device デバイス名
    - disk デバイス名
      - 実際に使用するデバイス名
    - address IPアドレス:port(TCP)
      - TCP portは resouce 毎にユニーク
    - meta-disk メタディスク指定
      - internal
      - パーティション指定 (外部 meta-disk)

# 初期動作

- drbd を起動する前にmetaデータ領域を作成する

  # drbdadm create-md リソース名

- drbdを起動する (両方で)

  # /etc/init.d/drbd start

- drbd の状態を確認する

  # cat /proc/drbd
  # /etc/init.d/drbd status

- 初期同期を実行する (primary 側のみ)

  # drbdadm -- --overwrite-data-of-peer primary リソース名

- ファイルシステムを作成する

  # mke2fs -j DRBDデバイス名 (例: /dev/drbd0)

# drbdの状態

- 初期同期前

```
0: cs:Connected st:Secondary/Secondary ds:Inconsistent/Inconsistent C r---
```

- 初期同期実行直後

```
0: cs:Connected st:Secondary/Secondary ds:UpToDate/Inconsistent C r---
    [>.................] sync'ed:    1.0%  (102132/ 1048026)K
```

- 初期同期完了後

```
0: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
```

# 手動フェールオーバー

- 手動で切り替えるには
  - mount してたら umount する
  - status を secondary に変える
  - (反対側で) primary にする
  - mount する

    [例]

    p:# umount /drbd0

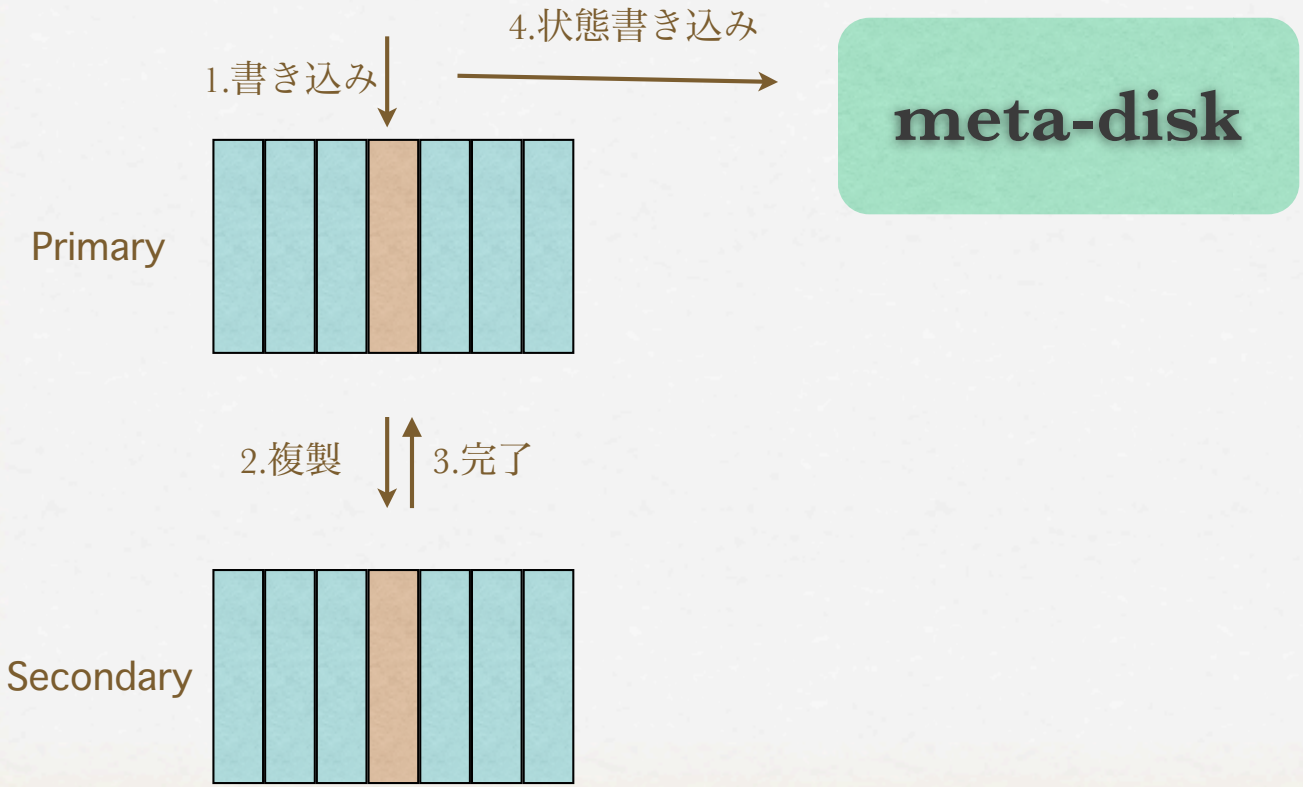
    p:# drbdadm secondary r0

    s:# drbdadm primary r0

    s:# mount /devd0

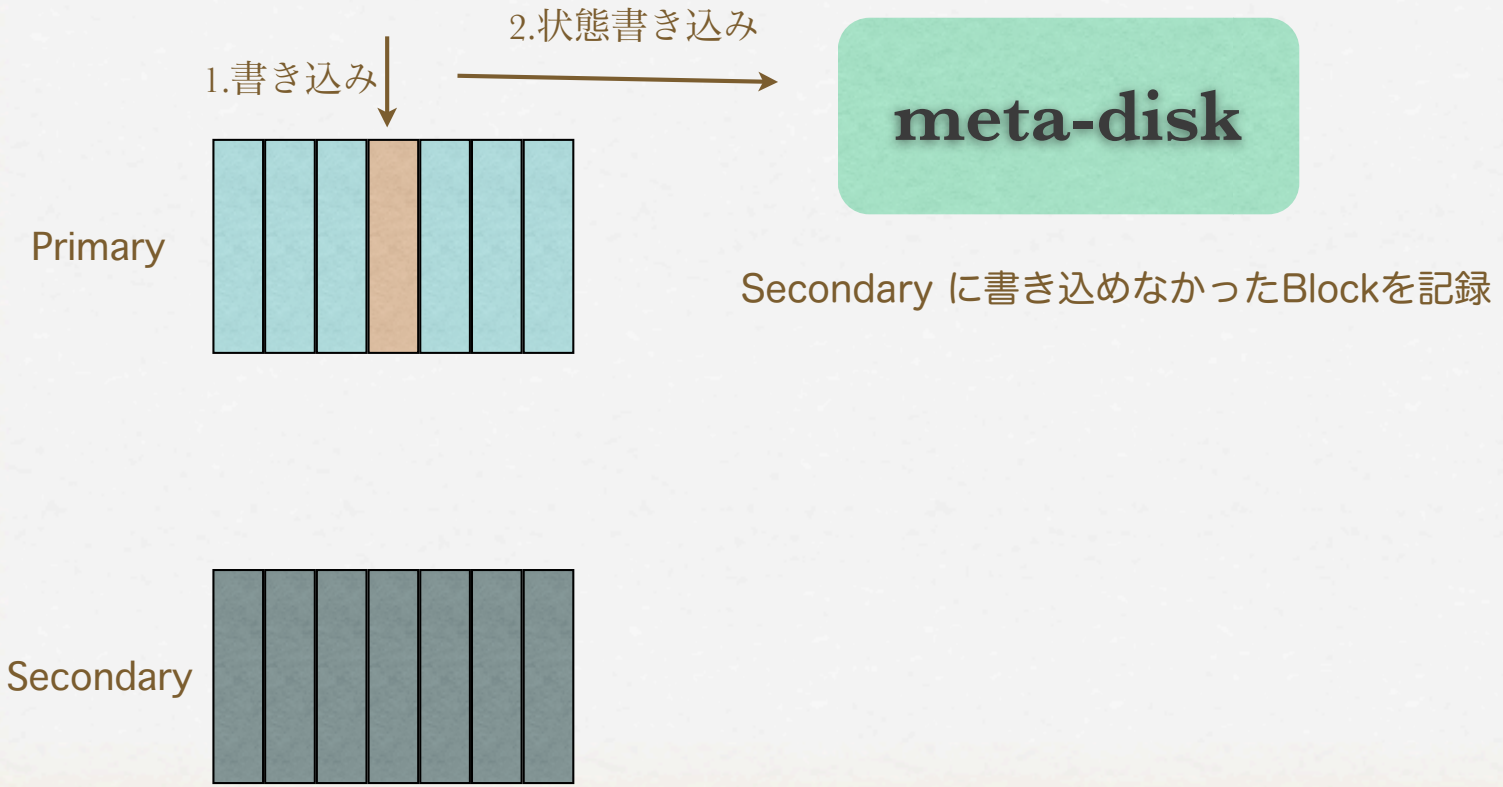
# メタデータ(1)

- 正常時



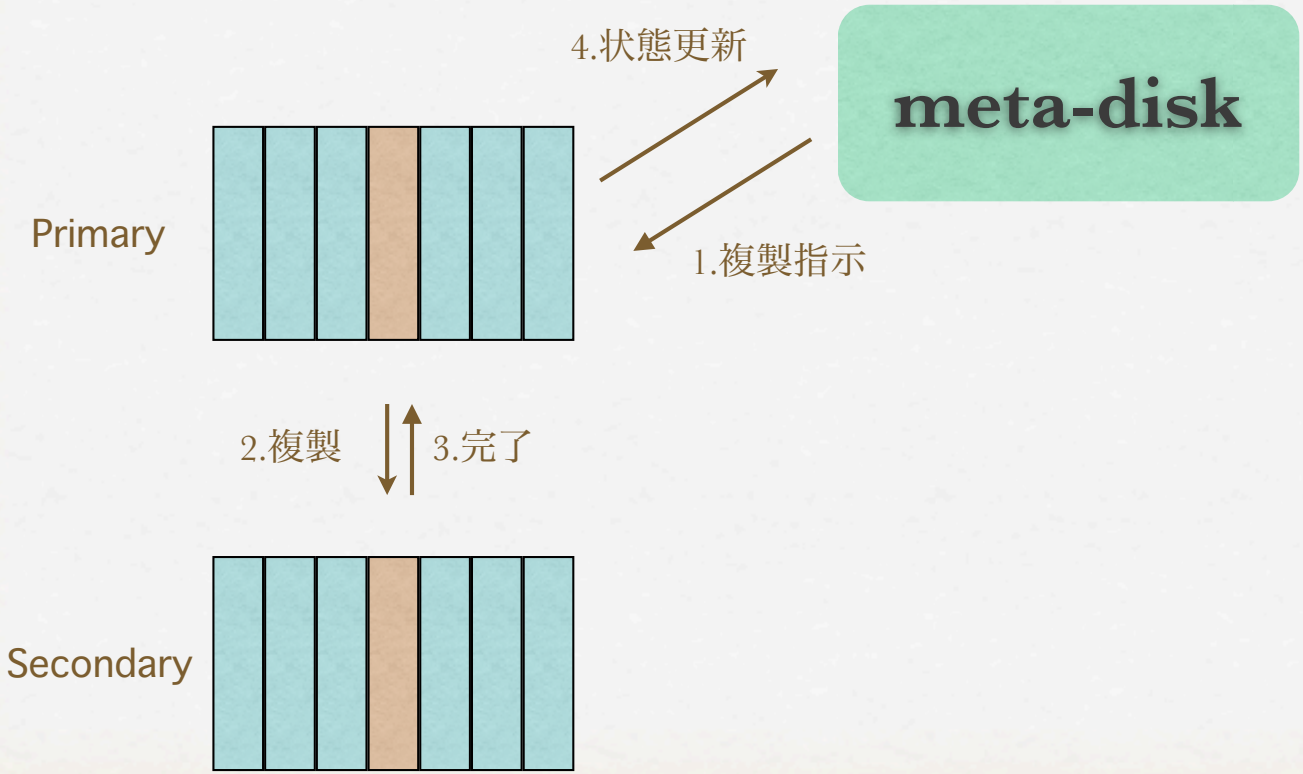
# メタデータ(2)

- Secondary down 時



# メタデータ(3)

- Secondary復旧時



# LVM(1)

- Linux Volume Manager の略
- 複数のディスク(PV)をグループ化してVolume Group (VG)を構成出来る
- VGから任意の大きさのLogical Volume(LV)を作成出来る。
- PV、VG、LVは追加、削除、サイズ変更が出来る。
- 単一ドライブの容量を超えるデータ領域を実現出来る。
- LVの仮想的なコピー(SnapShot)が作成出来る。

# LVM(2)

- 物理ボリューム(PV)の登録
  - LVMに登録するには、パーティションIDを8Eにする必要がある。(fdisk とかで)
- VGの作成
  - PVを束ねてVGを作る。(PV1個でもOK)
  - /dev/VG名 が作られるが、LV作らないと現れない。
- LVの作成
  - 実際にmountするVolume。
  - 作成時にサイズの指定が必要。
  - VGを使い切らないようなサイズ管理を行った方が、後々便利。

# LVM(3)

- 作成例

```
# fdisk /dev/sdb
```

2GB のエリアを想定

パーティションIDを8Eにしてwriteする

```
# pvcreate /dev/sdb

# vgcreate vg01 /dev/sdb

# lvcreate -L 1G -n lv00 /dev/vg01
```

# LVMでSnapShot(1)

- drbd は Secondary 状態では mount 出来ない
- でも実diskにはデータが書き込まれる
- LVM だとスナップショットがとれる
- しかもスナップショットデバイスは mount ＯＫ(←重要)
- Secondary 側である時点の完全な Backup が取れる
- ただし、書き込み速度が大幅に低下するので、高速書き込み必要な場合は要注意。

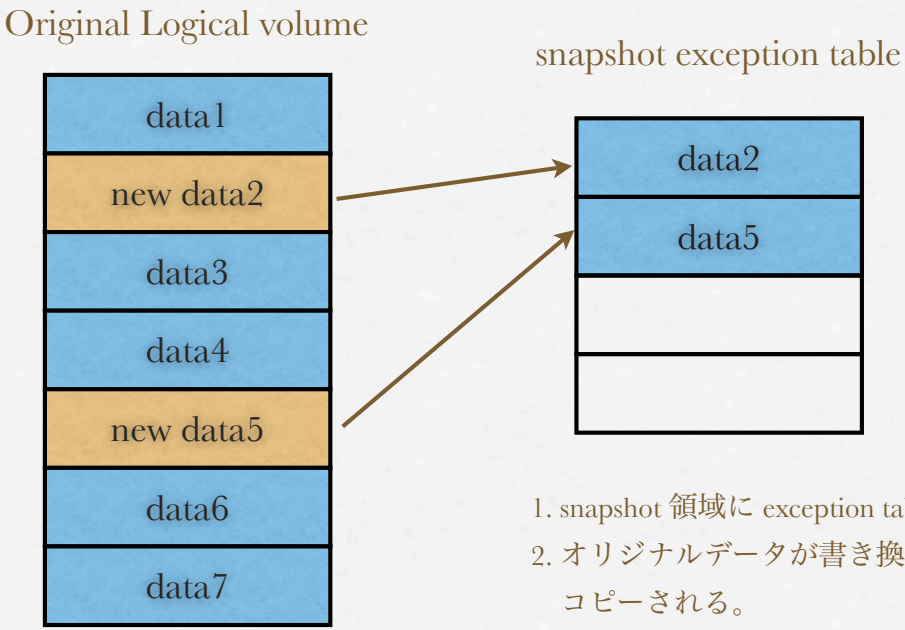
# LVMでSnapShot(2)

- オペレーション例

```
# lvcreate -s -L 512M -n lvsnap0 /dev/vg01/lv00
  Logical volume "lvsnap0" created
# dd if=/dev/vg01/lvsnap0 of=/var/backup/lv00-20090516.img
# lvremote -f /dev/vg01/lvsnap0
  Logical volume "lvsnap0" successfully removed
```

# LVMでSnapShot(3)

## SnapShotの動作イメージ




1. snapshot 領域に exception table が作られる。初期状態は空っぽ。
2. オリジナルデータが書き換えられると、位置とオリジナルデータが table にコピーされる。
3. LVM はオリジナル論理ボリュームと table の情報を組み合わせて、snapshot 作成時点の論理ボリュームを提示する。

# heartbeatとは？

- HAシステムを構築する為の便利なソフト
- 2台のシステム間で相互監視しながら動く(V1)
  - V2形式だと3台以上の構成が可能
- 相手が一定時間以上返事しないとフェールオーバー動作を開始する。
- haresources に記述されている resource を stop/start させることによりフェールオーバー動作を行う。
- DRBD用の script が用意されているので便利

# heartbeat設定(1)

- /etc/ha.d/authkeys
  - ホスト間の認証に使用する。
  - permission が 0600 じゃないと駄目
- /etc/ha.d/ha.cf
  - heartbeat の動作パラメータの設定
  - ほとんど default で問題ない
- /etc/ha.d/haresource
  - サービスするリソースを記述する

# heartbeat設定(2)

## 設定例

```
# cat  /etc/ha.d/authkey
auth 1
1 sha1 public
```

```
# cat /etc/ha.d/ha.cf
logfile         /var/log/ha.log

keepalive       2
deadtime        30
warntime        10
initdead        120

udpport         694

ucast           eth0 172.16.63.12
node            drbd-demo1.sengawa.jp drbd-demo2.sengawa.jp

auto_failback   on
```

```
#cat /etc/ha.d/haresources
drbd-demo1.sengawa.jp 172.16.63.10 drbddisk::r0 Filesystem::/dev/drbd0::/drbd0::ext3
```

# フェイルオーバー

- 以下の場合にフェイルオーバーが発生する
  - 対向のheartbeatとdeadtimeで指定した時間通信が途絶えた場合。
  - 対向のマシンでhb_takeoverが実行された時
  - 自分がhb_standbyを実行した時

# 事例(1)

- web server
  - ただのContents Serverにのみ使用。
  - フェイルオーバー時にhttpdが再起動するのでsessionは引き継がれない。

# 事例(2)

- dhcp server
  - config と lease data を drbd 上に置くと、ほぼ断無しでフェイルオーバーする。
  - ただしVIP使う関係でclientとの間に(高めの)FireWallが存在するとはまる。

# 質問他

- 何かございましたら、お気軽にお聞きください。
  - E-mail: sat@sengawa.jp
  - IRC: sat
  - mixi: http://mixi.jp/show_profile.pl?id=9698
- **長い間、ご静聴ありがとうございました。**