File System EXT4 & Lỗi tham số VOLUME (Hỏng SUPERBLOCK)

1 Giới Thiệu Vấn Đề

Hệ thống file Ext4 là nền tảng lưu trữ hiện đại nhất trong dòng Extended Filesystem của Linux, kế thừa và mở rộng từ Ext2/3. Đặc trưng bởi khả năng xử lý volume lớn, các tính năng đảm bảo nhất quán dữ liệu, và hỗ trợ journaling. Một điểm yếu phổ biến của các hệ thống file là nguy cơ hư hại các thông số cấu trúc định danh, mà điển hình là Superblock – vùng metadata then chốt quyết định toàn bộ truy xuất dữ liệu.

Lý thuyết phục hồi siêu dữ liệu (metadata recovery) dựa trên nhận định: hệ điều hành chỉ cần Superblock nguyên vẹn để truy cập dữ liệu, còn các data block phía sau vẫn còn khả năng bảo toàn kể cả khi quá trình mount thất bại hoàn toàn. Do đó, phân tích cấu trúc logic của Superblock và hiểu rõ chiến lược backup của Ext4 sẽ là nền tảng cho mọi kỹ thuật phục hồi chuyên sâu.

2 Cơ Sở Lý Thuyết

2.1 Kiến Trúc Block Group – Cốt Lõi của Ext4

Khác với hệ thống tập trung, Ext4 chia volume thành các Block Group liền kề, mỗi group như một hệ thống file thu nhỏ, mục tiêu là tối ưu hóa hiệu suất I/O và giảm phân mảnh vật lý. Việc này dựa trên nguyên lý locality: các dữ liệu liên quan sẽ được lưu trữ cùng nhau, giảm số lần truy cập tới đầu đọc/ghi (trên ổ cứng truyền thống).

Thành phần của một Block Group:

- Inode Table: Bảng chỉ mục file, lưu metadata, quyền truy cập, vị trí các block dữ liệu.
- Data Blocks: Vùng lưu trữ dữ liệu thật của file.
- Bitmaps: Bản đồ trạng thái "bận" hay "rảnh" cho inode/block, cho phép hệ thống quản lý và cấp phát bộ nhớ hiệu quả.
- Group Descriptors: Metadata chỉ định vị trí, trạng thái của từng Block Group.
- Trong Group đầu tiên (Block Group 0), chứa thêm Superblock chính.

Lý luận hiệu năng: Việc lưu Inode cùng Data Block của file vào cùng group giúp giảm seek time, nhất là với ổ cứng cơ (non-SSD). Ngoài ra, cách chia này còn giúp hệ thống "hồi phục cục bộ" các phần nhỏ hơn khi xảy ra lỗi.

Công thức tính toán Block Group cho ext4:

Giả sử block size là bs (bytes), mỗi Block Group thường chứa:

$$\#\text{BlockGroupSize} = bs \times \text{BlocksPerGroup}$$

Trong đó, với block size 4096 bytes và BlocksPerGroup là 32,768 (default), sẽ ra khoảng 128 MB/ group. Như vậy mỗi volume sẽ gồm nhiều group, backup Superblock được rải đều trong các group đó.

2.2 Superblock – Vai Trò, Cấu Trúc và Cơ Chế Dự Phòng

a) Vị Trí & Ý Nghĩa:

Superblock chính luôn nằm tại offset 1024 bytes trong Block Group 0, không chịu ảnh hưởng bởi boot sector. Đây là vùng gốc quản lý các thông số toàn cục: loại hệ thống file, block size, tổng số block/inode, trạng thái journal, v.v.

b) Cấu Trúc Logic của Superblock:

Một số trường thông tin tiêu biểu và logic kiểm tra:

Tham số	**Kích thước**	**Ý nghĩa**
s_magic	2 bytes	Magic number (0xEF53), nhận diện ext4
s_block_size	4 bytes	Kích thước mỗi block (theo lũy thừa)
s_blocks_count	4 or 8 bytes	Tổng số block toàn volume
s_inodes_count	4 or 8 bytes	Tổng số inode toàn volume
s_blocks_per_group	4 bytes	Số block/trong một group
s_inodes_per_group	4 bytes	Số inode/trong một group
s_journal_inum	4 bytes	Vị trí inode của journal file

Toán học block size: Ext4 tính block size logic bằng:

$$\text{BlockSize} = 1024 \times 2^{s\_log\_block\_size}$$

Với hầu hết hệ thống: $s\_log\_block\_size = 2$ nghĩa là block size = 4096 bytes.

c) Quá trình mount – phân tích logic:

Quy trình nhận diện file hệ thống:

1. Hệ điều hành đọc offset 1024 → lấy Superblock.
2. Kiểm tra magic number (0xEF53) và các trường phải có giá trị hợp lệ (block size $\neq 0$, số inode > 0).
3. Nếu hợp lệ, hệ thống mount volume. Nếu không, báo lỗi "bad superblock".

Như vậy, sai lệch tại Superblock hầu như làm tê liệt toàn bộ hệ thống file – dữ liệu không mất nhưng không thể truy cập do thiếu thông tin định dạng.

2.3 Cơ Chế Backup Superblock – Xóa Đi Điểm Lỗi Đơn Nhất

Bài toán SPOF (Single Point of Failure): Superblock là điểm lỗi đơn nhất trong cấu trúc truyền thống. Nếu bị ghi đè hoặc hỏng vật lý, mọi truy xuất đều thất bại; dù thực tế data block vẫn còn nguyên.

Giải pháp ext4 – dự phòng phân tán: Ext4 tự động tạo nhiều bản backup Superblock tại các Block Group có chỉ số lẻ hoặc bội số 3 (ví dụ: 1, 3, 5, 7...). Các vị trí backup này tuân theo công thức:

$$\text{BackupBlock Idx} = n \times \text{BlocksPerGroup}$$

Với block size 4096 bytes, BlocksPerGroup = 32,768, nên các backup rải ở block: 32,768 (Group 1), 98,304 (Group 3), 163,840 (Group 5), ...

Đặc tính phục hồi: Các bản backup là độc lập và hoàn toàn đầy đủ. Việc phục hồi chỉ cần "sao chép" backup thành Superblock chính. Do backup được lưu ở nhiều nơi, nguy cơ mất đồng thời là cực kỳ thấp, trừ khi lỗi vật lý diện rộng.

Ý nghĩa thực tiễn:

- Khi mount lỗi, hoặc ext4 phát hiện bất hợp lệ ở Superblock, có thể sử dụng backup như nguồn dữ liệu thay thế.
- Tiện ích `e2fsck` hỗ trợ nạp Superblock backup bằng tham số `-b [block_number]`, minh chứng khả năng phục hồi hệ thống file nhanh chóng, miễn là backup còn nguyên vẹn.

3 Quá Trình Hư Hại & Khả Năng Phục Hồi

Bản chất của vấn đề không chỉ nằm ở việc một vùng metadata bị ghi đè, mà là mối quan hệ chặt chẽ giữa siêu dữ liệu định danh và dữ liệu thực tế. Lý thuyết phục hồi nhấn mạnh:

- Một hệ thống file được thiết kế tốt phải giảm tối đa các SPOF.
- Việc rải backup siêu dữ liệu vào các vùng khác nhau sẽ tăng khả năng phục hồi mà không cần can thiệp vào data block.

Phân tích các bước thất bại của mount/e2fsck/mke2fs cho thấy: khi siêu dữ liệu gốc bị mất, quá trình tự động nhận diện và dò tìm backup (thông qua thông số block size, cấu trúc) bị "vòng lặp Catch-22" – tức là phần mềm phục hồi cần siêu dữ liệu chính để xác định vị trí backup, nhưng siêu dữ liệu chính đã hỏng. Việc này lý giải tại sao ext4 phải dùng danh sách backup thiết lập cứng (theo chuẩn).

4 Hiện Thực Hóa Giải Pháp & Phân Tích Thực Nghiệm

4.1 Phương Pháp Luận Thực Nghiệm

Để kiểm chứng cơ sở lý thuyết đã phân tích tại Mục 2 và 3, một kịch bản thực nghiệm đã được xây dựng và tự động hóa. Mục tiêu của thực nghiệm là mô phỏng chính xác tình huống phá hủy Superblock chính và kiểm tra hiệu quả của giải pháp phục hồi tự động đã được phát triển.

Quy trình kiểm thử được thiết kế để bao quát 4 giai đoạn cốt lõi, sử dụng một kịch bản Shell (`test_recovery.sh`) để điều phối và một kịch bản Python (`recover_ext4.py`) làm giải pháp phục hồi chính:

- **Giai đoạn 1 - Khởi tạo:**

```
[0/6] Đang dọn dẹp môi trường cũ (nếu có)...
$ umount /mnt/recovery_test || true
umount: /mnt/recovery_test: not mounted.
$ rm -f test_image_1gb.img
$ mkdir -p /mnt/recovery_test

[1/6] Đang tạo file ảnh 1GB và format ext4...
$ dd if=/dev/zero of=test_image_1gb.img bs=1M count=1024 status=none
$ mke2fs -t ext4 -F test_image_1gb.img > /dev/null 2>&1

[2/6] Đang thêm dữ liệu kiểm thử ('test_data.txt')...
$ mount -o loop test_image_1gb.img /mnt/recovery_test
$ echo "DU LIEU NAY PHAI DUOC PHUC HOI" > /mnt/recovery_test/test_data.txt
$ ls -l /mnt/recovery_test
total 20
drwx------ 2 root root 16384 Nov 17 07:52 lost+found
-rw-r--r-- 1 root root    31 Nov 17 07:52 test_data.txt
$ umount /mnt/recovery_test
```

- Tạo file ảnh dung lượng 1GB sử dụng `dd`.
- Định dạng file ảnh với hệ thống file ext4 bằng `mke2fs`.
- Mount volume nhằm tạo file dữ liệu `test_data.txt` để kiểm chứng tính toàn vẹn sau phục hồi.

- **Giai đoạn 2 - Phá hoại:**

```
[3/6] Đang LÀM HỎNG Superblock chính (offset 1024)...
$ dd if=/dev/zero of=test_image_1gb.img bs=1 count=100 seek=1024 conv=notrunc status=none
```

- Mô phỏng lỗi "sai tham số volume" bằng cách ghi đè 100 bytes dữ liệu rỗng (số 0) trực tiếp lên vị trí offset 1024 (vị trí của Superblock chính):

  `dd if=/dev/zero of=[image_file] bs=1 count=100 seek=1024 conv=notrunc`

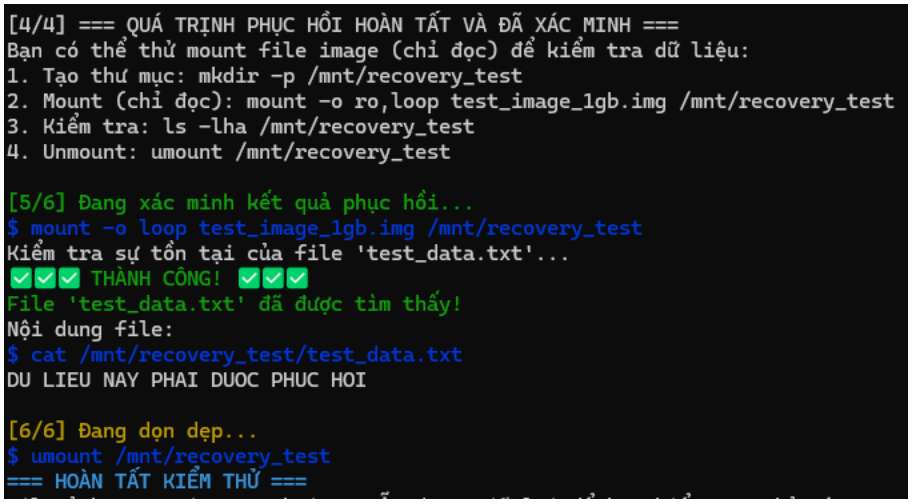
- **Giai đoạn 3 - Phục hồi:**

```
[4/6] Đang chạy script phục hồi 'recover_ext4.py'...
$ python3 recover_ext4.py test_image_1gb.img
--- Trình Phục Hồi Superblock EXT4 ---
[INFO] Mục tiêu phục hồi: test_image_1gb.img

[1/4] Đang thử tìm backup superblock tự động (Kế hoạch A)...
[INFO] Kế hoạch A thất bại (có thể do superblock chính đã hỏng).
[INFO] Chuyển sang Kế hoạch B: Sử dụng danh sách backup tiêu chuẩn.
[INFO] Danh sách sẽ thử: [32768, 98304, 163840, 229376, 294912]

[2/4] Bắt đầu quá trình sửa chữa...

--- Đang thử với backup block: 32768 ---
[INFO] Đang thực thi: e2fsck -f -y -b 32768 test_image_1gb.img
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
Block bitmap differences:  +(98304--98432) +(163840--163968)
Fix? yes

Free blocks count wrong for group #1 (32639, counted=32638).
Fix? yes

Free blocks count wrong (249189, counted=249188).
Fix? yes

Free inodes count wrong for group #0 (8181, counted=8180).
Fix? yes

Free inodes count wrong (65525, counted=65524).
Fix? yes

Padding at end of inode bitmap is not set. Fix? yes


test_image_1gb.img: ***** FILE SYSTEM WAS MODIFIED *****
test_image_1gb.img: 12/65536 files (0.0% non-contiguous), 12956/262144 blocks

[STDERR]:
e2fsck 1.45.5 (07-Jan-2020)
```

```
[THÀNH CÔNG] Sửa chữa thành công sử dụng block 32768 (mã lỗi: 1).
Superblock chính đã được phục hồi.

[3/4] Đang chạy kiểm tra lần cuối để xác minh (sử dụng superblock chính)...
[INFO] Đang thực thi: e2fsck -f -y test_image_1gb.img

--- Kết quả xác minh ---
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
test_image_1gb.img: 12/65536 files (0.0% non-contiguous), 12956/262144 blocks

[STDERR]:
e2fsck 1.45.5 (07-Jan-2020)

[THÀNH CÔNG] Xác minh hoàn tất (mã lỗi: 0). Hệ thống file đã ổn định.

[4/4] === QUÁ TRÌNH PHỤC HỒI HOÀN TẤT VÀ ĐÃ XÁC MINH ===
Bạn có thể thử mount file image (chỉ đọc) để kiểm tra dữ liệu:
1. Tạo thư mục: mkdir -p /mnt/recovery_test
2. Mount (chỉ đọc): mount -o ro,loop test_image_1gb.img /mnt/recovery_test
3. Kiểm tra: ls -lha /mnt/recovery_test
4. Unmount: umount /mnt/recovery_test
```

  - Thực thi kịch bản `recover_ext4.py` trên file ảnh đã bị lỗi.

- **Giai đoạn 4 - Kiểm chứng:**



  - Mount lại file ảnh sau phục hồi.
  - Kiểm tra sự tồn tại và xem nội dung của `test_data.txt`.

4.2 Phân Tích Kịch Bản Phục Hồi (`recover_ext4.py`)

Giải pháp hiện thực hóa không phải là một lệnh đơn lẻ mà là một "bộ điều phối" thông minh, nhằm hóa giải vòng lặp Catch-22:

- **Kế hoạch A (Tự động):** Cố gắng dùng `mke2fs -n` để dò tìm danh sách backup. Trong thực nghiệm, bước này thất bại vì `mke2fs` cần đọc Superblock chính để xác định block size – vốn đã bị hỏng.
- **Kế hoạch B (Fallback):** Khi kế hoạch A thất bại, kịch bản chuyển sang kế hoạch B, lặp qua một danh sách backup dự phòng tiêu chuẩn như: `[32768, 98304, 163840, ...]` áp dụng kiến thức chuyên môn về chuẩn ext4 nhằm phá vỡ vòng lặp.
- Với mỗi block backup, kịch bản chạy `e2fsck -f -y -b [block]` cho đến khi một lần nhỏ nhất trả về mã lỗi thành công (0 hoặc 1), sau đó vòng lặp dừng.

4.3 Phân Tích Kết Quả Thực Nghiệm

Quá trình thực thi `test_recovery.sh` (gồm `recover_ext4.py`) cho ra kết quả như sau:

- **Kế hoạch A thất bại**: Ngay đầu, kế hoạch tự động dò backup thất bại, xác nhận lý thuyết.
- **Kế hoạch B kích hoạt**: Kịch bản sang thử backup đầu tiên tại block 32768.
- **Quá trình sửa chữa:** `e2fsck` phát hiện nhiều lỗi metadata (ví dụ: block bitmap khác biệt, free blocks count sai) và với cờ `-y` đã tự động sửa chữa. Quan trọng nhất, `e2fsck` đã dùng bản backup 32768 để sửa đè Superblock chính.
- **Mã lỗi 1**: `e2fsck` thoát với mã lỗi 1, nghĩa là lỗi đã được phát hiện và chữa thành công; kịch bản ghi nhận thành công và dừng.

- **Xác minh**: Kịch bản tiếp tục chạy `e2fsck` không dùng cờ `-b`, trả về mã 0 (không lỗi) xác nhận Superblock chính mới hợp lệ.
- **Toàn vẹn dữ liệu**: Volume mount thành công, file `test_data.txt` vẫn tồn tại nguyên vẹn.

4.4 Kết Luận Thực Nghiệm

Mặc dù Superblock chính bị phá hủy hoàn toàn, cơ chế dự phòng metadata (backup Superblock) của ext4, khi kết hợp cùng logic phục hồi thông minh có thể vượt qua vòng lặp Catch-22, cho phép khôi phục truy cập volume hoàn toàn mà không làm mất dữ liệu người dùng.

Tài Liệu Tham Khảo

- Linux Kernel - Ext4 Filesystem Documentation
- e2fsck - ext2/ext3/ext4 Filesystem Checker Manual.
- Ext4 Filesystem Specifications - Design and Implementation.
- Red Hat Enterprise Linux - Filesystem Administration Guide.