GlusterFS in the Kernel

서버 수정 없이 FUSE를 커널로

기존 프로토콜을 유지하면서 클라이언트의 클러스터 판단과 VFS 통합을
커널 모듈로 옮길 수 있는가

제11회 한국 리눅스 커널 개발자 모임

2026년 9월 17일

김지현

potatogim@potatogim.net github.com/potatogim

발표자 소개

스토리지 관리 엔진을 만들며 이거저거 손대 온 잡부

김지현 글루시스/연구원 나부랭이

대한민국의 스토리지 전문 기업 글루시스에 근무 중인 개발자인지 엔지니어인지 뭐 하는 녀석인지 모를 생명체
사람의 모습을 하고 있으나 항간에는 그 실체가 변이종 감자고 신체 대부분이 녹말이라는 소문도

기업용 스토리지 관리 솔루션인 AnyStor(AnyStor-E)를 개발, 스토리지 관리 엔진 설계와 구현
분산 파일시스템, 블록, 백업을 한 엔진에서 다룸
관심 분야는 분산 파일시스템(GlusterFS, Lustre, DAOS), CXL 메모리, 저장 장치와 커널 I/O 경로

최근 활동 프로젝트

versity/versitygw: S3 게이트웨이에 RDMA RC 데이터 경로(hipobj-rc-v2)와 운영 제어, 진단 추가
ROCm/hipObject: RC 전송 프로토콜 결함 재현과 수정, hipobj-rc-v2 세션 프로토콜 제안과 구현
ROCm/rocm-ernic: RDMA NIC 에뮬레이터의 송신 오류 경로와 WRITE_WITH_IMM 루프백 수정
ScaleX-IO/uGDS: GPU Direct Storage 커널 드라이버에 AMD ROCm dma-buf 백엔드, 핫리무브 경쟁 수정
hpc/ior: AMD hipFile GPU Direct Storage 백엔드 추가, cuFile과 GPU 런타임 추상화
gluster/glusterfs: rpc의 RDMA QP 해제 경로 수정, aarch64 EC NEON SIMD 감지 수정
gluster/libgfapi-perl: Perl용 libgfapi 바인딩 개발과 릴리스

질문과 반례는 언제든 환영, 발표 뒤에도 이어서 논의

potatogim@potatogim.net github.com/potatogim

GLUESYS

목차

오늘 이야기의 순서

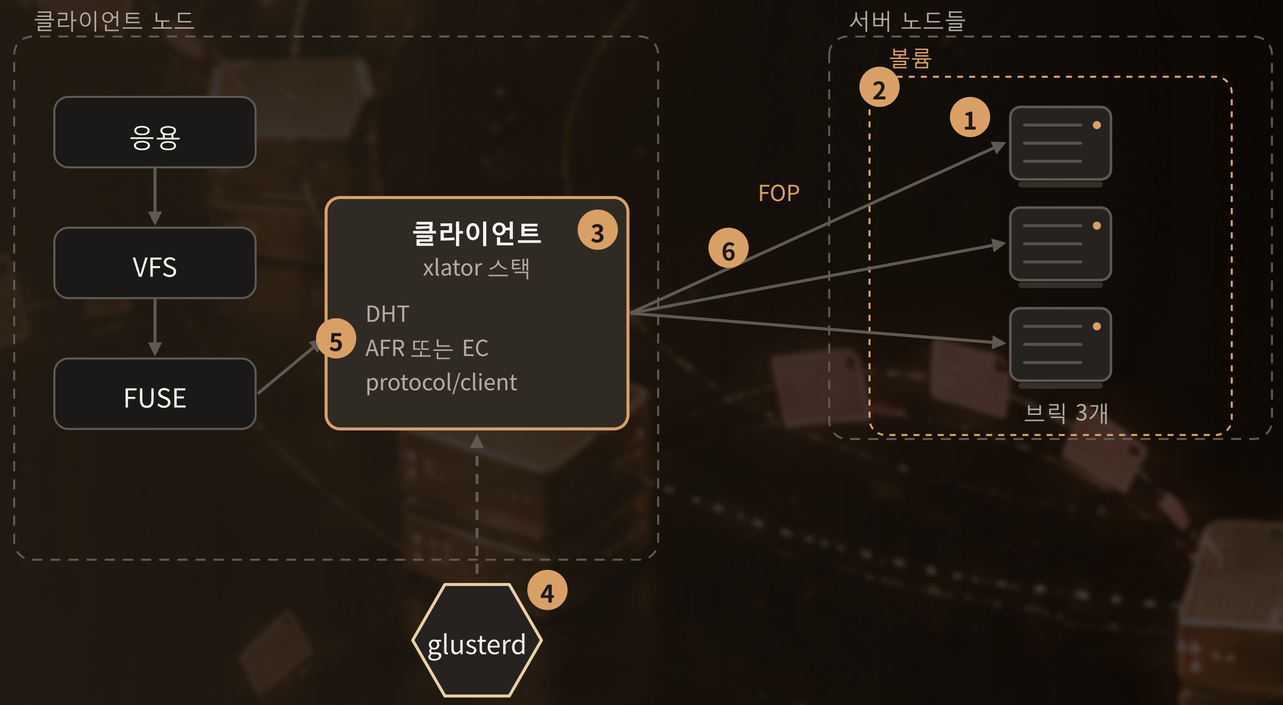
GlusterFS 빠른 개요

구성 요소와 DHT, AFR, EC의 역할

구성 요소
데이터 경로와 제어 경로
메타데이터 서버와 배치 책임으로 본 이웃들
볼륨 종류와 그래프
volfile 배포와 서버 그래프
DHT, AFR, EC의 역할
DHT 배치
AFR 복제
EC 소거 코드

구성 요소

여섯 가지 구성 요소가 그림의 어디에 있는지



1. **브릭(brick)**
   데이터를 저장하는 서버 프로세스와 그 디렉터리
2. **볼륨(volume)**
   브릭 묶음에 붙이는 이름과 정책 단위
3. **클라이언트(client)**
   파일시스템을 마운트한 주체, 기존에는 FUSE 데몬
4. **glusterd**
   볼륨 생성과 변경을 관리하는 제어 데몬
5. **xlator**
   그래프를 이루는 변환기 (배치, 복제, 소거, 프로토콜)
6. **FOP**
   lookup, write, setattr 같은 파일 연산 하나

핵심 전용 메타데이터 서버가 없고, 배치와 복제 판단은 클라이언트 쪽 xlator 스택에 있음

데이터 경로와 제어 경로

같은 그림 위의 두 경로, I/O에 glusterd는 끼지 않음

데이터 경로

응용의 파일 연산이 VFS와 FUSE를 지나 클라이언트에 도달
클라이언트의 xlator 스택이 배치와 복제를 판단
FOP를 RPC로 만들어 브릭에 직접 전송
응답도 브릭에서 클라이언트로 직접

제어 경로

glusterd는 볼륨 구성을 volfile로 만들어 배포
마운트 시 한 번, 구성이 바뀔 때 다시
I/O 경로에는 관여하지 않음

핵심 이미 연결된 브릭이 정상인 동안, glusterd 일시 단절만으로 데이터 RPC가 중단되지는 않음

메타데이터 서버와 배치 책임으로 본 이웃들

전용 메타데이터 서비스가 있는가, 배치 책임이 클라이언트에 얼마나 있는가

	기준 1: 전용 메타데이터 서비스		기준 2: 배치 판단이 어디에 있는가
NFS, SMB	서버	서버 안에서 판단	서버가 위치와 정합성을 정함
CephFS	서버 + MDS	MDS 있음	MDS가 메타데이터와 권한 관리, 클라이언트는 CRUSH로 데이터 위치 계산
Lustre	서버 + MDS	MDS 있음	MDS가 메타데이터, 클라이언트는 MDS가 준 레이아웃으로 OST에 접근
GlusterFS	서버 + 없음	없음	클라이언트 xlator가 배치와 복제, 소거 코딩을 조정

핵심 GlusterFS의 대가: 배치, 복제, 소거 코딩 판단이 사용자 공간 클라이언트에 몰림. 이 판단이 커널 포팅의 대상

볼륨 종류와 그래프

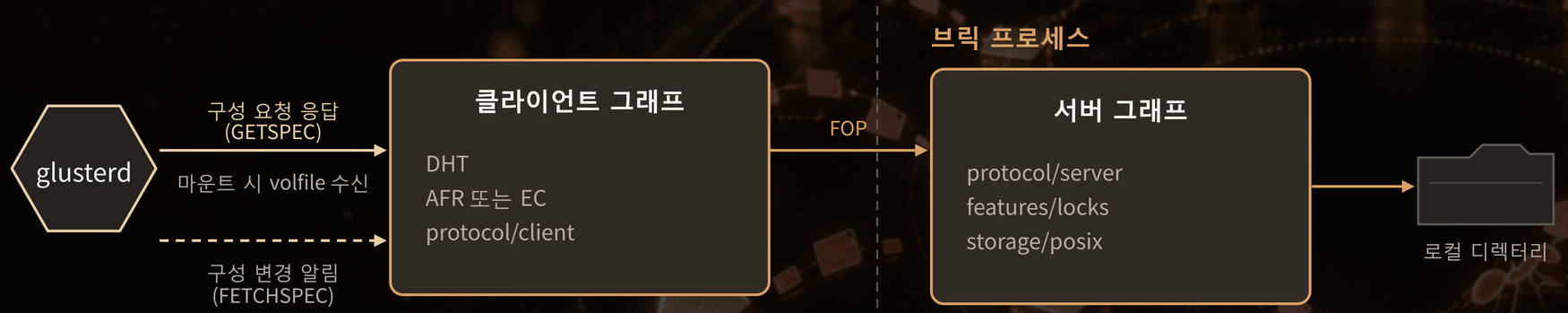
DHT 아래에 AFR 그룹 또는 EC 그룹, 파일 하나가 쓰이는 경로를 겹쳐 표시

호박색 선이 파일 하나의 쓰기 경로: 이름 해시로 그룹을 정한 뒤 그 그룹의 브릭 전부에 복제본 또는 조각을 씀

핵심 AFR과 EC는 한 파일이 차례로 지나는 단계가 아니라 볼륨 종류에 따른 분기, 볼륨 종류가 그래프를 정함

volfile 배포와 서버 그래프

클라이언트 그래프는 glusterd가 배포하고, 브릭 프로세스는 자기 그래프를 가짐



volfile은 볼륨 구성을 적은 텍스트
glusterd가 볼륨 종류에 맞게 만들어 클라이언트에 배포
구성이 바뀌면 glusterd가 구성 변경 알림으로 알리고 클라이언트가 구성 요청으로 다시 받음

브릭도 xlator 스택으로 동작하는 프로세스
클라이언트가 보낸 FOP를 받아 로컬 파일 연산으로 수행
원격 아이노드 잠금(INODELK)과 xattr도 여기서 처리

핵심 두 개의 그래프: 클라이언트 그래프는 volfile로 받고, 서버 그래프는 브릭 프로세스가 가짐. 커널 포팅은 앞쪽만 대상

DHT, AFR, EC의 역할

DHT는 자리, AFR은 복제본 일치, EC는 조각 복원

DHT

배치

파일 이름 해시로 자리를 정함
디렉터리마다 담당 범위표
재분배 중이면 linkto로 안내

AFR

복제

같은 데이터를 브릭 여러 개에
정족수 이상 성공이면 성공 반환
불일치는 자가 복구(heal)로 수렴

조각 6개 중 같은 버전 4개로 복원

EC

소거 코드

원본 조각을 인코딩한 조각으로 분산
같은 버전 조각 k개로 복원
잘못 섞으면 복원 실패나 오류

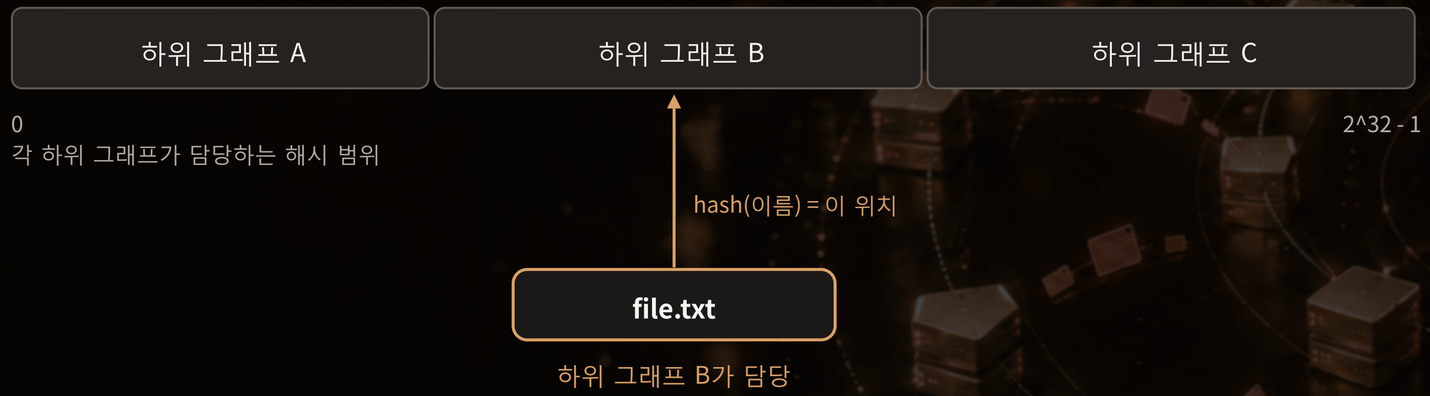
조각 버전은 조각마다 붙는 버전 번호로 어느 쓰기까지 반영됐는지를 뜻함

핵심 세 xlator 모두 클라이언트가 실행. 전용 메타데이터 서버 대신 클라이언트가 배치, 정족수(quorum), 조각 버전(fragment version)을 결정

DHT 배치

디렉터리마다 하위 그래프(child)별 해시 범위를 두고, 파일 이름 해시로 담당 하위 그래프를 찾음

디렉터리 레이아웃(layout) xattr



생성 위치와 조회 라우팅의 단일 기준이 이 해시
재분배 중인 파일은 해시 위치에 linkto 파일을 두어 실제 위치를 안내
커널 클라이언트는 같은 해시 함수로 같은 자리를 계산해야 기존 파일을 찾음

핵심 자리를 정하는 판단이 클라이언트에 있음. 해시 함수 포팅과 레이아웃 스냅샷 규칙은 5장 '포팅하며 깨진 것 열 가지'의 DHT 레이아웃 이슈에서

AFR 복제

같은 데이터를 브릭 여러 개에 쓰고, 정족수 이상 성공이면 응용에 성공을 돌려줌

쓰기

정족수 2인 replica 3 예시, 정상 길이 쓰기 2개 성공이면 성공 반환
실패한 브릭은 나중에 자가 복구로 다른 복제본과 같게 맞춤

읽기

읽기 후보(readable, 최신으로 확인된 복제본) 중에서 선택
정족수 실패나 복제본 분기가 정합성 위협

자가 복구: 복제본 불일치를 복구하는 과정, 자가 복구 데몬 또는 클라이언트가 수행

읽기 후보: 최신으로 확인된 복제본. 이 커널 구현에서는 결과가 불명확한(UNKNOWN) 쓰기를 정족수를 채웠더라도 오류로 처리

핵심 정족수 판단과 읽기 후보 선택이 클라이언트에 있음. 동시 쓰기 순서와 잠금 직렬화는 5장 '포팅하며 깨진 것 열 가지'의 AFR 이슈에서

EC 소거 코드

원본 k조각을 인코딩한 조각 n개를 브릭마다 저장하고, 같은 버전 조각 k개로 복원

복원 조건

같은 유효 버전의 조각 k개가 있어야 복원
아무 조각이나 k개가 아니라 버전이 맞아야 함
잘못 섞으면 복원 실패나 데이터 오류
조각 버전 확인도 클라이언트의 몫

4+2 예시: 브릭 6개 중 같은 버전 조각 4개가 살아 있으면 원본을 만들 수 있음
브릭마다 조각의 비트 배치가 기존 서버와 똑같아야 기존 조각과 섞여 복원됨

핵심 조각마다 붙는 버전 번호와 비트 배치를 커널에서도 똑같이 맞춰야 기존 조각과 섞여 복원됨. 비트 배치는 5장 '포팅하며 깨진 것 열 가지'의 EC 이슈에서

커널로 옮기는 이유

FUSE 경로의 비용과 서버 무수정 전제

FUSE 경로의 구조

FUSE 비용 측정 사례

커널로 옮기면 남는 것과 사라지는 것

운영 중인 서버에 그대로 마운트

프로토콜 호환의 범위

FUSE 경로의 구조

요청과 응답이 사용자 공간과 커널을 지나는 순서

1. 시스템 호출(syscall)로 커널 진입
2. /dev/fuse로 요청 전달, 데몬 깨우기
3. 데몬이 xlator 스택 실행
4. RPC 왕복
5. /dev/fuse로 응답 전달
6. 응용 깨우기와 복귀

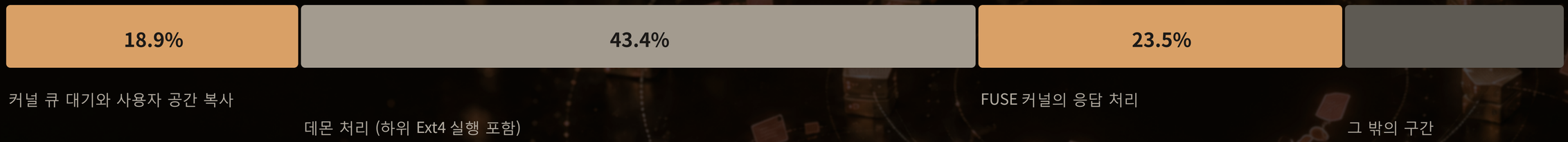
시스템 호출 모드 전환과 데몬 요청 전달을 나눈 개념도, 대기와 깨우기에 스케줄러 관여

핵심 이 경로를 타는 것은 캐시 미스나 서버 요청이 필요한 연산, 비용은 워크로드에 따라 다름
판단 로직이 데몬에 있는 한, 경로를 줄여도 데몬이 죽으면 마운트 전체가 멈추는 의존은 남음

FUSE 비용 측정 사례

두 연구 사례가 측정한 FUSE 경로의 비용

Vangoor 외, FAST 2017, Table 4, StackfsBase, HDD, 4KB 순차 쓰기 1스레드 1파일
쓰기 한 번의 평균 56.7µs를 구간별 비율로



43.4%에는 하위 파일시스템 실행이 들어 있어 FUSE 자체의 비용이 아님, 해석에 주의

RFUSE, FAST 2024, 2.2절, NullFS 빈 생성(CREATE)

사용자 공간 실행 구간 18.1µs
응답을 기다리는 응용의 깨우기 39µs

핵심 연구 사례가 보여 주는 것은 구간이 존재한다는 사실이지 이 포팅의 개선 폭이 아님

커널로 옮기면 남는 것과 사라지는 것

경로는 짧아지지만 서버 왕복은 그대로

glfs.ko 안에서 xlator 로직을 커널 함수로 실행

사라지는 것

/dev/fuse 요청과 응답 전달
데몬 깨우기와
컨텍스트 스위치
사용자 공간 데몬 의존

남는 것

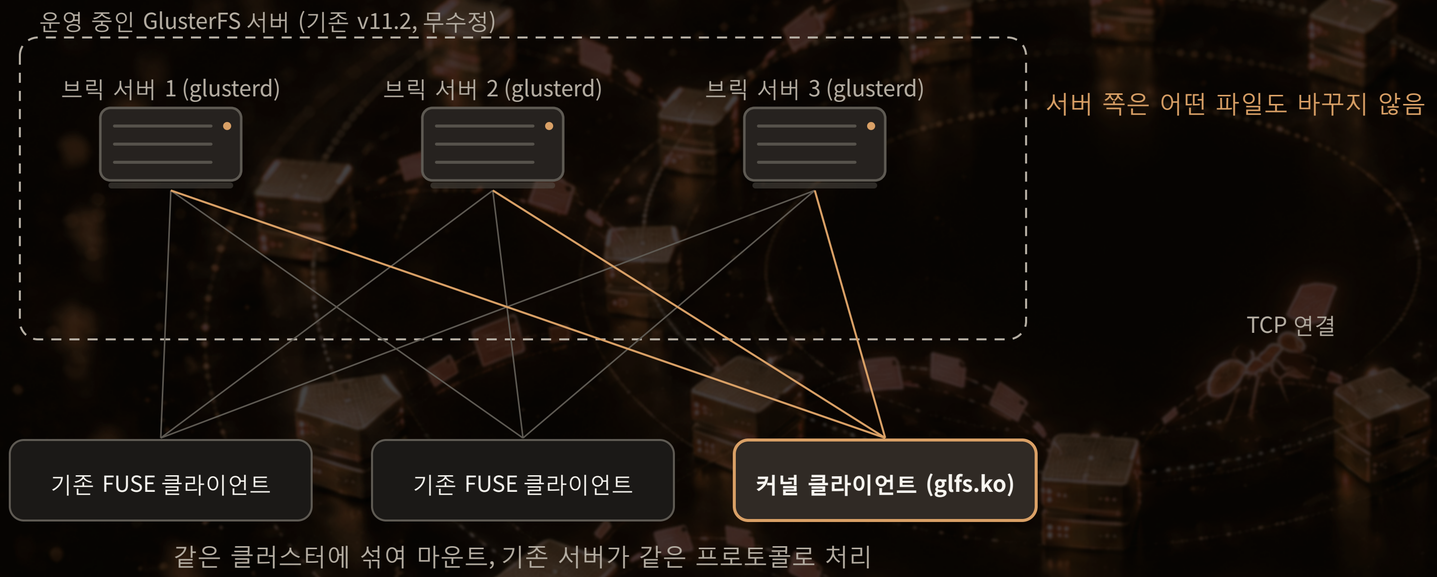
RPC 왕복 대기
데이터 복사 (사용자 버퍼,
커널 버퍼, 네트워크)
커널 안의 sleep 지점

줄어드는 것은 프로세스 사이 전달, 깨우기, 스케줄러 개입. 연구 사례가 측정한 바로 그 구간

핵심 커널 안이라도 서버 왕복과 네트워크 지연은 어느 경로든 똑같이 지불

운영 중인 서버에 그대로 마운트

서버는 기존 v11.2 그대로, 노드별로 클라이언트만 교체



운영 중인 서버에 그대로 마운트하는 것이 이 작업의 출발점
바꾸는 것은 그 노드의 VFS 아래뿐, 기존 FUSE 클라이언트와 공존
기존 서버가 지원하는 핸드셰이크(handshake)와 인증, XDR 규칙을 준수

핵심 기존 서버를 유지하고 노드별로 클라이언트만 교체하는 것이 목표. 그래서 프로토콜 호환이 전제

프로토콜 호환의 범위

전송별로 무엇을 무수정으로 유지하는지

TCP

기존 v11.2 서버 무수정
ONC-RPC와 gfx XDR 절차 전부 동일
문서에 없는 프로토콜 규칙은 기존 구현을 따름
기본 경로, 발표의 검증 대부분이 여기

RDMA

GlusterFS 8 릴리스에서 공식 제거되어 별도로 다시 구현
전송 계층을 갖춘 별도 서버 빌드에서 검증
클라이언트 빌드도 GLFS_RDMA=y로 분리, 기본은 n
서버 이중화 구성은 아직 확인 못 함

무수정 원칙의 실제 의미

기존 서버가 같은 프로토콜 처리 경로로 커널 클라이언트와 FUSE 클라이언트를 함께 받음
원격 flock을 서버가 구분하려면 잠금 클래스(lock-class) 확장이 필요한데 이것은 따로 준비하는 경로, 기본 경로의 서버는 그대로
기존 클라이언트와 함께 쓸 수 있는지 상호운용 시험으로 계속 확인

핵심 호환의 기준은 사양 문서가 아니라 기존 구현의 실제 동작. 5장 '포팅하며 깨진 것 열 가지'의 프로토콜 함정 일곱 가지가 그 사례

glfs.ko 구조

glfs.ko의 내부 구성

- FUSE 경로와 커널 경로
- glfs.ko와 mount helper
- 포팅의 범위
- 기본 자료 구조와 커널 대응
- VFS 연결점 세 곳
- 클러스터 계층과 구성 제어
- 모듈 구성 비율
- ONC-RPC와 XDR을 커널이 직접
- volfile 수명 관리

FUSE 경로와 커널 경로

같은 응용, 같은 브릭, 가운데만 다름

기존 경로 (FUSE)

애플리케이션

↓

VFS

↓

FUSE 커널 모듈, /dev/fuse 대기

↓

FUSE 클라이언트 데몬 (xlator 스택)

↓

TCP로 브릭에

요청마다 커널과 데몬 사이를 오감

포팅 후 (glfs.ko)

기존 애플리케이션, 수정 없음

↓

VFS

↓

glfs.ko (클러스터 계층, RPC, 전송)

↓

TCP로 기존 브릭에 무수정

커널 안 직접 함수 호출, /dev/fuse 전달 없음

핵심 응용, 서버, 프로토콜 전부 무수정, 바뀌는 것은 그 노드의 VFS 아래뿐

glfs.ko와 mount helper

커널 모듈 하나와 최소한의 사용자 공간 helper

glfs.ko (out-of-tree 모듈)

클러스터 계층	DHT, AFR, EC, 잠금
통신 계층	ONC-RPC, XDR, gfx XDR 자체 구현
전송	TCP 기본, RDMA는 InfiniBand verbs API를 쓰는 별도 빌드 옵션
구성 제어	volfile 수신과 갱신, 구성 버전 기록(admission), 차단 상태(FENCED)

mount(2)

mount helper (사용자 공간)

kmount/glfs-mount.c

브릭 목록을 source 문자열로 조립

매개변수를 만들어 mount(2) 호출

상시 데이터 경로에는 관여하지 않음

사용성은 아직 기존 mount.glusterfs에 못 미침

out-of-tree 모듈로 유지하는 비용은 감수

핵심 모듈 하나가 클러스터 판단과 통신을 다 맡고, helper는 마운트 순간에만 등장

포팅의 범위

xlator 그래프를 커널에서 그대로 돌리지 않음

volfile → 커널 표현으로 변환 → 고정 커널 객체 → DHT, AFR, EC, 잠금 알고리즘 수행

기존 프로토콜과 클러스터 알고리즘은 그대로 다시 구현

동적 xlator 그래프 실행 대신 volfile을 커널 자료구조로 바꿔 고정 객체가 수행

VFS 의미 차이(캐시, 쓰기 완료, 잠금)는 4장 '커널 경로에서 달라지는 것'에서 따로 다룸

바뀌는 것과 안 바뀌는 것

바뀌는 것: 그 노드의 VFS 아래, 즉 glfs.ko가 맡는 부분

안 바뀌는 것: 응용, 서버, 프로토콜

핵심 포팅 대상은 클라이언트 그래프의 판단 로직과 통신, 서버 그래프는 손대지 않음

기본 자료 구조와 커널 대응

GlusterFS의 다섯 자료 구조가 glfs.ko에서 무엇이 되는가

xlator

기능 단위 모듈, 그래프로 쌓임

DHT, AFR, EC도 전부 xlator

커널: k_dht, k_afr, k_ec 고정 객체

FOP

조회(LOOKUP), 쓰기 등의 요청

xlator 스택을 내려가 RPC로 브릭에

커널: VFS 콜백 안에서 직접 호출

사전(dict)

키와 값의 목록, 요청과 응답에 실림

인증 정보, 옵션, xattr을 여기에 담음

커널: FOP는 gfx_dict XDR로

볼륨 설정(SETVOLUME)은 구형

GFID

파일마다 붙는 128비트 고유 ID

경로가 아니라 GFID로 브릭에 요청

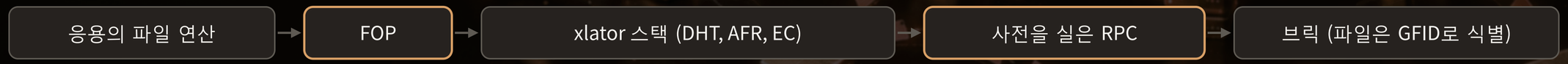
커널: 아이노드(inode)와 GFID 대응

volfile

볼륨 그래프의 구성 파일

glusterd가 배포, 옵션과 브릭 목록

커널: 구성 스냅샷으로 파생

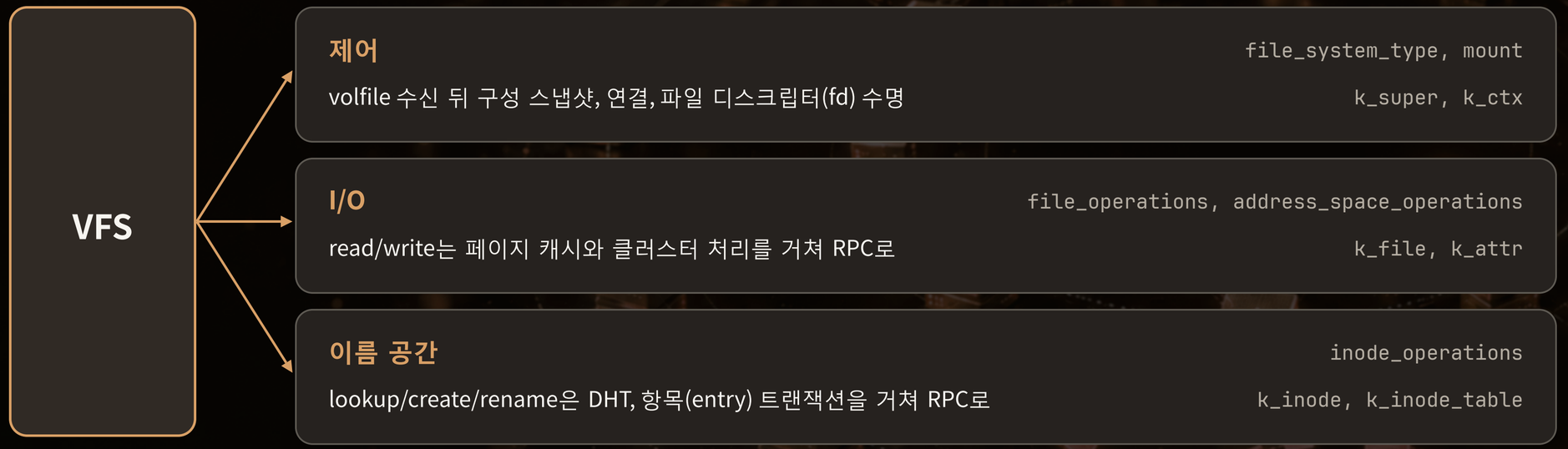


호박색 줄이 커널 쪽 대응. 전송 형식과 식별 규칙은 그대로 따르고, 내부 실행 구조는 커널에 맞게 구현

핵심 전송 형식과 식별 규칙은 그대로, 내부 실행 구조는 커널 객체로. 사전은 5장 '포팅하며 깨진 것 열 가지'의 프로토콜 함정에서 다시

VFS 연결점 세 곳

제어, I/O, 이름 공간(namespace)이 각각 다른 VFS 콜백으로 들어옴



핵심 연산마다 VFS가 서로 다른 콜백을 부르고, 세 연결점 뒤에서 같은 클러스터 계층을 씀

클러스터 계층과 구성 제어

VFS 연결점 뒤에서 판단을 맡는 모듈들

DHT

k_dht

AFR

k_afr_*, k_heal

EC

k_ec_*

원격 잠금

k_rlk

AFR: k_afr_txn, lock, inodelk, entrylk, readsubvol. EC: k_ec_fop, consistency, inodelk, gf, simd, heal

구성 제어

구성 버전 기록(admission)

연산이 시작할 때 그 시점의 구성 버전을 적어 둠
연산 도중 구성이 바뀌어도 시작할 때 적어 둔 구성으로 끝까지 진행

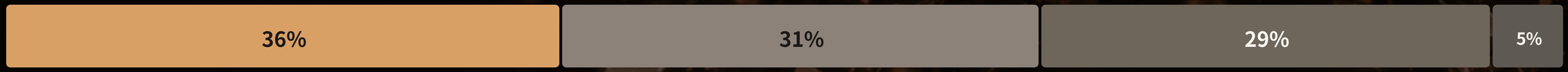
차단 상태(FENCED)

지원하지 못하는 구성 변경이 오면 접근을 막는 상태
되돌릴 수 있으면 원래 구성 복원 후 재개

핵심 판단 로직은 고정 객체, 구성 변경은 구성 버전 기록과 차단 상태로 진행 중 연산과 분리

모듈 구성 비율

실제 코드 줄 수 비율, 주석과 공백 포함, 헤더와 테스트 코드 제외



- VFS 연결, fop 실행, 속성과 캐시 재검증
  `k_super, k_ctx, k_file, k_inode, k_attr, k_xattr`
- 클러스터 알고리즘 (DHT, AFR, EC, 자가 복구, 잠금)
  `k_dht, k_afr_*, k_ec_*, k_heal, k_rlk`
- 통신 스택 (ONC-RPC, XDR, gfx, 전송)
  `rpc_clnt/, k_xdr, k_rpc_clnt, k_async, k_tcp, k_rdma`
- 구성과 보조 5% (volfile 갱신, 구성 버전 기록, 차단 상태)
  `k_volrefresh, k_volcfg, k_admit, k_fence, k_fsync_retry`

세 덩어리가 거의 같은 무게

통신 스택도 커널 안에
KUnit 테스트 코드는 구현 코드의 약 3분의 2 분량
헤더의 절반은 커널과 mount helper가 공유하는 include/glfs.h

핵심 클러스터 알고리즘만이 아니라 통신 스택까지 커널 안에 직접 구현했다는 뜻

ONC-RPC와 XDR을 커널이 직접

프로토콜은 그대로, 구현만 커널 안에

k_xdr가 인코딩과 디코딩 전부 담당

- 볼륨 설정 인증: volfile의 인증 정보(username, password)를 사전에 실어 전송
- 파일 연산 버전(fops-version) 같은 핸드셰이크 절차
- 마지막 조각(last-fragment) 비트, RPC 레코드 경계 표시(record marking) 같은 세부
- 서버가 지원하는 핸드셰이크, 인증, XDR 규칙을 그대로 준수

TCP

k_tcp
기존 v11.2 서버 무수정

RDMA

k_rdma, InfiniBand verbs
GLFS_RDMA=n이 기본

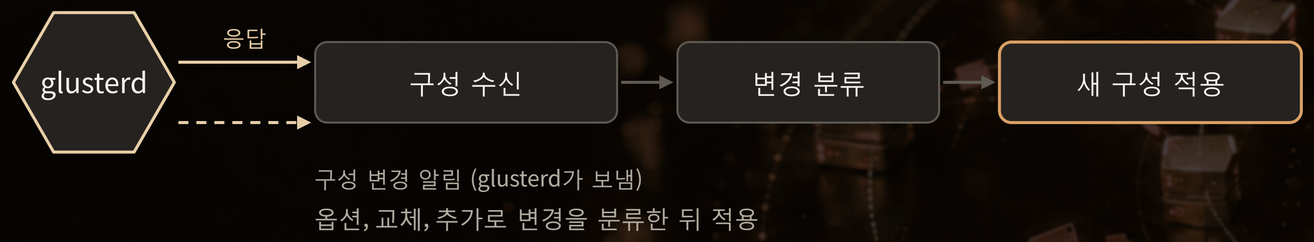
전송 모드는 동기(synchronous)와 비동기(asynchronous)

기본은 비동기, 전송 스레드의 요청 처리 방식일 뿐
되쓰기(writeback) 캐시나 시스템 호출의
비동기 완료와는 별개

핵심 커널 sunrpc에 연결하지 않고 GlusterFS의 인증, 콜백, gfx XDR을 자체 RPC 계층에서 처리

volfile 수명 관리

k_volrefresh가 받아서 분류하고, 적용



구성 변경 알림 (glusterd가 보냄)

옵션, 교체, 추가로 변경을 분류한 뒤 적용

백업 volfile 서버

첫 glusterd가 끊기면 후보를 바꿔서 다시 수신

구성 버전 기록

구성 버전마다 기록을 남기고, 진행 중 FOP의 연결 출처는 시작 시점 구성

구성 변경의 분류와 차단 상태 처리는 5장 '포팅하며 깨진 것 열 가지'의 구성 변경 이슈에서

핵심 마운트 시 한 번 받고, 바뀌면 알림을 받아 다시 받고, 적용은 구성 버전 기록으로 진행 중 연산과 분리

커널 경로에서 달라지는 것

같은 왕복, 다른 경로

FUSE 경로의 실행 모델

커널 경로의 실행 모델

같은 비용과 줄어드는 비용

쓰기 완료의 의미

의도된 차이

이미 맞춘 것과 남은 것

FUSE 경로의 실행 모델

xlator 스택이 별도 사용자 프로세스에서 실행

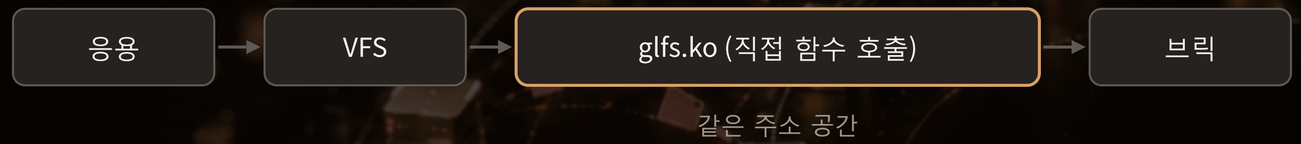
- xlator 스택은 별도 사용자 프로세스에서 실행
- 요청은 /dev/fuse를 거치고 응용 프로세스는 기다림
- 데몬 스레드로 넘어가며 컨텍스트 스위치 발생
- 응답은 커널을 거쳐 돌아오고 응용을 깨움

시스템 호출 모드 전환은 남고, 데몬으로의 요청 전달과 대기, 깨우기가 추가됨

핵심 판단 로직이 데몬에 있는 한 데몬이 죽으면 마운트 전체가 멈추는 의존이 남음

커널 경로의 실행 모델

같은 주소 공간, 직접 함수 호출



- 클러스터 로직은 VFS 콜백 안에서 직접 호출
- VFS에서 클러스터 처리는 커널 안 함수 호출로 이어짐
- 응답은 RX 스레드가 처리, 호출자는 완료 대기 (동기 모드는 옵션)
- 사용자 문맥과 커널 스레드를 그대로 사용

남는 것은 RPC 왕복 대기와 데이터 복사, 커널 안에도 sleep 지점

핵심 사용자 공간 데몬 의존 없음, 그러나 대기와 복사가 없어진 것은 아님

같은 비용과 줄어드는 비용

실행 모델 기준으로 나눈 비용 항목

같은 비용

- 서버 왕복 자체
- 네트워크 지연
- RPC 완료 대기
- 사용자 버퍼와 커널 버퍼 사이 복사
- 네트워크 전송 경로의 복사

줄어드는 비용

- 커널과 데몬 사이 요청 전달
- 데몬 깨우기와 컨텍스트 스위치
- 스케줄러 개입
- 사용자 공간 데몬 의존

연구 사례에서 측정한 바로 그 구간

핵심 서버 왕복과 네트워크 지연은 어느 경로든 똑같이 지불

쓰기 완료의 의미

양쪽 모두 즉시 쓰기(write-through) 경로가 기본

응용 write()

↓

VFS와 페이지 캐시 (dirty 폴리오(folio) 없음)

↓

glfs.ko write_end에서 즉시 쓰기 RPC

↓

브릭 (내구성은 동기화(FSYNC) RPC 때)

FUSE 클라이언트

캐시를 거치는 즉시 쓰기가 기본

커널 writeback-cache 옵션은 선택, 기존 v11.2도 기본 false

glfs.ko

즉시 쓰기 단일 경로, write_end가 바로 쓰기 RPC

dirty 폴리오를 만들지 않아 되쓰기 경로 없음

공유 쓰기 mmap 옵션을 켠 마운트만 되쓰기 경로 사용

쓰기 응답과 저장 장치 내구성은 별개, 내구성은 glfs_fsync의 동기화 RPC로

핵심 쓰기 완료의 뜻은 양쪽이 같고, 되쓰기 경로의 유무만 다름

의도된 차이

우회를 택한 네 가지

mmap	비공유(private) 쓰기 매핑은 COW로 허용, 공유 쓰기 매핑은 마운트 옵션 glfs_mmap_write(기본 비활성화)를 켠 마운트에서만, 꺼져 있으면 -EACCES
SETLKW	비동기 전송이면서 복제도 EC도 아닌 경로만
O_DIRECT	버퍼 입출력(buffered) 에뮬레이션과 강제 재검증, 무효화 실패는 오류 또는 나중에 처리
원격 flock	잠금 클래스 서버 조건, 기본 비활성화

핵심 FUSE 클라이언트와 다르게 둔 곳은 전부 의도된 선택

이미 맞춘 것과 남은 것

FUSE 클라이언트 완전 포팅은 목표이지 달성 선언은 아님

이미 맞춘 것

- 닫기 후 열기(close-to-open) 파일 속성과 데이터 재검증
- inodelk와 entrylk 직렬화, 기본 활성화
- 연결과 파일 디스크립터 수명 관리, 기본 활성화
- dentry 이름 재검증, 완료
- 공유 쓰기 mmap, 옵션 glfs_mmap_write로 완료

남은 것

- SETLKW 취소, 아직 결정하지 않음
- 정족수 옵션의 쓰기 경로 정책 통합
- glfs_mmap_write 기본 활성화 전환 결정
- 비동기 전송 모드는 되쓰기 캐시와 무관 (오해 주의)

핵심 차이를 찾는 방식은 그대로, FUSE 동작과 대조하고 재현 시험이 통과할 때까지 수정

포팅하며 깨진 것 열 가지

무엇을 지켜야 했고, 어디서 깨졌고, 어떻게 고쳤나

1 문서화되지 않은 프로토콜 규칙의 함정들
2 DHT 레이아웃과 linkto
3 AFR 동시 쓰기 순서와 직렬화
4 EC 비트 슬라이스
5 잠금 종류 세 가지
6 동시 붙여쓰기
7 닫기 후 열기
8 마운트 중 구성 변경
9 statfs 집계 산술
10 O_DIRECT 에뮬레이션

문서화되지 않은 프로토콜 규칙의 함정들

읽기 전용 TCP 연동 초기에 하나씩 만남, 문서에 없는 규칙은 기존 구현이 정답

#	함정	증상	어디를 고쳤나
1	Sun RPC 마지막 조각 비트 누락	브릭 응답 없음	레코드 마커 상위 비트
2	SETVOLUME 인증 누락	브릭 인증 거부	인증 정보(username, password) 사전
3	SETVOLUME 사전 값 불일치	핸드셰이크 실패	fops-version, client-version, opversion
4	FOP v400 미전환	"version not available"	gfx_dict, gfx_iattx XDR
5	gfx_open_rsp 필드 순서	응답 파싱 깨짐	op_ret, errno, dict, fd 순서
6	gfx_dict_skip이 빈 사전만 처리	-EBADMSG	본문을 길이만큼 건너뛰기
7	READDIRP 프로시저 번호 하나 차이	ls에 "Connection refused"	41을 40으로, OPENDIR 29를 20으로

핵심 일곱 개를 하나씩 보면 전부 "사양 문서가 아니라 기존 구현이 기준"이라는 같은 결론

함정 1 마지막 조각 비트

증상: 브릭이 영원히 응답하지 않음

Sun RPC 레코드 마커 (4바이트)

보낸 것	`0`	`0 0 0 len`	상위 비트 0, 브릭은 다음 조각을 기다림
고친 것	`1`	`0 0 0 len`	len | 0x80000000, 이 조각이 마지막 조각이라는 표시

수신 쪽은 반대로 상위 비트를 걷어내야 길이가 나옴: len & 0x7fffffff

RFC 5531의 레코드 마킹 표준인데, GlusterFS 문서에는 이 조각 규칙이 따로 적혀 있지 않음

원인

송신 길이에 상위 비트를 OR 하지 않음

브릭은 뒤에 올 조각을 계속 기다림

수정

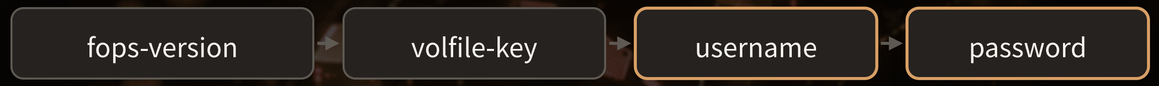
송신은 len | 0x80000000

수신은 & 0x7fffffff로 마스킹

함정 2 SETVOLUME 인증 누락

증상: 브릭이 인증을 거부

SETVOLUME 요청의 사전



인증 정보는 volfile 안에 있고, 이 정보를 사전에 실어 보내야 브릭이 받아 줌

기존 FUSE 클라이언트의 패킷을 덤프해 어떤 키가 오가는지 대조

원인

volfile의 인증 정보(username, password)를 볼륨 설정 요청의 사전에 넣지 않음

수정

volfile에서 읽은 두 값을 사전에 포함

문서가 아니라 패킷 덤프로 확인

함정 3 SETVOLUME 사전 값 불일치

증상: 핸드셰이크 실패

SETVOLUME 사전의 값들

`fops-version`	1298437 (매직 넘버, RPC 프로그램 버전과 다름), 서버가 검증
`remote-subvolume`	볼륨명이 아니라 브릭 경로, 서버가 검증
`client-version`	이 구현은 11.2를 전송, 서버는 없으면 로그만
`opversion`	함께 전송, 서버는 못 읽으면 로그만

원인

fops-version과 remote-subvolume이 문서와 다르거나 문서에 없음
필수 키와 서버의 실제 검증 조건은 구분해야 함

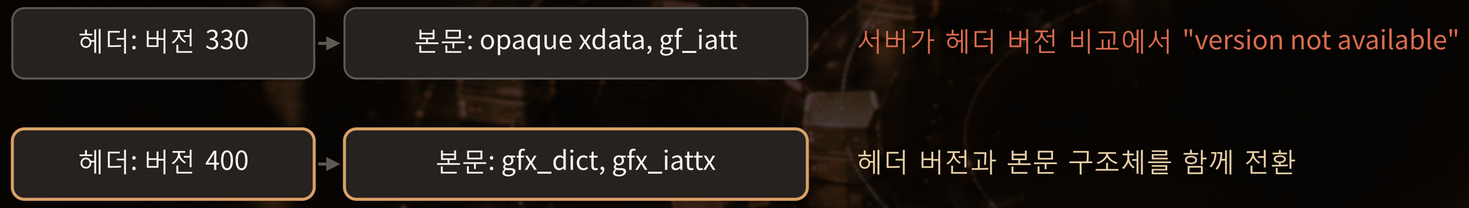
수정

fops-version 1298437, remote-subvolume은 브릭 경로
client-version 11.2와 opversion도 함께 전송

함정 4 FOP v400 미전환

증상: "version not available"

RPC 헤더의 프로그램 버전과 본문 구조체



거부는 헤더의 프로그램 버전 비교에서 나고, 본문 구조체는 버전에 맞춰 따로 바꿔야 함

원인

RPC 프로그램 버전 330 요청을 서버가 거부
본문도 기존 형식이라 400에 맞지 않음

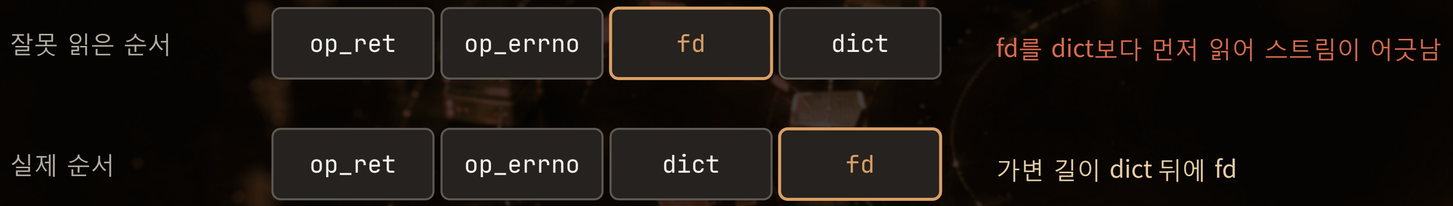
수정

헤더를 프로그램 버전 400으로 전환
본문도 gfx_dict, gfx_iattx로 함께 전환

함정 5 gfx_open_rsp 필드 순서

증상: 응답 파싱이 깨짐

gfx_open_rsp 디코딩 순서



XDR은 필드 순서가 곧 사양, 한 필드만 어긋나도 뒤의 모든 값이 쓰레기

원인

fd를 dict보다 먼저 디코딩

가변 길이 dict 뒤의 위치가 밀림

수정

op_ret, errno, dict, fd 순서로 수정

기존 구현의 .x 정의를 기준으로 대조

함정 6 gfx_dict_skip이 빈 사전만 처리

증상: -EBADMSG

응답 안의 사전을 건너뛰는 함수

전에는 `xdr_size` `count` `pairs_len` 항목이 0개인 빈 사전만 처리, 항목이 있으면 -EBADMSG로 거부

지금은 `xdr_size` `count` `pairs_len` `pairs 본문` 본문 바이트 수, 항목 수, 쌍 배열 길이의 세 정수를 읽은 뒤 본문 바이트 수만큼 건너뜀

관심 없는 사전이라도 세 정수와 본문 바이트 수를 정확히 소비해야 다음 필드가 맞음

원인

항목이 있는 사전을 오류로 거부
본문 길이를 소비하지 않아 뒤 필드 어긋남

수정

세 정수 12바이트 뒤 본문을 바이트 수만큼 건너뜀
내용은 무시해도 길이는 존중

함정 7 READDIRP 프로시저 번호

증상: ls가 "Connection refused"를 냄

GFS3 프로시저 번호

연산	보낸 번호	실제 번호
READDIRP	41 (실제로는 RELEASE)	40
OPENDIR	29	20

서버는 RPC GARBAGE_ARGS로 거부했는데 클라이언트가 이것을 -ECONNREFUSED로 매핑

그래서 TCP 연결 문제로 보였고, 일곱 개 중 가장 오래 걸린 함정

원인

프로시저 번호가 하나씩 어긋남

RPC 거부를 연결 거부로 잘못 매핑

수정

41을 40으로, 디렉터리 열기(OPENDIR) 29를 20으로

RPC 거부는 -EPROTO와 accepted_stat 로그로

함정에서 배운 것

표준과 XDR 정의, 서버 구현의 대조

전에는 RPC 거부를 전부 연결 거부로 보고했기 때문에 프로토콜 문제가 네트워크 문제처럼 보였음

함정들의 공통점

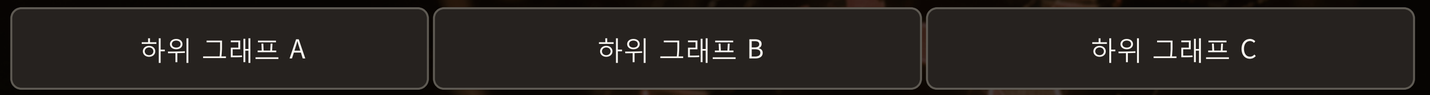
- GlusterFS 문서에 없거나 문서와 다른 값은 실제 서버가 기준, 표준에 있는 것은 표준이 기준
- 기존 FUSE 클라이언트의 패킷 덤프와 .x 정의가 가장 정확한 참조
- 오류 매핑이 틀리면 진짜 원인이 다른 층의 문제로 보임

핵심 놓친 프로토콜 규칙을 표준, XDR 정의, 서버 구현과 대조해 상호운용 시험으로 확인

DHT 레이아웃과 linkto

1장 'GlusterFS 빠른 개요'의 배치 규칙을 커널이 같은 값으로 계산해야 함

1. 디렉터리의 레이아웃 xattr

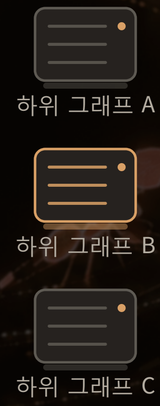


h가 들어가는 범위

2. 파일 이름을 해시

3. 범위를 가진 하위 그래프가 담당

생성 위치와 조회 라우팅의 단일 기준이 이 해시, 해시 함수는 Davies-Meyer 구조에 TEA 변형 라운드



하위 그래프는 xlator 바로 아래 단
복제 그룹, 소거 그룹, 또는 브릭 하나

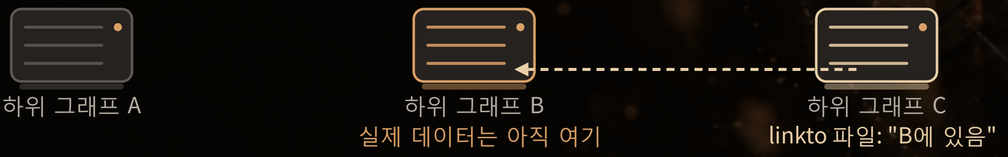
핵심 커널 클라이언트가 같은 해시로 같은 자리를 계산해야 기존 클라이언트가 만든 파일을 찾음

linkto 파일

해시상 위치와 실제 데이터 위치가 다를 때

재분배 전

재분배 뒤 (레이아웃이 바뀌어 해시상 위치가 하위 그래프 C)



linkto가 하는 일

- 해시상 위치에 놓인 특수 파일
- 실제 데이터가 있는 하위 그래프를 가리킴
- 조회가 linkto를 만나면 그 하위 그래프로 한 번 더
- 재분배가 끝나면 데이터가 옮겨지고 linkto는 정리

핵심 커널 클라이언트도 linkto 추적과 잔여 링크 파일(stub) 정리를 그대로 구현

기존 DHT와 같은 답을 내야 하는 판단 지점

k_dht.c가 기존 DHT와 같은 동작을 해야 하는 지점

레이아웃 조회와 캐시	디렉터리 레이아웃 xattr를 조회해 캐시하고 하위 그래프별 해시 범위로 담당 결정
해시 함수 동일값	gf_dm_hashfn을 그대로 포팅, 서버 libglusterfs 해시와 같은 값을 냄을 확인
파일 생성 라우팅	라우팅 결정과 사전 검사는 같은 레이아웃 스냅샷에서, 그 사이 담당이 바뀌면 기존 범위로 보냄
linkto 추적과 잔여 링크 파일 정리	조회 결과가 어긋나면 linkto 추적과 전체 탐색으로 흡수, 잔여 링크 파일은 정리

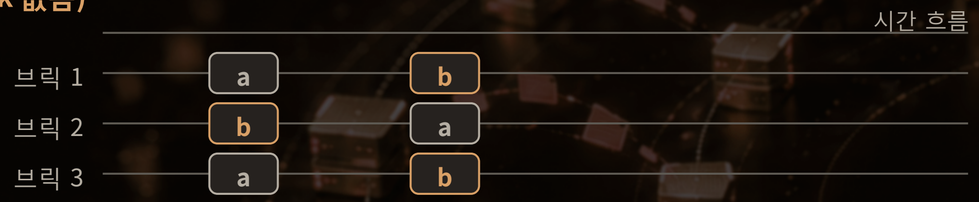
처음에는 해시 범위를 하위 그래프 수로 균등하게 나눈 것으로 가정했다가 기존 파일을 못 찾았고, 지금은 디렉터리 xattr의 레이아웃을 그대로 읽어 결정

핵심 해시가 다르면 같은 이름이 다른 하위 그래프로 향해 기존 파일을 못 찾기 때문에 서버 해시와 동일값을 확인

AFR 동시 쓰기 순서와 직렬화

replica 3에서 두 마운트가 같은 파일에 동시에 쓰면 브릭마다 적용 순서가 달라짐

직렬화 포팅 전 (inodelk 없음)



마운트 A의 a와 B의 b가
브릭마다 다른 순서로 적용

이후 읽는 위치에 따라 서로 다른 내용

원인

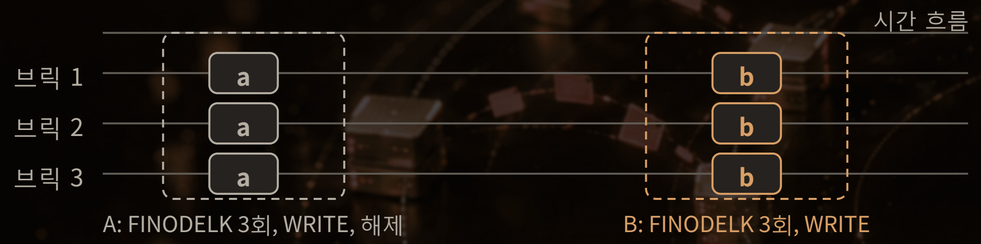
- 마운트 간 쓰기 직렬화를 맡는 inodelk가 커널 클라이언트에 없었음
- 포팅 중에는 시간차로 3분의 2만 잠금을 얻은 상태를 정족수 성공으로 인정해 일부 브릭에만 기록
- 측정값과 정정 내역은 노트와 6장 '남은 일과 검증 범위'의 시험 결과 표에

핵심 쓰기 분기는 조용히 생기고 읽을 때 드러남, 복제본이 갈라지면 정합성이 깨짐

FINODELK 직렬화

A가 세 브릭 전체에 잠금을 잡은 뒤 쓰고, 풀고 나서 B

직렬화 포팅 후



세 브릭 모두 같은 순서

경합으로 거부되면 해제 확인 후 유한 재시도

해법

- 연결과 파일 디스크립터가 유효한 대상 집합을 트랜잭션 시작에 한 번 확정
- 그 집합 전체에서 파일 아이노드 잠금(FINODELK) 획득이 끝난 뒤에만 쓰기 (GFS3_OP_FINODELK)
- 기존 AFR의 논블로킹 빠른 경로(nonblocking fast-path) 안전 규칙을 그대로 포팅

핵심 잠금 프로토콜 자체를 기존 클라이언트가 실제로 쓰는 방식 그대로

직렬화 규칙과 잠금 소유자

교착을 피하고, 불명확한 결과(UNKNOWN)는 오류로

규칙

- GF_LK_SETLK 논블로킹만, SETLKW 대기 금지 (교착 회피)
- 경합 거부는 해제 확인 후 유한 재시도
- 결과가 불명확하면(UNKNOWN) 정리 후 -EIO, 자동 재실행 없음
- 쓰기 뒤 해제가 불명확해도 성공으로 접거나 다시 쓰지 않음

잠금 소유자(owner)와 도메인

- 도메인 <vol>-replicate-<group>, 자가 복구 데몬과 같음
- 범위 [0, EOF]
- 소유자는 트랜잭션마다 8바이트 빅 엔디언(big-endian) 값을 1회 발급
- 같은 클라이언트 안에서 값이 겹치면 별개 트랜잭션의 잠금을 같은 소유자로 볼 위험

핵심 커널 문맥에서 블로킹 대기는 교착 위험, 그래서 논블로킹과 유한 재시도만

EC 비트 슬라이스

기존 서버의 조각과 비트 단위로 같아야 함

512바이트 청크(chunk)를 어떻게 해석하는가

바이트 단위(byte-wise) `b0` `b1` `b2` `b3` (중략) `b511` 바이트마다 GF(2^8) 곱셈, 매핑이 다름

비트 슬라이스(bit slicing) `plane 0` `plane 1` `plane 2` `plane 3` `plane 4` `plane 5` `plane 6` `plane 7`

같은 비트 자리를 모아 비트 평면(plane) 8개, 평면마다 u64 8개, 곱셈은 평면 시프트(shift)와 XOR

왜 바꿀 수 없는가

- 기존 EC가 비트 슬라이스로 연산하므로 브릭에 저장된 조각의 비트 배치가 그 결과
- 바이트 단위로 다시 구현하면 계산상 같은 GF 연산이라도 배치가 달라 기존 조각과 섞이지 않음

핵심 연산 방식은 선택이 아니라 호환 조건

비트 슬라이스 코드

GF(2^8)에서 2를 곱하는 xtime

kernel-client/k_ec_method.c, glfs_ec_bs_xtime (u64 in[64]에서 out[64]로)

```
/* GF(2^8)x2 bitslice. in[64] to out[64]. plane shift + reduce(0x11d) */
out[i] = p7; /* plane 0: reduce */
out[8+i] = p0; /* plane 1: shift */
out[2*8+i] = p1 ^ p7; /* plane 2: shift+reduce */
out[3*8+i] = p2 ^ p7; /* plane 3 */
out[4*8+i] = p3 ^ p7; /* plane 4 */
out[5*8+i] = p4; /* plane 5: shift */
out[6*8+i] = p5; /* plane 6: shift */
out[7*8+i] = p6; /* plane 7: 구 MSB shift */

/* 곱셈 muladd: coeff 이진 분해, 1인 비트마다 64 u64 전체에
XOR 누적 후 xtime으로 한 단계 진행 */
```

읽는 법

1. 최상위 비트가 넘치면 0x11d로 환원(reduce)
2. 그 환원 항이 어느 평면에 더해지는지가 곧 배치
3. 청크 하나는 평면 8개, 평면마다 u64 8개, 그래서 in[64]
4. 배치를 하나라도 어기면 서버 조각과 비트가 어긋남

핵심 계산은 같아도 배치가 다르면 호환이 아님

SIMD 구현 선택

비트 슬라이스는 XOR와 시프트뿐이라 벡터화가 쉬움

빌드 확인만, ARM 런타임 검증 없음

모듈 적재 시 CPU가 지원하는 기능을 한 번 확인해 사용할 구현을 고름

작성 방식

- GCC 벡터 확장(vector extension), intrinsic 헤더 없이
- -msse2, -mavx2 플래그만으로 pxor와 movdqu 생성
- muladd 제어 구조는 스칼라와 코드 공유

커널 내부 제약

- SIMD 구간은 kernel_fpu_begin과 end로 보호
- 해당 SIMD 경로가 없으면 스칼라가 대신
- selftest는 선택된 구현의 xtime을 스칼라와 대조

핵심 스칼라가 기준, selftest는 선택된 SIMD의 xtime 결과가 스칼라와 같은지 확인

잠금 종류 세 가지

POSIX, flock, 열린 파일 디스크립터(OFD) 잠금은 소유자 단위와 충돌 관계가 다름

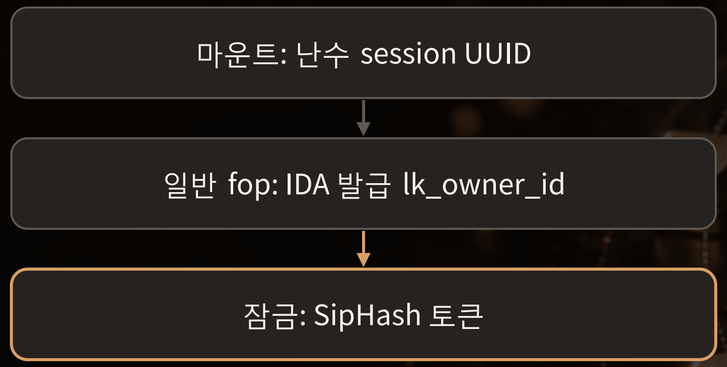
잠금 종류	소유자 단위	충돌 관계	프로토콜
POSIX (fcntl)	`files_struct`	OFD 잠금, 레코드 잠금(record lock)과 충돌	GFS3_OP_LK
OFD (열린 파일 디스크립터)	`struct file`	POSIX 잠금, 레코드 잠금과 충돌	GFS3_OP_LK
flock (BSD)	`struct file`	레코드 잠금과 별개 도메인	VFS 로컬, 옵션으로 잠금(LK)

잠금 클래스는 서버가 잠금 종류를 구분하게 하는 GlusterFS 쪽 확장, 커널 lockdep의 잠금 클래스(lock class)와는 무관

핵심 잠금 소유자 표현이 다른 세 종류를 커널 객체에서 전송 토큰으로 매핑해야 함

소유자 식별 3단계

마운트, 일반 fop, 잠금이 각각 다른 식별자를 씀



마운트마다 새로 만듦, 서버가 클라이언트를 구분하는 값

마운트 안에서 발급하는 기본 소유자

POSIX는 files_struct, flock과 열린 파일 디스크립터 잠금은 struct file 포인터를 토큰화해 전송

SipHash 토큰화

포인터값을 마운트 로컬 시드로 해시해 프로토콜에 실음, 서로 다른 포인터가 자동으로 분리되어 두 이름 공간이 섞이지 않음

같은 소유자의 merge, replace, close도 서버에서 정확히 처리

핵심 복제본 잠금 직렬화는 트랜잭션마다 별도 소유자를 씀

원격 flock과 잠금 클래스

선택 활성화 기능이고, 기존 서버는 잠금 종류를 구분하지 않음

원격 flock

- 기본은 VFS 로컬 처리
- flock_remote 옵션을 활성화하면 잠금 요청을 서버에 보냄
- 기존 서버의 posix-locks는 클래스를 구분하지 않고 하나의 소유자 이름 공간
- 그래서 POSIX와 flock이 서로 EAGAIN을 낼 수 있음

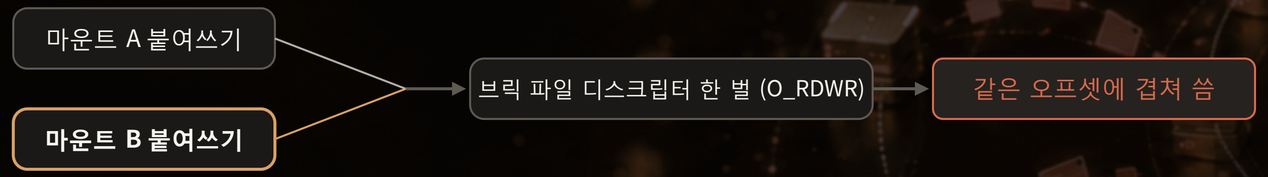
잠금 클래스 서버 확장

- 클래스를 xdata에 실어 서버가 구분하게 하는 확장
- 별도 준비 경로, 기본 경로에서는 서버 변경 없음
- 클래스를 안 보내는 레거시(legacy) 클라이언트는 POSIX 취급 보존
- 커널 lockdep의 잠금 클래스(lock class)와는 무관

핵심 TCP 잠금 경로에서 별도로 준비한 서버 확장, 기본 경로에서는 쓰지 않음

동시 붙여쓰기

두 노드가 같은 파일에 붙여쓰기(append)하면 같은 오프셋(offset)에 겹쳐 씀



- 브릭 파일 디스크립터는 아이노드마다 한 벌이고 항상 O_RDWR로 열림
- O_APPEND를 실어 보내지 않으면 두 노드의 줄이 같은 오프셋에 겹쳐 쓰임
- 포팅 전 측정에서 실제로 줄이 사라짐 (수치는 노트와 6장 '남은 일과 검증 범위'의 표)

기존 구현의 주석

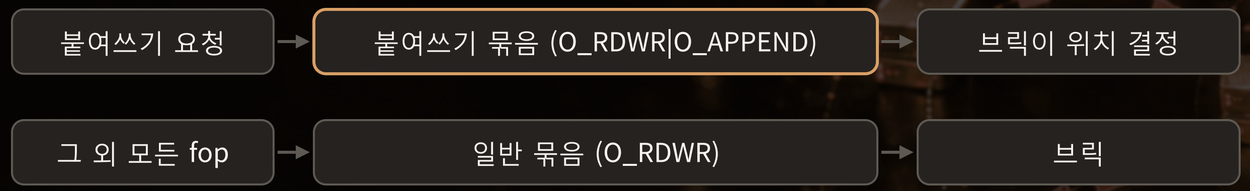
```
"Appending writes take full locks
so size won't change..."
(ec-inode-write)
```

기존 EC 구현은 전체 파일 잠금으로 크기를 고정하고
클라이언트가 논리 EOF를 붙여쓰기 위치로 사용

핵심 당시 커널 구현은 잠금도 O_APPEND도 없이 클라이언트가 계산한 오프셋으로 써서 다른 노드의 붙여쓰기를 몰랐음

파일 디스크립터 묶음

붙여쓰기용과 일반용, 브릭마다 두 개의 파일 디스크립터 묶음(bank) 유지



- 묶음은 브릭마다 같은 열기 플래그로 연 파일 디스크립터의 집합
- 일반 묶음은 O_RDWR로 열어 붙여쓰기가 아닌 모든 fop 담당
- 붙여쓰기 묶음은 O_RDWR|O_APPEND로 열어 붙여쓰기만 전담

열기 시점

- open에 O_APPEND가 있으면 즉시 붙임
- F_SETFL이나 RWF_APPEND로 뒤늦게 붙으면 첫 붙여쓰기 때 열기
- 처음 열기 실패만 일반 묶음의 위치 지정 쓰기
- 연 뒤나 기존 묶음 복구 실패는 오류

핵심 다른 노드의 붙여쓰기 데이터를 낡은 오프셋으로 덮지 않기 위한 장치

볼륨 유형별 규칙

glfs_append_write_admit이 볼륨 유형마다 다르게 판단

DHT 분산만	붙여쓰기 묶음의 O_APPEND로 브릭이 위치 결정
AFR 복제	inodelk 직렬화 위에서만 허용, inodelk가 꺼져 있으면 -EOPNOTSUPP
EC 소거	조각 정렬과 2차 범위 문제를 풀기 전까지 -EOPNOTSUPP로 거부

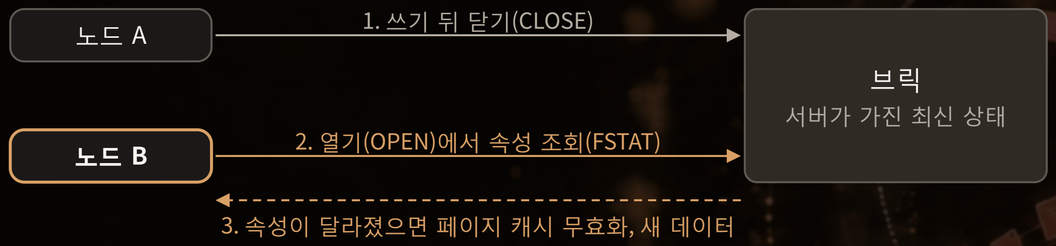
쓰기 뒤 처리

성공한 쓰기의 쓰기 후 속성(poststat)으로 로컬 EOF를 갱신, 서버의 붙여쓰기 위치가 로컬 예상과 다르면 나중에 반드시 캐시 무효화

핵심 EC 볼륨은 조각 정렬 문제를 풀기 전까지 >>로 연 파일의 실제 붙여쓰기에 -EOPNOTSUPP, 열기 자체는 허용

닫기 후 열기

A가 쓰고 닫은 뒤 B가 열면 새 데이터가 보여야 함



닫기 후 열기 일관성(close-to-open)

- NFS의 닫기 후 열기 일관성 보장과 동일
- 열린 상태에서 읽는 동안의 변경은 대상이 아님

- 다른 노드의 쓰기와 내 페이지 캐시의 불일치
- 기존 FUSE는 fopen-keep-cache=off로 open마다 캐시를 버림
- 메타데이터 캐시 xlator인 md-cache는 서버 변경 통지(upcall)로 무효화
- read_consistency는 AFR stale read만 대상이라 이것만으로는 부족

핵심 비동기 경로는 서버 변경 통지도 받지만, 닫기 후 열기 일관성은 열 때 서버 속성을 직접 확인해 보장

속성 버전과 재검증

열 때마다 속성을 다시 확인하고, 경쟁은 속성 버전 번호와 변경 대기로

속성 조회(STAT) 전 속성 버전 번호(seq) 기록

↓

응답 도착, 그 사이 변경(mutation)이 겹쳤나?

↓

겹쳤으면 폐기하고 재조회

↓

두 번 폐기되면 변경 대기(mutation barrier)

↓

기존 변경 종료 대기 후 재조회

체계

- 열 때마다 속성 조회로 서버 속성 강제 재검증, 달라졌으면 페이지 캐시 무효화
- 속성 버전 번호, 변경 대기, 캐시 검증 TTL 1초
- TTL은 관측 지연의 상한이 아님
- 아이노드별 뮤텍스(mutex)로 재검증끼리 직렬화
- 쓰기 후 속성은 현재 속성만 갱신

핵심 변경 대기는 새 변경 등록을 잠시 막고 기존 변경이 끝나기를 기다려 재조회가 끝없이 밀리는 것을 방지

재검증 세부 규칙

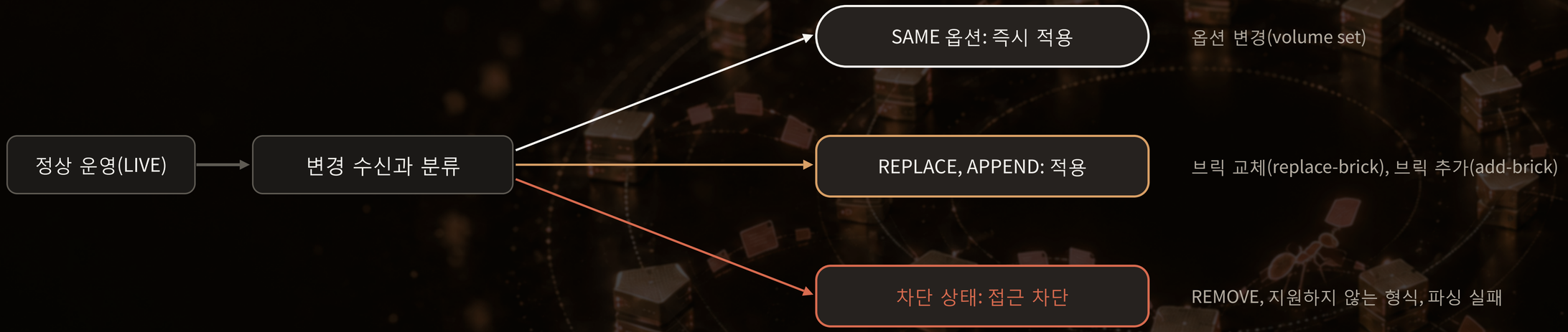
삭제된 열린 아이노드, 조회, 쓰기 후 속성, TTL

삭제된 열린 아이노드	gfid로 속성 조회가 안 되므로 일반 묶음의 파일 디스크립터를 고정해 속성 조회로 재검증
조회 (LOOKUP)	초기화만 담당, 갱신은 버전 번호를 가진 속성 조회가 담당
쓰기 응답의 쓰기 후 속성	크기 갱신, 붙여쓰기 위치 불일치는 나중에 반드시 무효화
cache_attr와 TTL	재검증 성공 시 갱신, 검증 TTL 1초는 관측 지연 상한이 아님
dentry 이름 재검증	항목 TTL 1초, 부정(negative) dentry는 캐시하지 않음, d_seq 확인과 stamp를 자식 잠금 아래에서, 완료

핵심 시험 결과와 실제 클러스터 기록은 노트와 6장 '남은 일과 검증 범위'의 시험 결과 표에

마운트 중 구성 변경

volfile이 바뀌면 먼저 종류를 나눔



원칙

진행 중인 FOP는 시작 시점 구성으로 끝내고 새 FOP만 새 구성 적용, 구성 버전 기록의 구성 버전으로 진행과 적용을 분리

핵심 마운트를 내리지 않고 구성을 바꾸는 것이 목표, 안전하지 않은 변경은 막는 쪽으로

변경 종류별 구현 상태

SAME, REPLACE, APPEND, READDIRP

SAME	옵션 변경(volume set)은 새 구성을 파생해 즉시 적용, 실제 클러스터에서 read-subvolume 변경 반영
REPLACE	브릭 교체(replace-brick)는 새 요청 막기, 대기 중 잠금 요청 처리, 잠금과 파일 디스크립터 넘겨받기 순으로 적용
APPEND	브릭 추가(add-brick)는 새 디렉터리 분배, 레이아웃 재조정(fix-layout) 전 기존 디렉터리는 mkdir -ESTALE, rename과 rmdir은 부분 변경 위험이 있어 완료 뒤 수행
READDIRP	열거 재개 시 토폴로지 버전 변화를 -ESTALE로 알리고 rewinddir로 회복

핵심 이전 브릭에 잡힌 잠금을 넘기지 않으면 잠금이 사라지므로 REPLACE는 순서가 전부

차단 후 상태와 적용 후 제한

지원하지 않는 변경은 접근을 막고, 적용 뒤에도 남는 제한

차단 후 상태

- read는 -EIO
- 매핑 접근은 페이지 폴트(fault)
- 되돌릴 수 있는 변경은 원래 구성 복원 후 접근 재개
- 조용한 오동작보다 차단이 안전

적용 후 제한

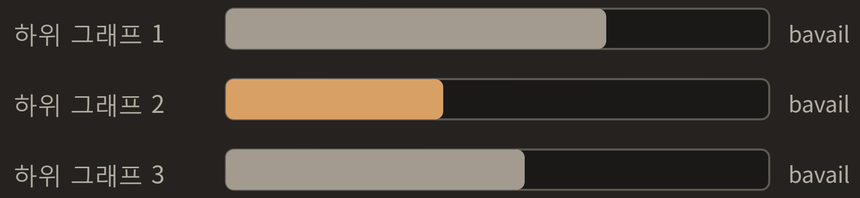
- 교체 위치에 원격 잠금 기록이 있거나 불명확한 상태(UNKNOWN)인 아이노드
- 해당 아이노드의 기존 열기는 -EBADF
- close는 허용
- 새로 열면 새 구성으로

핵심 구성 버전 기록의 구성 버전이 있어야 "어떤 클라이언트가 이전 구성을 보고 있는지"를 알 수 있음

statfs 집계 산술

AFR은 구조체 전체 선택, EC는 단위 정규화 뒤 필드별 최소

AFR 그룹



bavail이 가장 작은 하위 그래프의 구조체 전체를 선택

필드마다 따로 최소를 고르지 않음, 기존 AFR 규칙 그대로

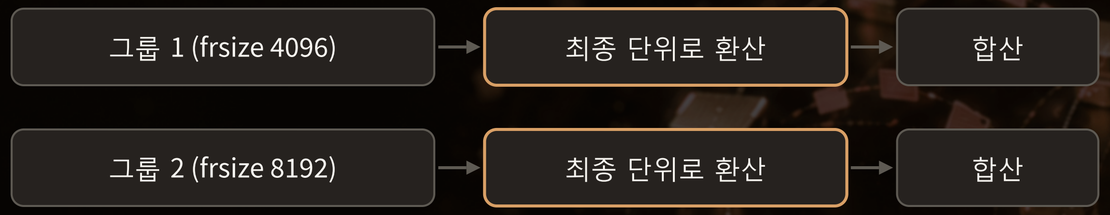
EC 그룹

- 먼저 블록 단위(frsize)를 하나로 맞춤
- 그 다음 필드마다 최소를 고름
- blocks, bfree, bavail은 데이터 조각 수 k를 곱함
- 기존 EC 규칙 그대로

핵심 같은 "최소"라도 AFR은 하위 그래프 단위, EC는 필드 단위

그룹 사이 집계

DHT는 최종 단위를 먼저 정하고 그룹별로 환산해 합산



최종 단위를 먼저 정하고 그룹마다 환산한 뒤 더함
기존 구현은 누적합을 다시 내리는 방식이라 순서에 의존
그래서 기존 재환산과 값이 다를 수 있음

순서가 문제인 이유

- 정수 나눗셈은 내림이라 나누는 시점마다 조금씩 버려짐
- 그룹마다 한 번씩 환산하고 더하면 버림이 그룹 수만큼, 누적해서 한 번 나누면 한 번
- 어느 쪽이 맞는지보다 어느 쪽이 기존과 같은 값을 내는지가 기준

핵심 df가 보여 주는 숫자가 FUSE 클라이언트와 달라지면 사용자는 버그로 봄

statfs에서 틀리기 쉬운 계산

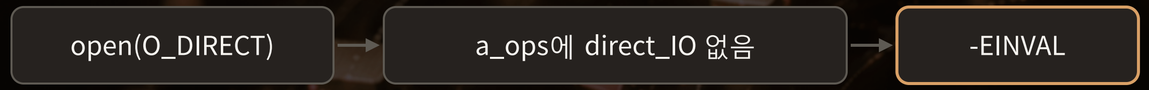
중간 곱 오버플로우, 내림 순서, frsize wrap

중간 곱 오버플로우	blocks 곱하기 frsize가 u64를 넘을 수 있음 mul_u64_u64_div_u64로 128비트 중간값을 씀
내림 순서	먼저 내리면 1블록 손실 `floor(1001*2*4096/8192)=1001, floor(1001*4096/8192)*2=1000`
frsize wrap	응답 메시지의 u64 frsize가 단위 상한 2^32을 넘을 수 있음 넘으면 잘못된 하위 그래프로 보고 배제
포화 위치	포화는 최종 단위에서만 중간 단계에서 포화시키면 값이 어긋남

핵심 검증 결과와 남은 확인(혼합 frsize, quota-deem-statfs)은 6장 '남은 일과 검증 범위'에

O_DIRECT 에뮬레이션

direct_IO가 없으면 6.12 VFS는 O_DIRECT open을 -EINVAL로 거부



dd oflag=direct, 데이터베이스, 백업 도구가 아예 열지 못함

대안 1: direct_IO를 직접 구현

- 클러스터 조립 RMW와 오염 표시(poison) 처리까지 복제해야 함
- 위험이 커서 택하지 않음

대안 2: 플래그 무시

- attr TTL 1초 동안 기존 내용을 읽게 됨
- 캐시 우회라는 의미가 깨져서 택하지 않음

핵심 API로 허용하는 것과 데이터가 지나는 경로는 별개로 봐야 함

에뮬레이션 구현

FMODE_CAN_ODIRECT와 범위 무효화로 흉내

open: 정규 파일에 FMODE_CAN_ODIRECT

↓

read_iter: 속성 강제 재검증

↓

미리 읽기(readahead)와 버퍼 입출력 읽기 (IOCB_DIRECT 내림)

↓

범위 무효화

↓

write_iter: 쓴 뒤 범위 무효화

왜 IOCB_DIRECT를 내리는가

- 범용(generic) 버퍼 입출력 경로를 그대로 재사용하기 위해
- 호출자가 같은 kiocb를 계속 보기 때문에 끝나면 되돌림
- 쓰기는 즉시 쓰기라 서버에 바로 도달
- 미리 읽기는 요청 범위로 제한

핵심 무효화는 비대칭, 읽기는 전후로 하고 쓰기는 끝난 뒤에

에뮬레이션의 한계

에뮬레이션으로 극복하지 못한 것들

무효화 실패	읽기 전이면 오류, 읽은 뒤면 다음 재검증 때 반드시 무효화하도록 표시
복사	사용자 버퍼와 폴리오(folio) 사이, 그리고 네트워크 경로의 복사는 그대로 남음, 무복사(zero-copy)가 아님
이중 버퍼링	페이지 캐시를 거치므로 응용 버퍼와 페이지 캐시에 같은 데이터가 두 번
정렬	정렬 제약은 FUSE direct_io처럼 없음, 단 구현과 지키는 범위는 다른 길
mmap 폴트	O_DIRECT 파일 디스크립터의 mmap 폴트에는 미리 읽기(readahead)가 없음

핵심 검증 과정에서 배운 것(비교 대상 설계)과 시험 결과는 6장 '남은 일과 검증 범위'에

남은 일과 검증 범위

확인한 것, 아직 안 맞는 것, 남은 일

검증 도구와 환경

무엇으로 확인했는가

KUnit	분리 모듈 36개, DHT 레이아웃, AFR 트랜잭션, EC 조각 버전, 파일 디스크립터 수명, volfile, statfs, dentry, mmap
실제 클러스터 상호운용	3대의 RL10(Rocky Linux 10, 커널 6.12) 서버에서 기존 v11.2 서버 대상 테스트
회귀 시험	변경마다 볼륨과 옵션 조합 10모드 전부 (dht, rep3, ec, 닫기 후 열기 등)
lockdep	잠금 순서 위반 경고 0을 통과 조건으로
POSIX 호환성	xfstests 진행 중, pjdfstest는 다음 순서로 POSIX 의미 검사

핵심 순수 로직은 KUnit, 기존 서버 대응은 실제 클러스터

시험 결과

이슈별 주요 수치

이슈	결과	비고
프로토콜 함정	7건 수정 뒤 v11.2 서버와 핸드셰이크, 디렉터리 읽기(READDIRP) 정상	
AFR 동시 쓰기	포팅 전 40%와 44% 분기, 포팅 후 100회 분기 0	포팅 중 측정에서는 10회 중 3회 분기
EC 비트 슬라이스	스칼라가 기존 11.2 조각 6/6 일치, SIMD는 selftest	
동시 붙여쓰기	두 노드 400/400, md5 일치, 찢어짐 0 (포팅 전 394/400)	inodelk on, lifecycle on
닫기 후 열기	실제 클러스터 재검증 8/8, 변경 대기 경쟁 시험 통과	
구성 변경	REPLACE 실제 클러스터 시험 통과, read-subvolume 변경 반영	
statfs	KUnit 11/11, frsize가 같은 하위 그래프로 이루어진 6개 볼륨 blocks 일치	혼합 frsize 등가는 미확인
O_DIRECT	KUnit 6/6, 실제 클러스터 10/10 (비교 대상과 실패 주입 포함)	
dentry 재검증	KUnit 3, 실제 클러스터 dentry_reval_check.sh	
공유 쓰기 mmap	KUnit mmap 5, attr 12, 실제 클러스터 10/10, 막아야 할 경우 4가지도 확인	

핵심 열 가지 이슈 모두 각 행의 범위에서 재현 시험을 통과, 실제 클러스터 시험과 기존 서버 대조로 확인

검증 공백

코드가 없는 것과는 다른 문제

lockdep 미포함 경로	EC 마운트, umount -f, 재연결 중 정리, 두 마운트 아이노드 잠금 경쟁
statfs	혼합 frsize 실제 클러스터 등가는 미확인, quota-deem-statfs 미포팅
시그널 중단	일반 RPC 대기의 시그널 중단 지원은 제한적, DHT SETLKW는 취소를 요청하지만 서버의 최종 판정 대기

핵심 확인한 범위와 아직 못 본 범위를 나눠 두었고, 못 본 범위가 다음 시험의 목록

아직 안 맞는 것

미해결 결함, 코드는 있지만 끝나지 않은 것

정족수 옵션	옵션 구문 분석과 DHT 조회 집계까지, 쓰기 경로 정책 통합은 미완
SETLKW 취소	기존 연결 정리 중 대기 잠금 취소가 아직 없음
REMOVE 볼륨 축소	데이터 이동 중 I/O 처리와 함께 아직 없음
ctime.noatime	구문 분석 완료, 같은 아이노드 두 번째 열기의 atime 미갱신이 남음
read-consistency 옵션	호환 표시일 뿐, AFR 읽기 후보 선택은 옵션과 무관하게 항상 수행

핵심 dentry 이름 재검증과 공유 쓰기 mmap은 이 목록에서 빠짐, 둘 다 완료

현재 제약 사항

아직 구현하지 않은 것, 서버에서 처리하는 것, 마운트 시점에 거부하는 것

아직 구현하지 않은 기능

- copy_file_range: 서버 측 오프로드(offload) 없이 읽기와 쓰기로 대체
- 스플릿 브레인(split-brain) 진단 인터페이스 없음, -EIO만 반환
- quota-deem-statfs, snapview-client, cloudsync 대응은 추후 지원

서버에서 처리

- read-only: 서버 xlator가 쓰기를 차단
- 클라이언트에 별도 read-only 구현 없음

마운트 시점 거부

- shard, cdc: 데이터와 프로토콜 형식을 바꿈
- arbiter, thin-arbiter: 읽기 후보 판단을 오인하게 함
- 대응 구현 전까지 volfile에 있으면 마운트 거부

핵심 조용한 오동작보다 마운트 시점에서 명시적 거부

옵션 기본값

k_mount_opts.h의 현재 기본값

inodelk	활성	AFR 쓰기 직렬화
entrylk	활성	항목 트랜잭션 직렬화
lifecycle	활성	연결과 파일 디스크립터 수명 관리
read_consistency	활성	호환 표시
volfile_refresh	활성	구성 갱신
async	활성	전송 모드
EC 일관성	비활성	
flock_remote	비활성	원격 flock
mmap_write	비활성	공유 쓰기 매핑

핵심 옵션은 동작 방식의 선택이지 기능 유무가 아님, mmap_write 기본 활성화 전환은 결정 전

다음 할 일

FUSE 클라이언트 포팅의 남은 작업

glfs_mmap_write 기본 활성화 전환
한 릴리스 동안 비활성화로 두고 전환하는 안, 결정 전

shard, cdc, arbiter, thin-arbiter
대응 구현 뒤 마운트 시점 거부 해제

SETLKW
커널 문맥의 블로킹 대기 문제가 남아 있어 디버깅 중

256 KiB RPC 버퍼
장기 할당 실패 시 writepages 재시도 완화

REMOVE 볼륨 축소
데이터 이동 중 I/O 처리와 함께 구현

정족수 쓰기 경로 통합
옵션 구문 분석과 조회 집계에 이어 쓰기 경로 정책까지

quota-deem-statfs, snapview-client, cloudsync
기존 클라이언트 그래프의 대응 기능 구현

pjdfstest
POSIX 의미 검사, 진행 중인 xfstests 다음에 진행

핵심 dentry 재검증과 공유 쓰기 mmap이 끝나 남은 것은 이 여덟 가지

달성 범위

이번 포팅으로 확인된 것

1. TCP는 기존 v11.2 서버와 무수정 호환, RDMA는 커널 안에 다시 구현해 호환 검증
2. 클라이언트 분산 로직인 DHT, AFR, EC, 잠금을 커널 VFS로 다시 구현
3. 직렬화, 붙여쓰기, 닫기 후 열기, 구성 전환을 재현 시험으로 확인
4. dentry 재검증과 공유 쓰기 mmap까지 완료, 남은 것은 다음 할 일 목록
5. KUnit, 실제 클러스터 상호운용, lockdep 경고 0을 통과 조건으로 삼은 검증

핵심 서버를 그대로 둔 채 클라이언트만 커널로 옮기는 일은 이 범위 안에서 동작을 확인

핵심 교훈

세 가지

1. **프로토콜 호환의 기준은 사양 문서가 아니라 서버가 실제로 받는 것**
   사양 밖 규칙은 재현 시험이 통과할 때까지 수정

2. **커널 문맥의 제약이 설계를 정함**
   블로킹 대기 대신 논블로킹과 유한 재시도, 잠금 소유자는 커널 객체를 토큰화

3. **닫기 후 열기 일관성은 서버 통지에 기대지 않고 열 때 직접 확인**
   커널로 옮겨도 대기와 복사는 남고, 사라지는 것은 데몬 왕복

Q&A

감사합니다

potatogim@potatogim.net

github.com/potatogim