Sistema de Archivos Distribuido Basado en Etiquetas (RedCloud Files)

Mauricio Sunde Jiménez, Luis Alejandro Arteaga Morales
Proyecto de Sistemas Distribuidos
Enero 2026

***Resumen*—RedCloud Files es un sistema de archivos distribuido basado en etiquetas que implementa un modelo AP (Availability + Partition Tolerance) con consistencia eventual acotada. El diseño utiliza arquitectura peer-to-peer sin líder, replicación completa, y protocolos de gossip y anti-entropía para convergencia en decenas de segundos. Se analizan ocho aspectos del diseño: arquitectura, procesos, comunicación, coordinación, nombrado, consistencia, tolerancia a fallos y seguridad, demostrando trade-offs conscientes según el teorema CAP.**

***Index Terms*—sistemas distribuidos, consistencia eventual, replicación, gossip protocol, teorema CAP**

I. Arquitectura

A. Posicionamiento en el Teorema CAP

RedCloud Files implementa un modelo AP (Availability + Partition Tolerance) que prioriza disponibilidad sobre consistencia fuerte. Durante particiones de red, el sistema continúa aceptando operaciones en ambas particiones, tolerando divergencia temporal con garantía de convergencia eventual.

B. Arquitectura de Tres Capas

Controllers (Capa de Metadatos): Gestionan usuarios, autenticación, metadatos de archivos y mapeo a chunks físicos. Arquitectura peer-to-peer sin líder, estado replicado mediante log de operaciones.

Chunkservers (Capa de Almacenamiento): Almacenan chunks de datos inmutables (4MB) con checksums SHA256. Replicación completa entre todos los nodos, gestión de tombstones para prevenir resurrección de datos.

CLI (Capa de Cliente): Interfaz ligera sin estado, descubrimiento dinámico de servicios, failover automático entre controllers.

C. Decisiones de Diseño Fundamentales

- **Consistencia eventual sobre fuerte:** Lecturas potencialmente desactualizadas a cambio de disponibilidad continua
- **Replicación completa sobre particionamiento**
- **Sin líder sobre consenso:** Opera con cualquier número de nodos sin quorum
- **Inmutabilidad:** Chunks write-once/read-many simplifican replicación

II. Procesos

A. Tipos de Procesos

Controllers: N instancias replicadas ($N \geq 1$) sin estado compartido, peers equivalentes sin jerarquía, coordinación mediante replicación.

Chunkservers: M instancias ($M \geq 1$) con replicación completa, sin particionamiento de datos, cada nodo puede servir cualquier chunk.

Clientes: Instancia única por sesión, efímeros, sin estado persistente.

B. Concurrencia y Escalabilidad

Modelo basado en event loops asíncronos para alta concurrencia. Tareas periódicas en background: gossip (2s), anti-entropía (30s), descubrimiento de peers (30s), garbage collection.

Escalabilidad horizontal: agregar controllers aumenta capacidad de procesamiento, agregar chunkservers aumenta throughput I/O. Limitaciones: costo de sincronización, overhead de almacenamiento lineal, bound de convergencia crece con nodos.

III. Comunicación

A. Arquitectura Multi-Protocolo

REST/HTTP: Cliente-Controller para simplicidad y debugging. JSON para metadatos, multipart para binarios, Bearer tokens para autenticación.

gRPC: Servidor-Servidor para rendimiento. Serialización binaria Protocol Buffers, streaming bidireccional, multiplexación HTTP/2.

B. Patrones de Mensajería

Gossip Protocol: Diseminación epidémica. Cada nodo selecciona $K = 2$ peers aleatorios cada $T = 2s$, envía resumen de estado, peers solicitan información faltante. Convergencia $O(\log N)$ rondas, tolerante a pérdida de mensajes.

Anti-Entropy: Sincronización exhaustiva. Cada $T' = 30s$, nodo contacta 1 peer aleatorio, intercambian resúmenes completos, transfieren diferencias bidireccionales. Garantiza convergencia eventual.

Streaming Bidireccional: Transferencia incremental de chunks.

IV. Coordinación

A. Diseño Sin Líder

Arquitectura leaderless sin algoritmos de consenso. Elimina punto único de falla, no requiere quorum, pero requiere resolución determinística de conflictos.

B. Ordenamiento Causal

Vector Clocks: Cada nodo mantiene vector V_i donde $V_i[j]$ es conocimiento sobre eventos del nodo j. Relaciones: $V_1 < V_2$ (happens-before), $V_1 \parallel V_2$ (concurrencia). Controllers incrementan contador al generar operación, mergen al recibir: $V[k] = \max(V_{local}[k], V_{remote}[k])$.

Operaciones Idempotentes: Identificadores únicos previenen efectos duplicados, permiten reintentos seguros.

Operaciones Diferidas: Cola para operaciones con dependencias no satisfechas, reintentos cuando dependencias lleguen.

C. Resolución de Conflictos

Last-Write-Wins (LWW): Para campos escalares, timestamp más reciente gana, tie-breaker con `controller_id`. Determinística pero puede perder escrituras concurrentes.

Merge Semántico: Para conjuntos (tags), union de operaciones concurrentes, tombstones para eliminaciones.

Reconciliación Post-Partición: Detección via gossip $\rightarrow$ resumen via anti-entropía $\rightarrow$ identificación de divergencias $\rightarrow$ transferencia bidireccional $\rightarrow$ aplicación con resolución determinística $\rightarrow$ convergencia. Bound: "decenas de segundos" en LAN.

V. Nombrado y Localización

A. Descubrimiento DNS

DNS como único mecanismo de service discovery. Aliases: `controller` (todos los controllers activos), `chunkserver` (todos los chunkservers activos). Round-robin DNS para distribución básica. Caché local con TTL (30s refresh, 10min retention) como fallback.

B. Identificación

UUIDs para todos los recursos: controllers (`hostname_uuid`), users/files/chunks/operations (`uuid`), API keys (`dfs_uuid`). Generación descentralizada sin coordinación, colisiones estadísticamente imposibles (2^{128}).

C. Localización

Metadata: Cualquier controller puede servir queries. Sin sharding, convergencia al mismo estado, cliente contacta cualquier controller disponible.

Chunks: Cualquier chunkserver puede servir cualquier chunk. Sin particionamiento, replicación completa elimina necesidad de routing.

VI. Modelo de Consistencia y Estrategia de Replicación

A. Consistencia Eventual

Propiedades: (1) Safety - sin actualizaciones, todos convergen; (2) Liveness - actualizaciones eventualmente visibles; (3) Bounded Convergence - convergencia acotada tras reconexión.

NO garantiza: Linearizability, Causal Consistency Global, Read-Your-Writes.

B. Replicación de Metadatos

Operation Log: Cambios de estado generan operations con tipo, payload, timestamp, vector clock. Replicación mediante gossip (rápido, $O(\log N) \times 2s \approx 10 - 20s$) + anti-entropía (exhaustivo, hasta 30s).

Operaciones: `USER_CREATED`, `API_KEY_UPDATED`, `FILE_METADATA_UPDATED`.

C. Replicación de Datos

Full Replication: Chunk escrito a todos los chunkservers disponibles, faltantes reciben via anti-entropía. Sin distinción primaria/secundaria.

Write: descubrir via DNS $\rightarrow$ enviar en paralelo $\rightarrow$ esperar ACK mayoría $\rightarrow$ retornar éxito. Read: descubrir $\rightarrow$ seleccionar arbitrariamente $\rightarrow$ reintentar si falla.

Integridad: Checksums SHA256 verificados en escritura, lectura y anti-entropía.

D. Gestión de Eliminaciones

Tombstones: Eliminación genera tombstone replicado indicando "eliminado en timestamp T". Anti-entropía propaga tombstones, nodos eliminan copia antigua. Persisten indefinidamente (GC conservativo).

VII. Tolerancia a Fallos y Resiliencia

A. Clasificación

Crash (Fail-Stop): Detección por timeouts, failover a nodos restantes, recuperación via catch-up.

Particiones de Red: Detección cuando gossip falla, ambas particiones operan independientemente, healing automático tras reconexión.

Bizantinos: NO TOLERADO. Sistema asume fail-stop model.

B. Disponibilidad

Sistema disponible con: ≥ 1 controller operativo, ≥ 1 chunkserver operativo, cliente puede contactar controller. No requiere quorum ni mayoría.

Degradación gradual: fallos de $N - 1$ nodos reducen throughput pero mantienen operación.

C. Recuperación

Controller: Estado preservado - carga database, descubre peers, ejecuta gossip/anti-entropía, converge. Estado perdido - solicita ALL operations via anti-entropía, reconstruye estado completo. Tiempo: proporcional a log ($\sim 30 - 120s$).

Chunkserver: Escanea directorio, reconstruye índice, ejecuta anti-entropía, recibe chunks faltantes. Tiempo: proporcional a volumen (minutos-horas).

D. Particiones

Durante split-brain: ambas particiones aceptan operaciones, divergencia temporal. Tras healing: gossip reestablece contacto, anti-entropía transfiere operations bidireccional, resolución LWW, convergencia. Consecuencias: writes perdedoras descartadas, posible "rollback" observable.

VIII. Modelo de Seguridad

A. Superficie de Ataque

Vectores: Interceptación de tráfico (sin TLS), acceso no autorizado a almacenamiento (sin cifrado en reposo), denegación de servicio (sin rate limiting), inyección de operations maliciosas (peers sin autenticación criptográfica), enumeración de usuarios.

B. Autenticación

Usuarios: Username + password $\rightarrow$ bcrypt hash $\rightarrow$ API key generado. API key como bearer token. Rotación en cada login, sin expiración automática.

Entre Servicios: SIN autenticación mutua. Confianza implícita en red overlay. Apropiado solo para entornos de confianza.

C. Autorización

Control basado en propiedad: usuario solo accede archivos propios. Verificación: $owner\_id = user\_id$. Sin roles, sin compartir, sin permisos granulares.

D. Limitaciones

Sin cifrado end-to-end, sin anonimización, retención indefinida. Apropiado para: entornos de desarrollo/pruebas, LANs privadas. Inapropiado para: datos sensibles, ambientes multi-tenant públicos.

IX. Conclusiones

Fortalezas: Disponibilidad continua durante particiones, simplicidad operacional, escalabilidad horizontal, auto-reparación, sin punto único de falla.

Limitaciones: Consistencia débil, replicación completa costosa, seguridad básica, sin persistencia durable, resolución de conflictos simple.

Apropiado para: Desarrollo/staging, LANs privadas, almacenamiento temporal, clusters pequeños-medianos.

Inapropiado para: Datos críticos (financieros/médicos), multi-tenant público, geo-distribución WAN, escala masiva.

El diseño implementa correctamente modelo AP del teorema CAP, evita consenso distribuido, utiliza epidemic algorithms con propiedades teóricas esperadas, y satisface definición formal de eventual consistency con bound temporal especificado.