Curso
Herramientas de Computación en la Nube

Clase 3

Febrero 7 de 2026

Motores de almacenamiento

Profesor

Alvaro Mauricio Montenegro Díaz, Ph.D.
Universidad de la Sabana

Febrero 2026


LSM
Escrituras secuenciales
Memtable
SSTable
LSM Tree
Memtable
SSTable
Bloom Filler
BaLol | Eficiente para
Disco Docmental
Clave-valor
B-Tree
WAL
OLAP
OLTP
OLTP
Lecturas y escritutas por clave
Consultas agregadas masivas
BaLol |
Bacterals Scile
Durabilidad
Recuperación
Recuperación
Replicación
Analitica y compresion
Motores de almacenamiento
Amplificación | OLTP vs OLAP | Principios Físicos

1. Introducción

Los motores de almacenamiento constituyen el núcleo de los sistemas de datos, ya que determinan **cómo se organizan los datos en almacenamiento persistente**, cómo se accede a ellos y qué compromisos se asumen entre rendimiento, confiabilidad, consumo de recursos y complejidad operativa. Las decisiones a este nivel tienen un impacto directo en la escalabilidad, la latencia, la durabilidad y la facilidad de mantenimiento de los sistemas.

2. Objetivos fundamentales del almacenamiento

Un motor de almacenamiento debe proporcionar mecanismos eficientes para:

- **Insertar, actualizar y eliminar datos**
- **Recuperar datos por clave o por rango**
- **Mantener durabilidad frente a fallos**
- **Soportar concurrencia y recuperación**
- **Optimizar el uso de disco y memoria**

Estos objetivos suelen estar en tensión entre sí, lo que obliga a priorizar según los patrones de acceso predominantes.

3. Principios físicos del almacenamiento

Antes de analizar estructuras como LSM o B-tree, es necesario comprender que todo motor de almacenamiento está condicionado por la física del medio persistente.

Los discos magnéticos tradicionales favorecen lecturas y escrituras secuenciales, penalizando accesos aleatorios.
Las unidades SSD y NVMe reducen esta penalización, pero introducen nuevos fenómenos como desgaste por escritura y necesidad de gestión interna de bloques.

Todo motor de almacenamiento es, en esencia, una estrategia para optimizar:

- Localidad espacial
- Localidad temporal
- Minimización de accesos aleatorios

- Balance entre latencia y throughput

Las estructuras de datos no se diseñan en abstracto: se diseñan en función del comportamiento del hardware.

4. Índices como estructura central

Un índice es una estructura de datos que permite **localizar registros de manera eficiente** sin recorrer todo el conjunto almacenado. Los motores de almacenamiento se diferencian principalmente por **el tipo de índices que utilizan** y por cómo estos índices se mantienen y persisten en disco.

Los índices imponen un costo adicional en escritura y almacenamiento, pero reducen significativamente el costo de lectura. Por ello, su diseño debe alinearse con el patrón de uso del sistema.

5. Almacenamiento basado en logs (append-only)

5.1. Principio básico

El almacenamiento basado en logs organiza los datos como un **registro secuencial de escrituras**. Cada operación se añade al final del log, evitando escrituras aleatorias en disco.

5.2. Ventajas

- Escrituras secuenciales rápidas.
- Simplicidad conceptual.
- Buen aprovechamiento del ancho de banda del disco.
- Facilita la replicación y la recuperación.

5.3. Limitaciones

- Lecturas ineficientes sin índices auxiliares.
- Acumulación de versiones obsoletas de los datos.
- Necesidad de procesos de limpieza y compactación.

6. Índices Hash

6.1. Funcionamiento

Los índices hash mantienen una estructura en memoria que asocia claves con posiciones en el log. Permiten **búsquedas rápidas por clave exacta**.

6.2. Ventajas

- Acceso muy rápido por clave.
- Implementación simple.

6.3. Limitaciones

- No soportan consultas por rango.
- Escalan mal cuando el índice no cabe completamente en memoria.
- Dependencia fuerte del tamaño de la RAM.

7. Concepto de amplificación

Uno de los marcos analíticos más importantes en motores modernos es el concepto de amplificación.

Write Amplification

Cantidad de datos realmente escritos en disco en relación con los datos lógicos solicitados por la aplicación.

Read Amplification

Número de accesos físicos necesarios para satisfacer una lectura lógica.

Space Amplification

Cantidad adicional de espacio requerido debido a versiones temporales, compactación o índices.

Los motores LSM tienden a reducir la latencia de escritura a costa de mayor write y space amplification.

Los B-tree tienden a ofrecer lecturas más predecibles con menor read amplification.

Comprender estas métricas permite comparar motores de manera rigurosa.

8. Motores LSM (Log-Structured Merge Trees)

8.1. Idea central

Los motores LSM combinan:

- Escrituras secuenciales en estructuras en memoria (memtables).
- Persistencia periódica en disco en forma de **tablas ordenadas inmutables**.
- Procesos de **compactación** que fusionan archivos y eliminan datos obsoletos.

8.2. Componentes clave

- **Memtable**: estructura ordenada en memoria.
- **SSTables**: archivos ordenados e inmutables en disco.
- **Bloom filters**: reducen accesos innecesarios a disco.
- **Compaction**: reorganiza y consolida datos.

8.3. Ventajas

- Excelente rendimiento en escrituras.
- Buen soporte para grandes volúmenes de datos.
- Escrituras mayoritariamente secuenciales.

8.4. Costos y trade-offs

- Lecturas pueden requerir consultar múltiples archivos.
- Compactación consume recursos de CPU, I/O y espacio temporal.
- Mayor latencia en lecturas si la compactación no está bien ajustada.

9. Árboles B y B+

9.1. Estructura

Los árboles B organizan los datos en **bloques balanceados**, optimizados para acceso en disco. Cada nodo contiene múltiples claves, reduciendo la altura del árbol y el número de accesos a disco.

9.2. Características

- Soporte eficiente para búsquedas por clave y por rango.

- Actualizaciones in-place (sobrescritura).
- Estructura estable y ampliamente utilizada.

9.3. Ventajas

- Rendimiento predecible en lecturas.
- Excelente soporte para consultas por rango.
- Menor sobrecarga en lecturas comparado con LSM.

9.4. Limitaciones

- Escrituras más costosas (accesos aleatorios).
- Mayor complejidad para replicación basada en logs.
- Fragmentación y reequilibrado pueden afectar el rendimiento.

10. Comparación LSM vs B-tree

Aspecto	LSM	B-tree
Escrituras	Muy rápidas	Más lentas
Lecturas	Variables	Predecibles
Rangos	Buen soporte	Excelente soporte
Uso de disco	Mayor (duplicación temporal)	Más eficiente
Compacción	Necesaria	No requerida

11. OLTP vs OLAP como eje de decisión

Los motores de almacenamiento no deben analizarse en aislamiento, sino en función del tipo de carga dominante.

11.1. OLTP (Online Transaction Processing)

- Escrituras frecuentes y pequeñas.
- Consultas por clave exacta.
- Baja latencia crítica.
- Concurrencia alta.

Motores típicos:

- B-tree (PostgreSQL, MySQL, MariaDB InnoDB)

- LSM (RocksDB, Cassandra)

OLAP (Online Analytical Processing)

- Consultas agregadas masivas.
- Escaneo de grandes volúmenes.
- Acceso columnar preferido.
- Latencia menos crítica que throughput.

Motores típicos:

- Columnar (ClickHouse, DuckDB)
- Parquet + motores distribuidos (Spark)

El error arquitectónico común es intentar usar el mismo motor para ambos perfiles.

La elección depende del equilibrio deseado entre rendimiento de lectura y escritura, latencia y consumo de recursos.

12. Motores columnar y almacenamiento orientado a columnas

12.1. Organización

Los datos se almacenan **por columnas en lugar de por filas**, lo que permite leer solo los atributos necesarios para una consulta.

12.2. Ventajas

- Excelente compresión.
- Alto rendimiento en consultas analíticas.
- Reducción significativa de I/O para agregaciones.

12.3. Limitaciones

- Escrituras transaccionales costosas.
- No adecuado para cargas OLTP intensivas.
- Requiere procesos de ingesta y reorganización.
- Compresión y codificación

13. Los motores modernos emplean técnicas de:

- Codificación delta.
- Diccionarios.
- Run-length encoding.
- Bit-packing.

Estas técnicas reducen el tamaño en disco y mejoran el rendimiento al disminuir el volumen de datos transferidos.

14. Recuperación y durabilidad

Para garantizar durabilidad, los motores utilizan:

- **Write-Ahead Logging (WAL)**.
- Checkpoints periódicos.
- Reproducción del log tras fallos.

El diseño del mecanismo de recuperación afecta directamente el tiempo de reinicio y la confiabilidad del sistema.

15. Recuperación y consistencia

El Write-Ahead Logging no solo garantiza durabilidad; también permite mantener propiedades de atomicidad e integridad en presencia de fallos.

El proceso típico implica:

1. Registrar la intención de cambio en el WAL.
2. Confirmar persistencia del log.
3. Aplicar cambios en estructuras principales.
4. En caso de fallo, reconstruir el estado mediante replay.

El diseño del WAL afecta:

- Tiempo de reinicio.
- Coste de replicación.
- Complejidad operativa.

En sistemas distribuidos, el WAL suele convertirse en el mecanismo base de replicación.

16. Impacto arquitectónico

La elección del motor de almacenamiento influye en:

- Latencia y throughput.
- Modelo de replicación.
- Estrategias de backup.
- Escalabilidad horizontal.
- Costes operativos.

No es una decisión aislada, sino un elemento central de la arquitectura de datos.

17. EJEMPLOS DE TECNOLOGÍAS POR TIPO DE ALMACENAMIENTO

17.1. Motores basados en B-Tree

- PostgreSQL
- MySQL InnoDB
- SQLite
- Oracle Database

Uso típico:

- OLTP clásico
- Sistemas transaccionales bancarios
- ERP

17.2. Motores basados en LSM Tree

- RocksDB
- LevelDB
- Apache Cassandra
- ScyllaDB
- HBase
- CockroachDB (híbrido)

Uso típico:

- Alta tasa de escrituras
- Sistemas distribuidos
- Logs de eventos
- Sistemas de mensajería

17.3. Motores Columnar

- ClickHouse
- DuckDB
- Snowflake
- Amazon Redshift
- Google BigQuery

Uso típico:

- OLAP
- BI
- Analítica masiva

17.4. Motores basados en Hash

- Redis
- Memcached
- DynamoDB (modo key-value)

Uso típico:

- Caché
- Sesiones
- Acceso por clave exacta

17.5. Sistemas híbridos (Row + Column)

- PostgreSQL + extensión columnar
- SQL Server (Columnstore Index)
- SAP HANA

17.6. Sistemas especializados en almacenamiento distribuido

- Ceph
- MinIO
- RustFS (S3-compatible)
- Amazon S3 (object storage)

18. Síntesis

Los motores de almacenamiento no son implementaciones intercambiables; representan decisiones de ingeniería profundamente ligadas al patrón de carga, al medio físico y a los requisitos de confiabilidad.

Un motor LSM prioriza escrituras secuenciales y escalabilidad horizontal, aceptando costos de compactación.
Un B-tree prioriza estabilidad en lecturas y rangos, aceptando escrituras más costosas.
Un motor columnar optimiza agregaciones masivas, sacrificando eficiencia transaccional.

La elección correcta no depende de popularidad tecnológica, sino de comprender:

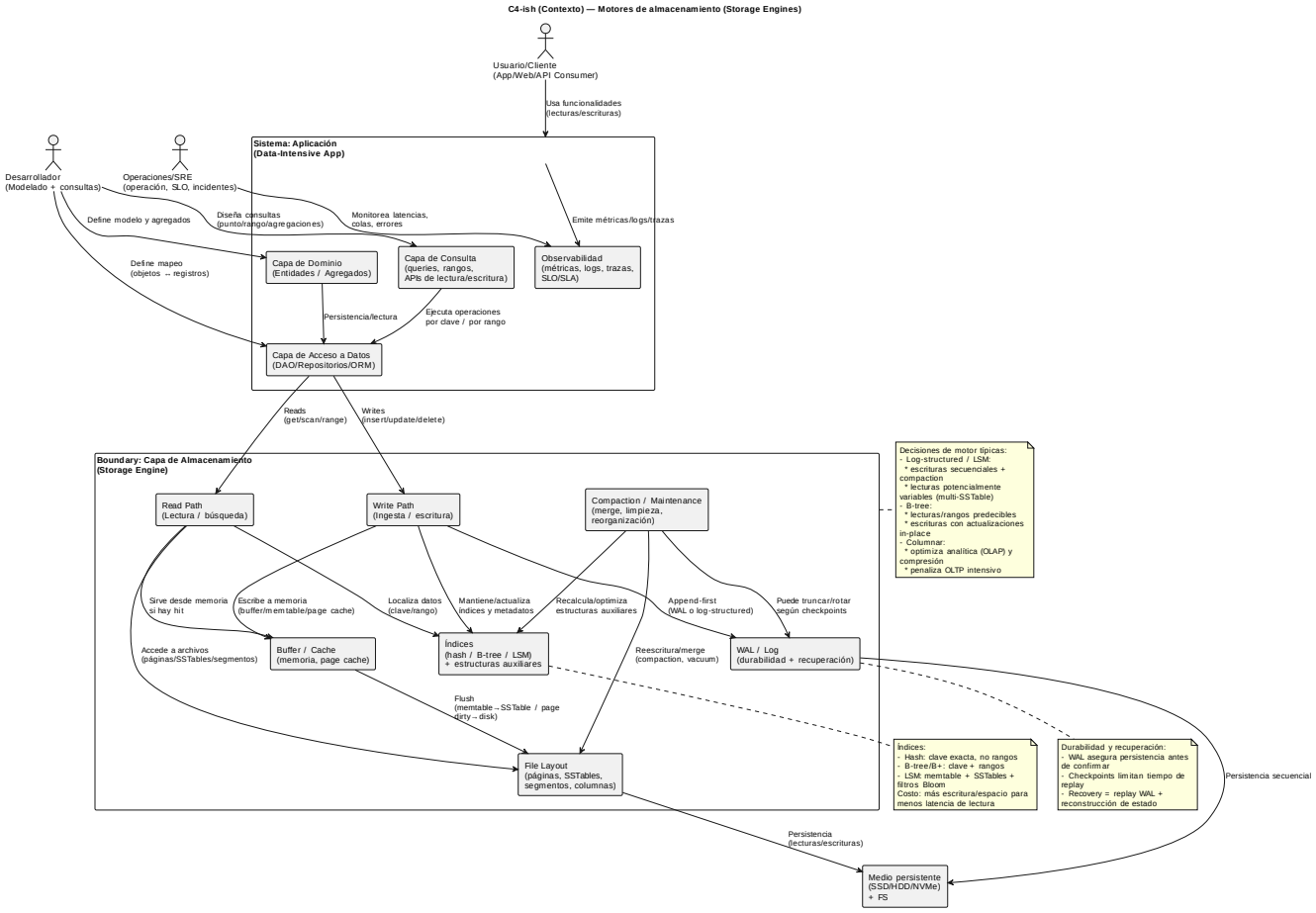
- la distribución real de lecturas y escrituras,
- la tolerancia a latencia,
- el crecimiento esperado,
- y los costos operativos asociados.

El motor de almacenamiento es una decisión arquitectónica de primer orden, no un detalle interno del sistema.

En resumen, una configuración eficiente podría ser:

Tipo	Motor	Uso ideal
B-Tree	PostgreSQL	OLTP estable
LSM	Cassandra	Escrituras intensivas
Columnar	ClickHouse	OLAP
Hash	Redis	Caché

19. Contexto: Hardware-Motor de almacenamiento-Carga



Fuente: Código en formato puml generado por ChatGPT

20. Diccionario de términos usados

Conceptos fundamentales de almacenamiento

Motor de almacenamiento (Storage Engine)

Subsistema responsable de cómo los datos se organizan físicamente en disco, cómo se indexan y cómo se garantiza su durabilidad.

Persistencia

Capacidad de conservar datos más allá del ciclo de vida del proceso o del sistema en memoria.

Bloque (Block)

Unidad mínima de lectura/escritura en disco. En motores tradicionales suele ser 4KB, 8KB o mayor.

Página (Page)

Unidad lógica de almacenamiento gestionada por el motor, generalmente alineada con bloques físicos.

Buffer Pool
Área en memoria donde el motor mantiene páginas recientemente accedidas para evitar lecturas físicas repetidas.

Page Cache
Caché administrada por el sistema operativo para almacenar bloques de disco.

Hardware y medio físico

HDD (Hard Disk Drive)
Disco magnético rotatorio. Alta penalización en accesos aleatorios, mejor desempeño en accesos secuenciales.

SSD (Solid State Drive)
Dispositivo de almacenamiento basado en memoria flash. Reduce latencia en accesos aleatorios.

NVMe (Non-Volatile Memory Express)
Protocolo de acceso a SSD diseñado para alta paralelización y baja latencia sobre PCIe.

Latencia
Tiempo entre la solicitud de I/O y su finalización.

Throughput
Cantidad de datos procesados por unidad de tiempo.

Estructuras de datos internas
B-Tree
Árbol balanceado optimizado para lecturas por rango y acceso ordenado. Minimiza altura del árbol.

B+Tree
Variante donde todos los datos están en hojas y nodos internos solo contienen claves.

LSM Tree (Log-Structured Merge Tree)
Estructura optimizada para escrituras secuenciales. Usa:

- Memtable (en memoria)
- SSTables (inmutables en disco)
- Proceso de compactación

Memtable
Estructura en memoria donde se almacenan escrituras antes de persistirse en disco.

SSTable (Sorted String Table)
Archivo ordenado e inmutable utilizado en motores LSM.

Bloom Filter
Estructura probabilística que permite verificar si una clave probablemente no existe, reduciendo lecturas innecesarias.

Índice
Estructura auxiliar que acelera búsquedas sobre claves.

Hash Index
Índice basado en función hash, eficiente para igualdad exacta, no para rangos.

Hash Function
Función que transforma una clave en un valor numérico para distribuirla en una estructura de almacenamiento.

Conceptos de amplificación

Write Amplification
Cantidad adicional de escritura física respecto a la escritura lógica solicitada.

Read Amplification
Número de accesos físicos necesarios para satisfacer una lectura lógica.

Space Amplification
Espacio adicional consumido por estructuras auxiliares o versiones temporales.

Durabilidad y recuperación

WAL (Write-Ahead Log)
Registro secuencial donde se anotan cambios antes de aplicarlos en estructuras principales.

Durabilidad
Garantía de que un cambio confirmado no se pierde ante fallos.

Checkpoint
Proceso que sincroniza el estado en memoria con el disco.

Replay
Reaplicación de registros del log tras un fallo.

Compaction
Proceso en LSM que fusiona archivos y elimina versiones obsoletas.

Tipos de carga

OLTP (Online Transaction Processing)
Sistema orientado a transacciones pequeñas, frecuentes y de baja latencia.

OLAP (Online Analytical Processing)
Sistema orientado a consultas analíticas masivas y agregaciones.

Escaneo secuencial
Lectura continua de bloques consecutivos.

Acceso aleatorio
Lectura o escritura en posiciones no contiguas.

Formatos analíticos

Almacenamiento columnar
Organización de datos por columnas en lugar de filas, optimizando agregaciones.

Parquet
Formato columnar ampliamente usado en data lakes.

ORC
Formato columnar optimizado para Hadoop.

21. Bibliografía especializada

Núcleo obligatorio

- Martin Kleppmann – *Designing Data-Intensive Applications*
- Goetz Graefe – *Modern B-Tree Techniques*
- Patrick O'Neil et al. – *The Log-Structured Merge-Tree (LSM-Tree)* (1996)

LSM moderno

- RocksDB documentation (Meta)
- Cassandra Architecture (Apache)

Columnar

- Stonebraker et al. – *C-Store: A Column-Oriented DBMS*

- ClickHouse whitepaper

WAL y recuperación

- ARIES Recovery Algorithm (Mohan et al., IBM Research)