数据库存储引擎的原理与实现

王思成

Jul 14, 2026

存储引擎是数据库的 I/O 心脏，它决定了数据如何持久化、如何被检索，以及如何在并发和故障场景下保持一致性。本文将沿着「数据结构—事务模型—工程实践」三条主线，依次剖析基于 B+Tree 的传统行存引擎、基于 LSM-Tree 的写优化引擎以及它们在真实系统中的调优与实现细节。

1　存储引擎的定义与边界

存储引擎可抽象为「数据结构 + 事务语义 + 持久化策略」三元组。数据结构负责组织磁盘上的记录和索引；事务语义包括 ACID 保证及 MVCC、锁、回滚段等并发控制机制；持久化策略则涵盖 WAL、Checkpoint、Double-write 等崩溃恢复手段。不同的引擎在三元组上做出不同取舍，从而适应 OLTP 还是 OLAP、读多写少还是写多读少、SSD 还是 HDD 等差异化场景。

2　存储引擎核心抽象

页是数据库与磁盘交互的最小单位，通常为 4 KB 或 8 KB。Buffer Pool 或 Block Cache 在内存中缓存热点页，以降低随机 I/O。记录格式决定了行数据如何编码、如何溢出到溢出页。以 InnoDB Compact 格式为例，变长字段长度列表、NULL 标志位、记录头信息与真实列值依次排列；当单行超过 8 KB 时，溢出部分被存入独立的溢出页，主记录仅保留 20 字节指针。

索引类型分为主键索引与二级索引、聚簇索引与非聚簇索引。聚簇索引的叶子节点直接存放完整行数据，因此主键查询可一次 I/O 完成；二级索引仅存储主键值，回表时再通过主键索引定位行记录。事务四大隔离级别对存储引擎提出不同要求：可重复读需要 MVCC 快照与 ReadView；可串行化则需要 Next-Key Lock 防止幻读；所有级别都需要回滚段保存旧版本以支持回滚与一致性读。

3　基于 B+Tree 的存储引擎

B+Tree 由 B-Tree 演化而来，所有键值均存储在叶子节点，内部节点仅存放索引键与子节点指针，极大提升了范围扫描的顺序 I/O 效率。Bw-Tree 进一步在内存中采用 delta chain 与 cache-line sized node，以减少写时复制开销。

节点分裂与合并是 B+Tree 动态维持平衡的核心操作。当插入使节点键数超过阶数 M 时，调用 split_node 将节点一分为二，并把中间键上移；删除导致节点键数低于 M/2 时，调用 merge_node 从兄弟节点借键或合并。以下 Go 风格伪码展示了分裂过程：

```
func split_node(n *BTreeNode) (*BTreeNode, Key) {
```

```
    mid := len(n.keys) / 2
    right := newNode()
    right.keys = append(right.keys, n.keys[mid+1:]...)
    right.children = append(right.children, n.children[mid+1:]...)
    n.keys = n.keys[:mid]
    n.children = n.children[:mid+1]
    return right, n.keys[mid]
}
```

上述代码将节点 n 的后半部分键与子节点复制到新节点 right，并返回 right 与中间键。时间复杂度为 O(M)，空间复杂度为 O(M)，其中 M 为节点阶数。

InnoDB 的页目录（Page Directory）与 Page Header 共同管理槽位与记录。Page Header 记录本页的层级、记录数量、最近修改 LSN 等元信息；Page Directory 以稀疏索引方式指向槽位，便于二分查找。插入、删除、范围扫描的性能受节点分裂概率影响，随机写入时写放大约为 1.3 – 2.0 倍。

锁与 MVCC 的协同保证了高并发下的隔离性。InnoDB 的 Next-Key Lock 由记录锁与间隙锁组成，可防止幻读；ReadView 记录活跃事务列表，查询时仅可见小于 ReadView 最小事务 ID 的版本；回滚段链表保存旧版本记录，支持回滚与一致性读。

4 基于 LSM-Tree 的存储引擎

LSM-Tree 通过将随机写转换为顺序写来提升写入吞吐。内存中的 MemTable 采用 SkipList 或红黑树组织，写操作先落 WAL 再更新 MemTable；当 MemTable 达到阈值时转为 Immutable 并刷盘生成 SSTable。读路径需依次检查 MemTable、Immutable MemTable 及所有层级的 SSTable。Bloom Filter 先快速判断键是否可能存在，随后 Block Index 定位 Data Block，最后读取记录。合并策略决定写放大与读放大：Size-Tiered 将同层文件合并；Leveled 按键范围分层，写放大近似 T×ln(N/M)，其中 T 为合并阈值，N 为总数据量，M 为单层容量。

LevelDB、RocksDB、HBase、Apache Cassandra 均基于 LSM-Tree，但合并策略与文件格式各异。RocksDB 支持 Universal、Leveled、FIFO 三种合并策略，并通过 Column Family 实现多租户隔离。

5 其他经典引擎速览

堆表将记录按插入顺序追加存储，PostgreSQL 与 MyISAM 采用此结构，适合顺序追加但不适合点查。列式存储将同列数据连续存放，Parquet、ORC、ClickHouse MergeTree 均采用此布局，极大提升分析型查询的 I/O 效率。面向日志的引擎如 SQLite WAL 与 Apache Kafka Segment，将数据以不可变日志形式追加，天然支持顺序写与时间旅行查询。

6 事务与恢复

WAL 是崩溃恢复的基石。LSN 单调递增标识日志记录；Checkpoint 将脏页刷盘并截断日志；Group Commit 批量提交以降低 fsync 次数。ARIES 恢复算法分为 Analysis、Redo、Undo 三阶段：Analysis 扫描日志重建 Dirty Page Table 与 Transaction Table；Redo 重放所有已提交与未提交操作；Undo 回滚未提交事务。

Crash-Safe 实现要点包括 InnoDB 的 Double-write buffer 与 RocksDB 的 WAL + MANIFEST 双重保障。

7 工程实践与性能调优

内存参数直接影响缓存命中率，innodb_buffer_pool_size 通常设为物理内存的 50% – 70%；rocksdb_block_cache 控制 SSTable 块缓存大小。并发控制上，MVCC 提供乐观读而行锁提供悲观写，二者可组合使用。压缩算法 Snappy、Zstd 与字典编码、RLE 可在 CPU 与 I/O 之间权衡。硬件亲和方面，Direct I/O 绕过页缓存，io_uring 与 SPDK 降低内核开销，PMem 提供持久化内存语义。Benchmark 套路包括 sysbench oltp、YCSB、db_bench 与 TPC-C，可分别衡量吞吐、延迟与一致性。

8 动手实现一个迷你存储引擎

目标是实现一个支持 KV 读写与崩溃恢复的最小引擎。技术栈可选 Go 或 Rust，底层可采用 BoltDB 风格的 B+Tree 或 LevelDB 风格的 LSM-Tree。代码结构分为 page.go 负责页管理、btree.go 实现 B+Tree 逻辑、wal.go 处理 WAL 序列化与 fsync、recovery.go 实现 ARIES 风格恢复。单元测试中可通过 kill -9 模拟宕机，随后重启校验数据一致性。

以下为简化版 WAL 追加与恢复伪码：

```
func appendWAL(l *WAL, op Operation) error {
   buf := encode(op)
   if _, err := l.file.Write(buf); err != nil {
      return err
   }
   return l.file.Sync()
}

func recover(l *WAL) error {
   for {
      op, err := decode(l.file)
      if err == io.EOF {
         break
      }
      apply(op) // 重放操作
   }
   return nil
}
```

appendWAL 先将操作编码后顺序写入文件，再调用 Sync 确保持久化；recover 顺序读取并重放所有操作，直至文件结束。时间复杂度为 O(N)，N 为日志记录数。

9 未来趋势

存算分离架构如 Aurora、PolarDB、TiDB TiFlash 将计算与存储解耦，支持独立扩展。新硬件 CXL、ZNS SSD、Persistent Memory 正在改变页大小与 I/O 模型。AI 辅助调参工具 OtterTune、Bao、SageDB 通过机器学习自动寻找最优参数组合。

B+Tree 适合读多写少场景，LSM-Tree 适合写多读少场景。存储引擎的本质是数据结构、并发控制与持久化策略的协同。理解其原理后，可在真实业务中按需选型并进行参数微调。

10 附录

推荐阅读包括《Transaction Processing: Concepts and Techniques》、《Designing Data-Intensive Applications》、RocksDB Wiki 与 MySQL Internals Manual。配套源码仓库可访问 GitHub 下的 yourname/mini-lsm。勘误与更新日志将持续维护。