MySQL 性能优化手册

一、版本适配与基础层优化

1. 硬件选型与版本差异

CPU：高并发场景优先选择多核 CPU（如 16 核 Intel Xeon），但需注意 MySQL 5.7 对超过 32 核的利用率下降明显，而 8.0 版本通过线程调度优化可提升 20% 并发支撑能力。例如电商秒杀服务器建议使用 24 核 CPU 搭配 8.0 版本。

内存：遵循“热数据 1.5 倍原则”，8.0 版本因元数据缓存优化，建议内存预留比 5.7 多 10%-15%。例如热数据 8GB 时，8.0 版本建议配置 16GB 内存。

磁盘：OLTP 场景必须使用 SSD。NVMe SSD 随机 IOPS 超 10 万，适合存放日志文件（加速事务提交）；历史数据可存放于 SATA SSD。8.0 版本的双写缓冲优化对 SSD 更友好。

2. 操作系统参数配置（Linux）

文件描述符：

```
# 临时生效
ulimit -n 65535
# 永久生效（编辑/etc/security/limits.conf）
echo "mysql soft nofile 65535" >> /etc/security/limits.conf
echo "mysql hard nofile 65535" >> /etc/security/limits.conf
```

同时在 MySQL 配置文件（my.cnf）中同步设置：

```
[mysqld]
open_files_limit = 65535
```

内核参数：

```
# 编辑/etc/sysctl.conf
net.ipv4.tcp_tw_recycle = 1  # 加速 TIME_WAIT 回收（5.7 推荐，8.0 可结合 tcp_autocorking）
net.ipv4.tcp_tw_reuse = 1    # 允许重用 TIME_WAIT 连接
vm.swappiness = 5           # 8.0 建议 5-10，降低内存交换频率
net.ipv4.tcp_max_syn_backlog = 65535  # 应对突发连接请求
sysctl -p  # 生效配置
```

二、核心参数调优（5.7 与 8.0 对比）

1. 内存参数

InnoDB 缓冲池：

5.7 配置：

```
innodb_buffer_pool_size = 10G
innodb_buffer_pool_instances = 8  # 实例数=CPU 核心数/2
```

8.0 配置：

```
innodb_buffer_pool_size = 10G  # 实例数自动优化，超 128GB 内存时建议设为 16
```

并发连接数：

5.7 计算：max_connections = (可用内存 - 缓冲池)/2MB，如 16GB 内存、缓冲池 10G 时设为 800。

8.0 计算：max_connections = (可用内存 - 缓冲池)/1.7MB，可提高至 1000，并新增：

```
connection_memory_limit = 100M  # 防止单连接内存泄露
```

2. 日志参数

事务日志：

5.7：innodb_log_file_size = 1G，最大支持 4G。

8.0：高并发场景可设为 2G-4G：

```
innodb_log_file_size = 2G
innodb_log_files_in_group = 2
```

查询缓存：

5.7 高写场景必禁：

```
query_cache_size = 0
query_cache_type = 0
```

8.0 已移除，无需配置。

三、索引优化深度解析

1. 索引类型与原理

B-Tree 索引：支持全值匹配、范围查询（如 BETWEEN）、前缀匹配（如 LIKE 'prefix%'），但需遵循最左前缀原则。联合索引(a,b,c)仅对 a=? AND b=?或 a=? AND b>?有效。

哈希索引：适合等值查询（=、IN），但不支持范围查询。InnoDB 自适应哈希索引自动优化高频访问数据。

覆盖索引：查询列全部包含在索引中，避免回表。例如：

```
CREATE INDEX idx_covering ON users(name, age) INCLUDE(email);
```

2. 优化实践

索引选择性：计算公式为选择性=不同值数量/总记录数，优先选择选择性高的列（如用户 ID 而非性别）。

避免索引失效：

禁止前导通配符（`LIKE '%keyword'`）。

避免隐式类型转换（如 `WHERE user_id = '123'`，user_id 为整数）。

慎用 `NOT IN`、`!=`等否定操作符。

分页查询优化：

低效写法（偏移量大时）

```
SELECT * FROM orders ORDER BY id LIMIT 10000, 20;
```

优化写法（延迟关联+覆盖索引）

```
SELECT * FROM orders INNER JOIN (SELECT id FROM orders ORDER BY id LIMIT 10000, 20) AS tmp USING(id);
```

四、查询优化核心工具

1. EXPLAIN 执行计划分析

关键指标解读：

type 列：性能从优到差顺序为 `system > const > eq_ref > ref > range > index > ALL`。出现 `ALL` 需警惕全表扫描。

rows 列：预估扫描行数，越小越好。

Extra 列：`Using filesort` 表示文件排序，`Using temporary` 表示临时表，需优化。

示例分析：

```
EXPLAIN SELECT * FROM users WHERE name = '张三' AND age > 30 ORDER BY created_at DESC;
```

若 type 为 range 且 rows 较大，需为 name 和 age 添加联合索引。

2. 慢查询日志

开启配置：

```
slow_query_log = ON
slow_query_log_file = /var/log/mysql/slow.log
long_query_time = 1  # 超过 1 秒的查询记录
```

分析工具：

```
pt-query-digest /var/log/mysql/slow.log > slow_analysis.txt
```

五、硬件与存储引擎专项优化

1. 硬件层优化

CPU 多核利用：设置 innodb_thread_concurrency = CPU 核心数×2，如 16 核 CPU 设为 32，充分利用并行处理能力。

内存带宽提升：采用双通道 / 四通道内存架构，选择低延迟 DDR4-3200MHz 内存，提升数据读写速度。

存储分层：

日志文件（redo log）存放于 NVMe SSD，事务提交速度提升 3 倍以上。

冷数据（如 3 年前订单）迁移至 SATA HDD，降低成本。

2. InnoDB 引擎优化

I/O 刷新方式：

```
innodb_flush_method = O_DIRECT  # 避免内核缓存，直接读写磁盘（Linux 推荐）
```

锁机制优化：

```
innodb_lock_wait_timeout = 10  # 缩短锁等待时间，避免长事务阻塞
```

六、高可用架构设计

1. 主从复制优化

半同步复制：

```
# 主库配置
plugin_load_add = "semisync_master.so"
rpl_semi_sync_master_enabled = 1
# 从库配置
plugin_load_add = "semisync_slave.so"
rpl_semi_sync_slave_enabled = 1
```

GTID 复制：

```
gtid_mode = ON
enforce_gtid_consistency = ON
```

2. 组复制（Group Replication）

三节点集群配置：

```
# 所有节点
plugin_load_add = "group_replication.so"
group_replication_group_name = "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
group_replication_start_on_boot = OFF
group_replication_local_address = "node1:33061"  # 各节点 IP+端口
```

自动故障转移：结合 MySQL Router 实现读写分离。

七、性能监控与持续优化

1. 实时监控工具

Percona Monitoring Plugins：

```
# 安装
wget https://repo.percona.com/apt/percona-release_latest.generic_all.deb
dpkg -i percona-release_latest.generic_all.deb
apt-get update
apt-get install percona-monitoring-plugins
```

监控指标：

QPS（Queries Per Second）：反映数据库负载。

Buffer Pool 命中率：理想值 > 95%。

InnoDB 行锁等待次数：需趋近于 0。

2. 定期维护策略

索引碎片整理：

```
ALTER TABLE users ENGINE=InnoDB;  # 重建表以整理索引碎片
```

统计信息更新：

```
ANALYZE TABLE orders;  # 优化器使用最新统计信息生成执行计划
```

八、典型场景优化案例

1. 电商秒杀场景

硬件配置：24 核 CPU + 64GB 内存 + NVMe SSD，8.0 版本支撑 20% 更高并发。

查询优化

优化前（全表扫描）

```
SELECT * FROM products WHERE stock > 0 LIMIT 1;
```

优化后（覆盖索引）

```
CREATE INDEX idx_stock ON products(stock) INCLUDE(id, name);
```

锁优化：使用乐观锁（`UPDATE products SET stock = stock -1 WHERE id=1 AND stock >0`）减少锁竞争。

2. 金融交易场景

日志安全配置：

```
innodb_flush_log_at_trx_commit = 1  # 事务提交时强制刷盘
sync_binlog = 1                     # binlog 同步策略
```

备份策略：

```
# 每日全量备份
mysqldump -u root -p --all-databases --single-transaction > full_backup.sql
# 每小时增量备份
mysqlbinlog --start-datetime="2024-01-01 00:00:00"
/var/log/mysql/binlog.000001 > incremental.sql
```

九、版本新特性应用（MySQL 8.0+）

1. 隐藏索引

测试索引必要性：

```
ALTER TABLE users ALTER INDEX idx_email INVISIBLE;  # 临时禁用索引
-- 观察性能变化后决定是否删除
ALTER TABLE users DROP INDEX idx_email;
```

2. 降序索引

优化倒序排序：

```
CREATE INDEX idx_desc ON logs(timestamp DESC);
SELECT * FROM logs ORDER BY timestamp DESC LIMIT 10;  # 直接使用索引排序
```

3. 函数索引

基于表达式优化：

```
CREATE INDEX idx_upper_email ON users(UPPER(email));
SELECT * FROM users WHERE UPPER(email) = 'ADMIN@EXAMPLE.COM';
```

十、性能优化检查清单

1. 基础检查

检查 EXPLAIN 执行计划，确保无 ALL 类型和 Using filesort。

监控 slow_query_log，优化超过 1 秒的查询。

2. 索引检查

使用 sys.schema_unused_indexes 视图删除无用索引。

确保联合索引符合最左前缀原则。

3. 配置检查

验证 innodb_buffer_pool_size 是否为物理内存的 60%-70%。

确认 max_connections 计算是否合理。

4. 硬件检查

使用 iostat 监控磁盘 I/O，确保 await 值 < 10ms。

通过 htop 检查 CPU 使用率，避免长期 > 80%。

通过以上系统化的优化策略和工具，可显著提升 MySQL 数据库的吞吐量、响应时间和稳定性，满足从 OLTP 到 OLAP 的多样化业务需求。