高性能 Mysql

存储引擎

- MyISAM
  - 崩溃修复: 执行表修复可能导致部分数据丢失
  - 表锁：支持读取时并发插入(CONCURRENT INSERT)
  - 索引：支持全文索引，支持延迟更新索引（每次修改完成后，不会立刻将修改的索引数据写入磁盘）
- innoDB
  - 支持事务
  - 支持自动崩溃恢复
  - 锁： 支持表锁和行级锁（相较于表锁有更高的并发度但是消耗更多的系统资源），通过 MVCC 提高并发度
  - 支持4种隔离级别：默认为 REPEATABLE READ(可重复读)，并且通过间隙锁防止幻读的出现
  - 索引： 支持聚簇索引（数据根据主键顺序进行排列），聚簇索引多主键查询有很高的性能，二级索引必须包含主键列

性能分析与测试

- 通过性能剖析进行优化
  - 基于执行时间的性能分析
  - 基于等待的性能分析
- 理解性能剖析
  - 值得优化的查询： 不要对一个占总响应时间 5% 的查询进行优化
  - 异常情况: 执行次数较少，但速度慢，占总响应时间不突出
  - 丢失的时间： 测量工具存在系统误差，导致 cpu 实际执行的时间 > 测得的时间
  - 被隐藏的细节： 平均响应时间会隐藏掉很多的细节，分析时应结合直方图，百分比，标准差等指标一起进行分析
  - mysql 性能剖析工具推荐： pt-query-digest
- 对应用程序进行性能剖析
  - 应用程序可能存在的性能问题
    - 外部资源，比如调用了 HTTP 请求
    - 需要处理大量的数据，比如分析超大的 XML 文件
    - 在循环中执行昂贵的操作，比如滥用正则表达式
    - 使用了低效的算法，比如使用了暴力搜索算法
    - 应用程序性能剖析工具推荐: New Relic
- 剖析 Mysql 查询
  - 剖析服务器负载
    - 慢查询日志
      - 开销最低，精度最高的·查询时间的工具，开启慢查询日志带来的磁盘 I/O 消耗可以忽略不计，而空间消耗可以通过定期 rotation 日志解决
      - 分析方法
        - 通过慢查询日志查询
        - 当权限不足无法在服务器查看时
          - pt-query-digest --processlist
          - 先通过 tcpdump 将网络数据包保存到磁盘，然后使用 pt-query-digest --type=tcpdump
      - 分析过程
        - 自顶向下分析
          - 使用 pt-query-digest --explain 分析慢查询日志，通过 V/M（离差指数） 列以及执行计划一般都能发现性能较低的查询
  - 剖析单条查询
    - Show Profile
      - 功能: 在服务器上执行的所有语句，都会测量其耗费的时间和其他一些查询执行状态变更相关的数据
      - 使用方法 : 1. set profiling = 1; 2.select * from x; 3.show profiles;
      - tip: profile 告诉我们在哪个部分耗费了最多的时间，但是并不会告诉我们为什么会这样
    - show status
      - 功能： 返回一些计数器，包括服务器级别的全局计数器（show global status），和基于会话级别的计数器（show status），部分全局级别的计数器也会存在 show status 中
      - 使用方法: 1.flush status; 2.select * from x; 3. show status where variable like 'Handler%' pr variable like 'Created%'
      - tip: show status 大部分结果都只是一个计数器，可以显示某些活动如读索引的频繁程度，但无法给出消耗了多少时间。最有用的计数器包括句柄计数器，临时文件和表计数器。使用 explain 虽然也可以获得大部分相同的信息，但 explain 只是估计的结果
    - performane_schema
      - 书中未详细说明，好奇通过 performance_schema 如何进行分析可查看 https://www.cnblogs.com/cchust/p/5061131.html
      - tip: 开启 performance_schema 会有较大性能损耗, ≈10%
  - 诊断间歇性问题
    - 间歇性问题往往需要花费更多的时间，此时需要在问题发生的地方通过观察资源的使用情况，并尽可能地测量数据，尽量不要采用试错的方式
    - 单条查询还是服务器问题
      - 所有程序变慢，又突然变好，每一条查询都变慢了，那么可能不是慢查询的原因。如果服务器整体运行没有问题，只有某条查询偶尔变慢，那么将注意力放到这条特定的查询上
      - show global status 持续观察 thread_running, thread_connected. queries。当每秒查询数（thread_running）下跌，而其他两个至少一个出现尖刺则说明服务器可能存在问题。
  - 其他剖析工具
    - user_statistics 表
      - percona 和 maridb 支持: SET GLOBAL userstat=1;
    - strace
      - 使用 strace 可以调查系统调用的情况, 其中使用 strace -cfp $(pidof mysqld) 可以查看调用的时间，当使用 strace 时会使 mysql 变慢
  - 总结
    - 1.定义性能最有效的方法是响应时间
    - 2.优化基于高质量，全方位，完整的测量
    - 3.测量的最佳点是应用程序
    - 4.完整的测量需要大量的数据，所以需要剖析工具
    - 5.大量的剖析工具只能统计 cpu 工作的时间，而不能统计 io 等待的时间，所以等待分析是很好的补充
    - 6.等优化的成本大于优化带来的提升时，停止优化

schema 与数据类型优化

- 选择优化的数据类型
  - 1.更小更好：选择正确存储数据的最小数据类型
  - 2.简单就好：简单的数据类型通常需要更少的 cpu 周期。比如整型比字符串操作代价更低
  - 3.尽量避免 Null：null 会使索引占用更多的存储空间，变得低效。所以尽可能设置默认值
- 整数类型
  - 为 INT 类型指定宽度对大多数应用是没有意义的，它不会限制值的合法范围，只是规定了 Mysql 的一些交互工具显示字符的个数
- 实数类型
  - DECIMAL 相比 DOUBLE 和 FLOAT 需要额外的空间和计算开销，所以应该只在需要精确计算的时候使用 DECIMAL
- 字符串类型
  - varchar 需要使用1个或2个额外的字节保存字符串的长度，当 update 数据且页内没有足够空间时 innodb 会页分裂
  - 当字符串的最大长度比平均长度大很多时选择 varchar 更合适