TiDB 在个推丨掌握这两个调优技巧，让 TiDB 性能提速千倍！

← 查看全部博客

作者：大海 案例实践 2022-02-14

> *以下文章来源于个推技术实践 ， 作者大海*

个推与 TiDB 的结缘

作为一家数据智能企业，个推为数十万 APP 提供了消息推送等开发者服务，同时为众多行业客户提供专业的数字化解决方案。在快速发展业务的同时，公司的数据体量也在高速增长。随着时间的推移，数据量越来越大，MySQL 已经无法满足公司对数据进行快速查询和分析的需求，一种支持水平弹性扩展，能够有效应对高并发、海量数据场景，同时高度兼容 MySQL 的新型数据库成为个推的选型需求。

经过深入调研，我们发现热门的开源数据库 TiDB 不仅具备以上特性，还是金融级高可用、具有数据强一致性、支持实时 HTAP 的云原生分布式数据库。因此，我们决定将 MySQL 切换到 TiDB，期望**实现在数据存储量不断增长的情况下，仍然确保数据的快速查询，满足内外部客户高效分析数据的需求**，比如为开发者用户提供及时的推送下发量、到达率等相关数据报表，帮助他们科学决策。

完成选型后，我们就开始进行数据迁移。本次迁移 MySQL 数据库实例的数据量有数 T 左右，我们采用 TiDB 自带的生态工具 Data Migration (DM) 进行全量和增量数据的迁移。

- 全量数据迁移：从数据源迁移对应表的表结构到 TiDB，然后读取存量数据，写入到 TiDB 集群。
- 增量数据复制：全量数据迁移完成后，从数据源读取对应的表变更，然后写入到 TiDB 集群。

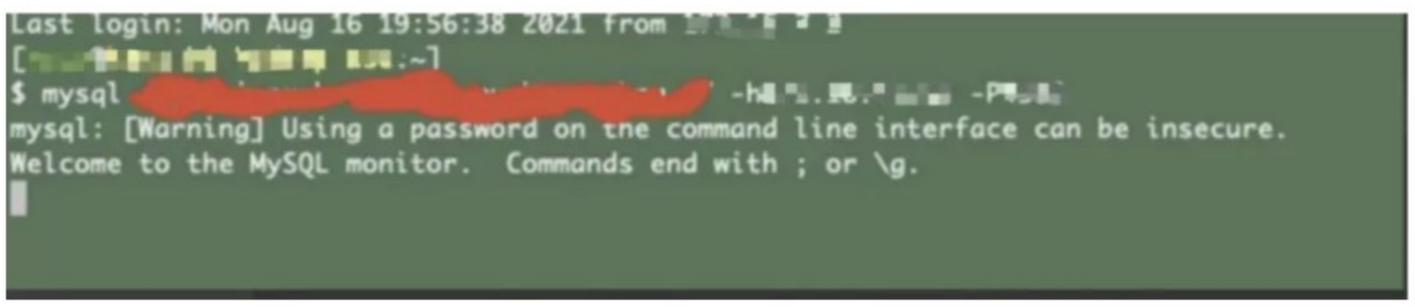
个推将 MySQL 数据迁移到 TiDB

当数据同步稳定之后，将应用逐步迁移到 TiDB Cluster。把最后一个应用迁移完成之后，停止 DM Cluster。这样就完成了从 MySQL 到 TiDB 的数据迁移。

注：DM 的具体配置使用详见官方文档。

陷入 TiDB 使用的“反模式”

然而，当应用全部迁移到 TiDB 之后，却出现了数据库反应慢、卡顿，应用不可用等一系列的问题。
如下图：



目录

登陆数据库时遇到卡顿

通过排查，我们发现有大量的慢 SQL 都是使用 load 导入数据的脚本。

慢 SQL 的导入耗时几十分钟

和业务方沟通后，我们发现有些导入语句就包含几万条记录，导入时间需要耗时几十分钟。
对比之前使用 MySQL，一次导入只需几分钟甚至几十秒钟就完成了，而迁到 TiDB 却需要双倍甚至几倍的时间才完成，几台机器组成的 TiDB 集群反而还不如一台 MySQL 机器。

——这肯定不是打开 TiDB 的正确姿势，我们需要找到原因，对其进行优化。

单个服务器负载过高

通过查看监控，发现服务器负载压力都是在其中一台机器上（如上图，红色线框里标注的这台服务器承担主要压力），这说明我们目前并没有充分利用到所有的资源，未能发挥出 TiDB 作为分布式数据库的性能优势。

打开 TiDB 的正确使用姿势

首先优化配置参数

具体如何优化呢？我们首先从配置参数方面着手。众所周知，很多配置参数都是使用系统的默认参数，这并不能帮助我们合理地利用服务器的性能。通过深入查阅官方文档及多轮实测，我们对 TiDB 配置参数进行了适当调整，从而充分利用服务器资源，使服务器性能达到理想状态。

下表是个推对 TiDB 配置参数进行调整的说明，供参考：

参数	调优前（默认设置）	调优后	备注
readpool.unified.max-thread-count	默认为机器CPU 数的80%（如机器为16核，则默认线程池大小为 12）	25	通常建议根据业务负载特性调整其CPU使用率为在线程池大小的60%～90%之间
storage.block-cache.capacity	系统总内存大小的45%	70GB	共享block cache的大小 默认值：系统总内存大小的 45% 建议：不超过系统内存的60%
raftstore.region-split-check-diff	region大小的 1/16	32MB	允许region数据超过指定大小的最大值
rocksdb.defaultcf.disable-auto-compactions	FALSE	TRUE	开启自动 compaction 的开关
raftstore.region-max-size	144MB	384MB	Region容量空间最大值，超过时系统分裂成多个Region
raftstore.region-split-size	96MB	256MB	分裂后新Region的大小，此值属于估算值
raftstore.split-region-check-tick-interval	10s	300s	检查region是否需要分裂的时间间隔，0 表示不启用
rocksdb.defaultcf.max-write-buffer-number	5	10	指最大memtable个数
rocksdb.writecf.max-write-buffer-number	5	10	指最大memtable个数
rocksdb.compaction-readahead-size	0	2MB	指异步Sync限速速率
readpool.storage.normal-concurrency	8	16	指处理普通优先级读请求的线程池线程数量 当8≤cpu num≤16时，默认值为cpu_num * 0.5；当cpu num大于8时，默认值为4；当cpu num大于16时，默认值为8，建议不超过50%

重点解决热点问题

调整配置参数只是基础的一步，我们还是要从根本上解决服务器负载压力都集中在一台机器上的问题。可是如何解决呢？这就需要我们先深入了解 TiDB 的架构，以及 TiDB 中表保存数据的内在原理。

在 TiDB 的整个架构中，分布式数据存储引擎 TiKV Server 负责存储数据。在存储数据时，TiKV 采用范围切分（range）的方式对数据进行切分，切分的最小单位是 region。每个 region 有大小限制（默认上限为 96M），会有多个副本，每一组副本，成为一个 raft group。每个 raft group 中由 leader 负责执行这个块数据的读 & 写。leader 会自动地被 PD 组件（Placement Driver，简称“PD”，是整个集群的管理模块）均匀调度在不同的物理节点上，用以均分读写压力，实现负载均衡。

TiDB 架构图

TiDB 会为每个表分配一个 TableID，为每一个索引分配一个 IndexID，为每一行分配一个 RowID（默认情况下，如果表使用整数型的 Primary Key，那么会用 Primary Key 的值当做 RowID）。同一个表的数据会存储在以表 ID 开头为前缀的一个 range 中，数据会按照 RowID 的值顺序排列。在插入（insert）表的过程中，如果 RowID 的值是递增的，则插入的行只能在末端追加。

当 region 达到一定的大小之后会进行分裂，分裂之后还是只能在当前 range 范围的末端追加，并永远仅能在同一个 region 上进行 insert 操作，由此形成热点（即单点的过高负载），陷入 TiDB 使用的“反模式”。

常见的 increment 类型自增主键就是按顺序递增的，默认情况下，在主键为整数型时，会将主键值作为 RowID ，此时 RowID 也为顺序递增，在大量 insert 时就会形成表的写入热点。同时，TiDB 中 RowID 默认也按照自增的方式顺序递增，主键不为整数类型时，同样会遇到写入热点的问题。

在使用 MySQL 数据库时，为了方便，我们都习惯使用自增 ID 来作为表的主键。因此，将数据从 MySQL 迁移到 TiDB 之后，原来的表结构都保持不变，仍然是以自增 ID 作为表的主键。这样就造成了批量导入数据时出现 TiDB 写入热点的问题，导致 Region 分裂不断进行，消耗大量资源。

对此，在进行 TiDB 优化时，我们从表结构入手，对以自增 ID 作为主键的表进行重建，删除自增 ID，使用 TiDB 隐式的 _tidb_rowid 列作为主键，将

```
create table t (a int primary key auto_increment, b int);
```

改为：

```
create table t (a int, b int)SHARD_ROW_ID_BITS=4 PRE_SPLIT_REGIONS=2;
```

通过设置 SHARD_ROW_ID_BITS，将 RowID 打散写入多个不同的 region，从而缓解写入热点问题。
此处需要注意，SHARD_ROW_ID_BITS 值决定分片数量：

```
SHARD_ROW_ID_BITS = 0 表示 1 个分片
SHARD_ROW_ID_BITS = 4 表示 16 个分片
SHARD_ROW_ID_BITS = 6 表示 64 个分片
```

SHARD_ROW_ID_BITS 值设置的过大会造成 RPC 请求数放大，增加 CPU 和网络开销，这里我们将 SHARD_ROW_ID_BITS 设置为 4。

PRE_SPLIT_REGIONS 指的是建表成功后的预均匀切分，我们通过设置 PRE_SPLIT_REGIONS=2，实现建表成功后预均匀切分 2^(PRE_SPLIT_REGIONS) 个 Region。

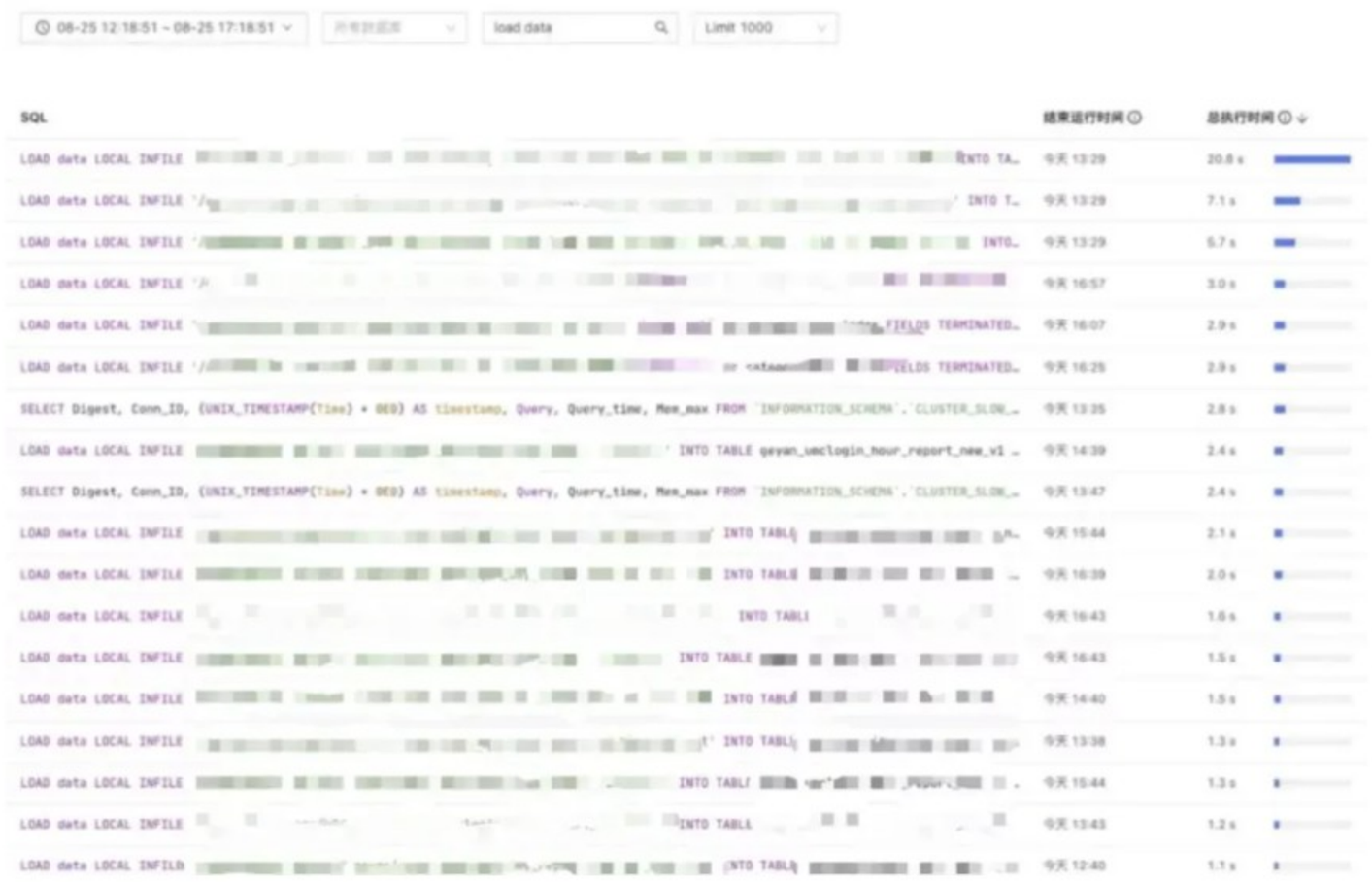
经验总结

- 以后新建表禁止使用自增主键，考虑使用业务主键
- 加上参数 SHARD_ROW_ID_BITS = 4，PRE_SPLIT_REGIONS=2

此外，由于 TiDB 的优化器和 MySQL 有一定差异，出现了相同的 SQL 语句在 MySQL 里可以正常执行，而在 TiDB 里执行慢的情况。我们针对特定的慢 SQL 进行了深入分析，并针对性地进行了索引优化，取得了不错的成效。

优化成果

通过慢 SQL 查询平台可以看到，经过优化，**大部分的导入在秒级时间内就完成了，相比原来的数十分钟，实现了数千倍的性能提升**。



慢 SQL 优化结果

同时，性能监控图表也显示，在负载高的时刻，是几台机器同时高，而不再是单独一台升高，这说明我们的优化手段是有效的，TiDB 作为分布式数据库的优势得以真正体现。

优化后，实现服务器负载均衡

总结

作为一种新型分布式关系型数据库，TiDB 能够为 OLTP（Online Transactional Processing）和 OLAP（Online Analytical Processing）场景提供一站式的解决方案。个推不仅使用 TiDB 进行海量数据高效查询，同时也展开了基于 TiDB 进行实时数据分析、洞察的探索。

性能优化