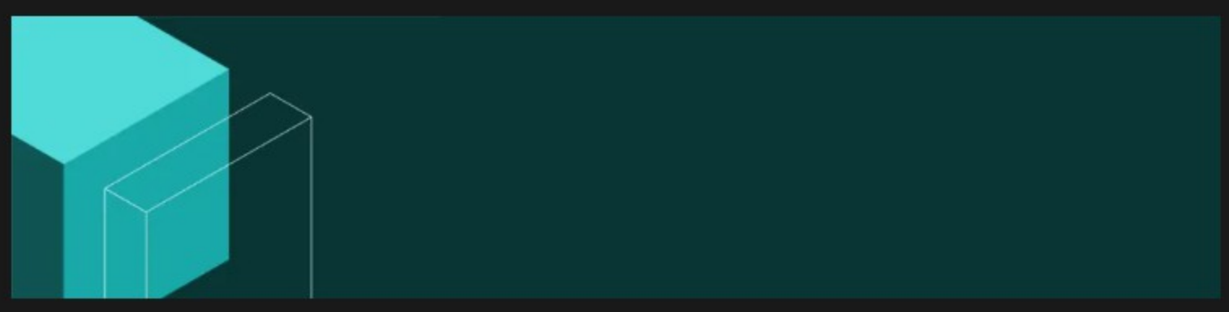
狂飙 50 倍 | TiDB DDL 框架优化深度解析

原创 TiDB 团队 PingCAP 2025年01月07日 19:30 上海



导读

在多租户大规模部署场景下，传统单机数据库的管理复杂性问题仍困扰着用户。

在 TiDB v6–v7 版本中，我们成功将 TiDB DDL 创建索引的性能提升了 10 倍，为用户带来了显著的体验改善。在 TiDB v8 版本中，我们对 TiDB DDL 语句执行流程进行了进一步的优化和重构，显著提升了框架的可扩展性和语句的执行效率，为未来实现 TiDB DDL 的真正分布式执行奠定了坚实基础。

本系列文章将从**原理解析、技术实现和应用实践**等角度**深入解析 TiDB DDL 框架如何提升系统的可扩展性，建表速度 50 倍提升（对比 TiDB v7.5），实现百万级别表的管理**。本文为系列文章的第一篇，将介绍 DDL 框架优化的效果和思路。

本文作者：居佳佳，郭铭浩，熊亮春

> **元数据**：用来定义并指导数据库系统如何解析与处理用户存在数据库中数据的数据。
> **General DDL 语句**：这类 DDL 语句的完成，只涉及元数据修改即可。
> **Reorg DDL 语句**：这类需要处理用户实际数据的 DDL 语句。

TiDB 在线 DDL 变更功能曾有效缓解了用户在数据库使用和演进过程中的痛点。然而，随着用户规模和数据量的不断增长，创建索引的性能瓶颈日益凸显。我们通过优化索引构建流程，将索引创建速度提升了 10 倍。随后，我们将索引创建任务迁移至分布式框架，进一步提升了大表索引的构建效率。

随着 SaaS 用户对 TiDB 的深入应用，百万级表的场景对 DDL 框架提出了更高要求。一方面，我们需要显著提升框架的吞吐量，以在有限时间内完成大量 DDL 操作；另一方面，需要保证框架在高并发、高负载下的稳定性与扩展性。为此，我们在过去半年里，重点优化了框架的扩展性，提升了 DDL 语句的执行效率。以下为部分测试结果：

1. TiDB v8.2

通过对比 TiDB v8.1 和 v8.2 的 DDL 任务执行 QPS，我们可以清晰地看到，v8.2 版本的性能得到了大幅提升。在 v8.1 中，DDL 任务的平均 QPS 约为 7，而 v8.2 则达到了 38，峰值更是达到了 80，性能提升了约 5 倍。这表明，TiDB 团队在 v8.2 版本中对 DDL 语句的执行效率进行了深入优化，取得了显著成效。

2. TiDB v8.3

与 v8.2 版本相比，TiDB v8.3 在 DDL 任务执行的 QPS 上实现了大幅提升。v8.3 的最大 QPS 达到 200 左右，平均 QPS 也提升至 180 左右，性能表现更加稳定。这表明 TiDB 团队在 v8.3 版本中对 DDL 语句的执行效率进行了持续优化，并取得了令人瞩目的成果。

启用 `Fast Create Table` 优化后，DDL 操作的每秒查询次数（QPS）可提升一倍，显著提高系统的整体吞吐量。

3. TiDB v8.5

为了全面评估 TiDB v8.5 版本中 DDL 性能的优化效果，我们在实验室搭建了一个专用的测试集群。该集群的硬件配置如下：

节点类型	数量	规格
PD	1	8C16G
TiDB	3	16C32G
TiKV	3	8C32G

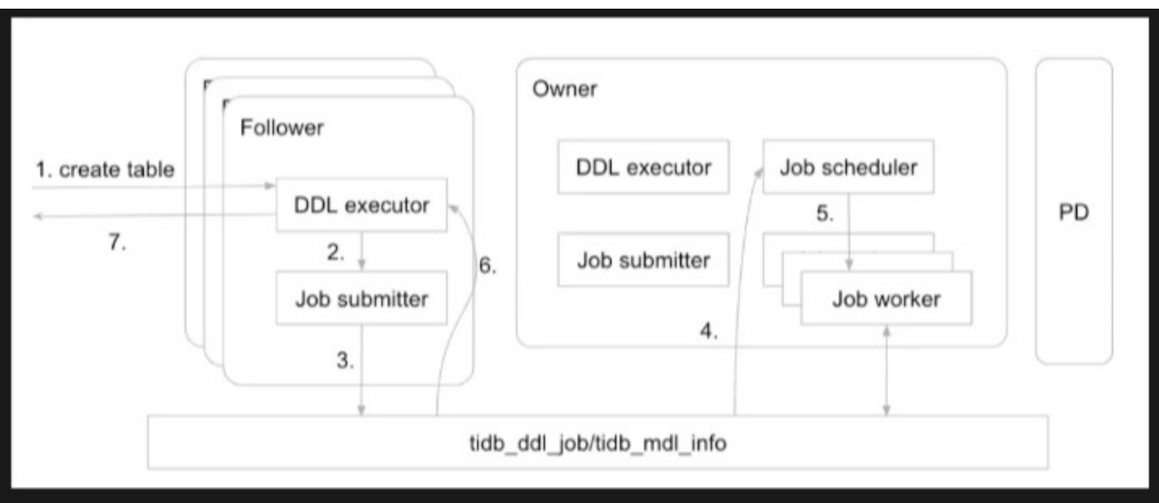
测试结果如下：

Operations	v7.5	v8.5	Description
Create 100K tables	3h49m	11m (**20X** faster) 4m (**50X** faster) if Fast Create Table enabled	Create tables inside single DB
Create 1M tables	more than 2 days	1h30m (**50X** faster)	Create 10K schemas, each have 100 tables.
Create 100K schemas	8h27m	15m (**32X** faster)	
100K add-column	6h11m	32m (**11X** faster)	All tables are created inside single DB

在百万张表场景下，**TiDB v8.5 版本的建表速度相比 v7.5 版本实现了 50 倍的性能提升。**

在深入探讨 TiDB DDL 优化之前，我们先来了解一下 TiDB DDL 语句的执行过程。TiDB 作为一款支持在线 DDL 的分布式数据库，其 DDL 操作能够在不影响业务的情况下进行。我们将重点介绍在线 DDL 变更的执行流程，包括 SQL 解析、Job 创建、后台执行等环节，为后续的优化讲解打下基础。

1 DDL 语句任务运行流程



TiDB DDL 执行流程

当用户通过客户端向 TiDB 提交一条 CREATE TABLE 语句时，TiDB 会依次执行以下步骤：

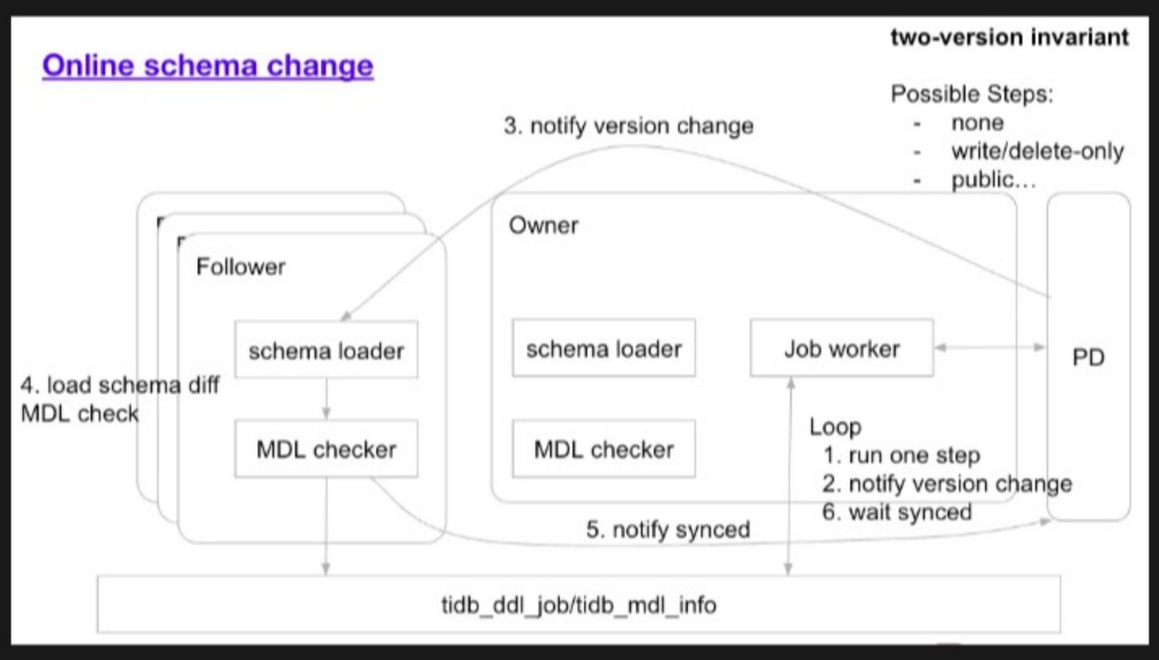
1. **SQL 解析**：TiDB 的 SQL 解析器会对该语句进行解析，生成相应的执行计划
2. **任务创建**：系统会为该 DDL 操作创建一个新的任务，并将任务信息添加到 DDL 任务队列中。
3. **任务调度**：DDL Owner 会从任务队列中取出待执行的任务，并交由调度器进行分配。
4. **作业执行**：调度器会为该任务分配一个 Job Worker，每个 Worker 负责执行一个任务。对于 reorg DDL 类型的任务，通常需要多个 Worker 并行处理。
5. **状态更新**：各个 Worker 会将执行状态反馈给系统，系统会实时更新任务的状态。
6. **结果返回**：一旦任务执行完成，系统会将执行结果先返回给接收 DDL 任务的 TiDB 节点
7. **结果返回**：再由 TiDB 节点将任务执行结果返回给客户端。

2 在线 Schema 变更简介

前面我们介绍了 TiDB DDL 任务的整体执行流程。接下来，让我们聚焦到在线 Schema 变更的细节上。当一个 Job Worker 接收到一个在线 DDL 任务时，它会按照以下步骤逐步完成任务：

- **执行单步变更**：Job Worker 会根据任务定义，执行一次在线 Schema 的变更。每一次变更都代表着 Schema 向目标状态迈进了一步，即进入下一个状态，可能的状态包括 write-only 和 delete-only 等。
- **状态更新**：完成单步变更后，Job Worker 会将当前的 Schema 状态更新到元数据中。

为了保证 schema 变更的正确性，online-schema 算法维持了以下的不变性：在任何时间，针对某个 schema 对象，整个集群中最多只有两个相连的状态存在。因此在执行完单步变更后，Job worker 需要等待系统中的所有 TiDB 节点都推进到改动后的状态，然后才能执行下一步，直到 Schema 达到最终的目标状态。



因此，在线 Schema 变更会循环执行以下几个步骤：

1. **推进变更状态**：Job Worker 会根据任务定义，将 Schema 向目标状态推进一步。这相当于为 Schema 的演进打了一个补丁。
2. **通知 PD**：变更完成后，Job Worker 会立即通知 PD（Placement Driver），告知其 Schema 版本已更新。PD 作为 TiDB 集群的“大脑”，负责协调各 TiDB 节点的状态。
3. **触发 TiDB 节点更新**：PD 会通过 ETCD 的 Watch 机制，实时监控 Schema 版本的变化。一旦检测到版本更新，PD 会立即通知所有注册到 PD 上的 TiDB 节点，要求它们更新本地 Schema 信息。这就像广播通知一样，确保所有 TiDB 节点都保持一致的 Schema 信息。
4. **MDL 检查**：TiDB 节点接收到更新通知后，会加载对应的 schema 变更，之后会触发 MDL（Metadata Locking）机制进行检查。MDL 就像一把锁，确保在 Schema 变更过程中，不会有其他操作同时修改 Schema，从而避免数据不一致。

5. **反馈执行结果**：MDL 检查通过后，TiDB 节点会将检查结果反馈给 PD，确认 Schema 更新已生效。
6. **等待所有节点同步**：Job Worker 会等待所有节点都确认 schema 更新已生效，即保证上面的不变性，之后 Job Worker 会继续处理该 Job 后续的变更任务。

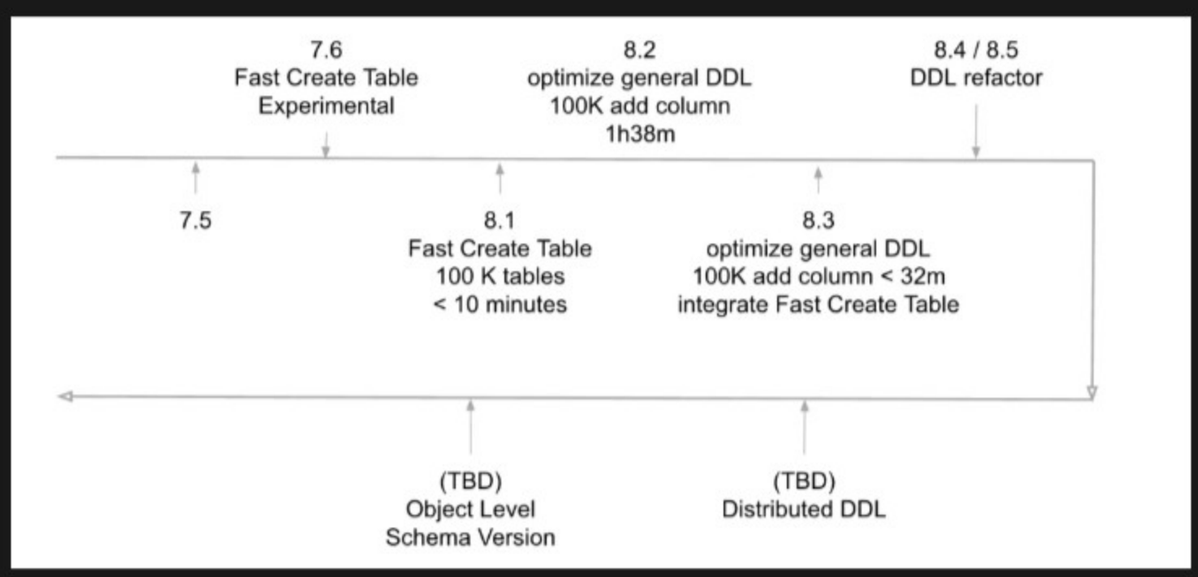
思考问题：

- **为什么需要 PD 参与？** PD 作为集群的协调者，负责通知各个 TiDB 节点更新 Schema 信息，确保集群的一致性。
- **ETCD 在这个过程中扮演了什么角色？** ETCD 作为分布式键值存储，存储了 TiDB 集群的元数据信息，包括 Schema 版本。PD 通过观察 ETCD 中的 Schema 版本变化来触发节点更新。
- **MDL 机制为什么重要？** MDL 机制保证了在 Schema 变更过程中，不会有多个操作同时修改 Schema，从而避免数据不一致。你可以想象成在修改一个文档时，需要先锁定文档，防止其他人同时修改。

通过以上步骤，TiDB 能够安全、高效地执行在线 Schema 变更，保证业务的连续性。

3 工程与最佳实践探索

1 工程思考



图：DDL 里程碑

如上图所示，我们针对 General DDL 类型的优化制定了一条清晰的路线图。考虑到 TiDB DDL 执行框架经过多年的迭代已相对稳定，为确保优化过程的安全性和高效性，我们在优化之初确立了以下几条原则：

- **客户需求驱动**：以客户需求为导向，每次优化都聚焦于解决最迫切的问题，避免大范围的重构。这种方式能有效保证优化质量，并快速交付给客户，提升客户满意度。
- **小步快跑**：将大型优化任务拆分成一系列小型的、可独立交付的子任务。这种方式不仅降低了开发难度，也便于快速验证优化效果，更符合敏捷开发的理念。
- **最小化影响**：每个子任务的设计都应尽量减少对现有系统的干扰，同时为后续的优化打下基础。

实践证明，这些原则在 DDL 优化项目中取得了良好的效果。我们成功将一个复杂的优化任务分解为多个可管理的子任务，并通过持续迭代的方式，逐步提升了 DDL 执行性能。

具体来说，我们通过以下方式来实现这些原则：

- **需求细分**：针对客户提出的 DDL 优化需求，我们进行深入分析，将其拆分为一系列具体的优化点。
- **独立子任务**：每个优化点都对应一个独立的子任务，并制定详细的开发计划和测试用例。
- **持续迭代**：每个子任务完成后，都会进行充分的测试和验证，并快速交付给客户。

在接下来的章节中，我们将详细分享我们在 DDL 优化过程中遇到的挑战、采用的解决方案以及取得的成果。

2 优化路线

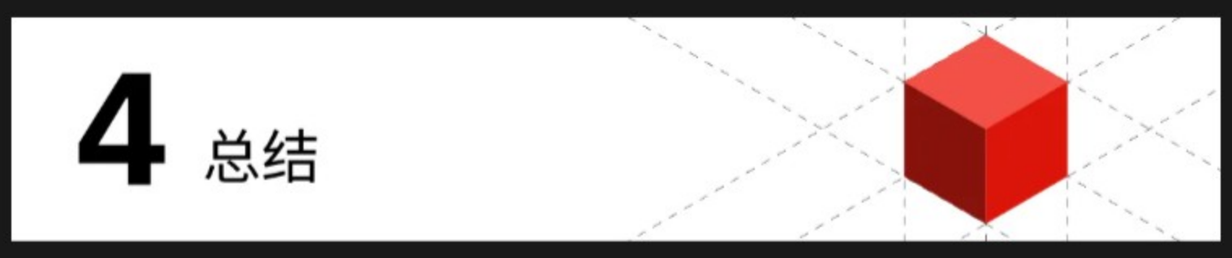
从 DDL 里程碑图中可以看出，在 TiDB v7.5 左右，我们面临着一个严峻的挑战：用户希望在一个 TiDB 集群中管理百万级表。然而，我们的测试结果显示，TiDB v7.6 在创建 50 万张表后性能急剧下降，无法支撑更大规模的建表需求。这表明 TiDB 在大规模建表场景下存在严重的性能瓶颈，阻碍了其在海量数据场景下的应用。为了解决这一问题，我们团队投入了大量精力，对 TiDB 的核心组件进行了深入优化。

基于上述工程思考原则，我们深入探索并结合实际情况，制定了如下优化路线：

1. **目标明确：优化起点**：首先，我们发现创建表在 DDL 语句中较为特殊，因此将其作为优化起点。通过深入分析，我们将创建表的需求独立出来，以便集中优化。
2. **聚焦关键：优化策略**：同时，为了避免过度定制化，我们着力于识别通用优化点。经过仔细分析，我们确定将“快速建表”作为突破口，并从 v7.6 版本开始落地。我们的目标是快速、稳定地交付给客户，提升产品竞争力。
3. **效果显著：性能提升**：经过优化，我们在 v8.1 版本实现了在 4 小时 17 分钟内创建 100 万张表，我们并没有止步于此。
4. **持续改进：深入优化**：在 v8.2 版本，我们在此基础上进一步深入优化通用 DDL 操作，对优化点进行了更细致的定位分析，并取得了显著的性能提升。v8.3 版本，我们继续优化，将创建一百万张表的时间缩短至 1.5–2 小时。
5. **夯实基础：代码重构**：为了更好地支持后续优化，我们在 v8.4 和 v8.5 版本中对 DDL 代码进行了重构，提升了代码的可维护性。
6. **展望未来：分布式革新**：未来，我们将重点打造一个分布式原生的 DDL 执行框架，实现极致的 DDL 执行性能。通过并行化 DDL 执行、分布式事务支持和智能化资源调度等技术，我们将充分发挥分布式系统的优势，提升 DDL 操作的效率。

下图是我们到 8.1 版本时，优化的效果展示：

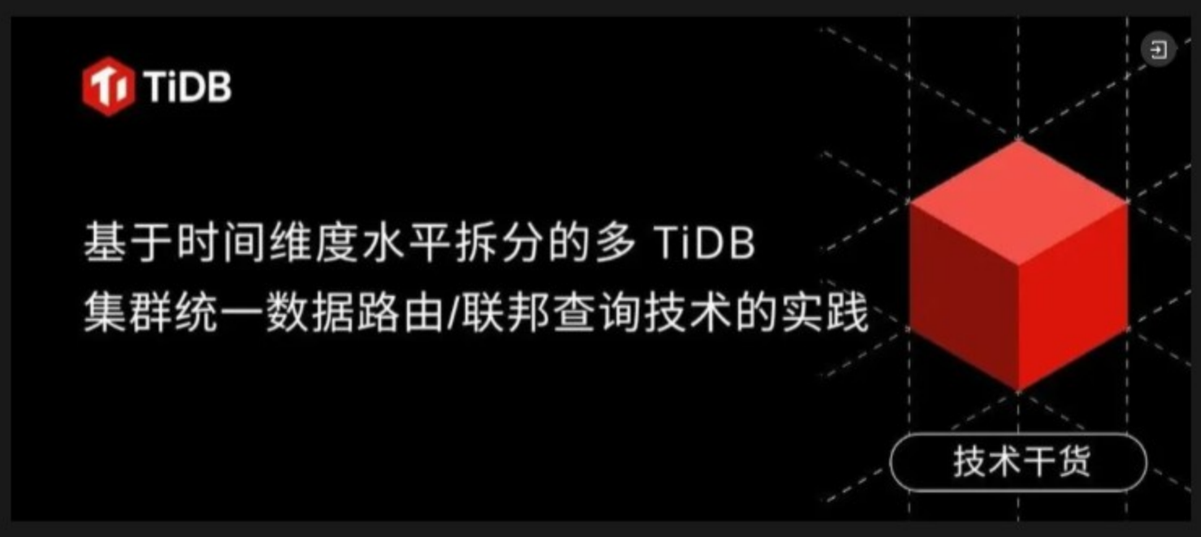
Table Num	TiDB v7.5	TiDB v7.6	TiDB 8.0	Perf enhancment ratio
100k	3h30m	26m	9m39s	21.76
300k	59h20m	estimate > 6 hours	30m38s	118



DDL 框架的重构是一项复杂而艰巨的任务，但其带来的收益是巨大的。通过重构，我们可以打造一个更加稳定、高效、可扩展的 DDL 框架，为未来的业务发展提供坚实的基础。

本文介绍了 TiDB v8 版本中 DDL 优化的效果和重构的思考，在接下来的文章中，我们将深入解析 TiDB DDL 重构的技术实现。

/ 相关推荐 /



基于时间维度水平拆分的多 TiDB 集群统一数据路由/联邦查询技术的实践

TiDB 优化器丨执行计划管理及实践


点击文末【阅读原文】，立即下载试用 TiDB！
订阅号
@PingCAP
点击关注
服务号
@TiDB Club
点击关注
视频号
@PingCAP
点击关注
PingCAP 官网
pingcap.com
点击访问
Bilibili
@TiDB_Robot
点击关注
知乎
@PingCAP
点击关注
微信搜一搜
TiDB

阅读原文