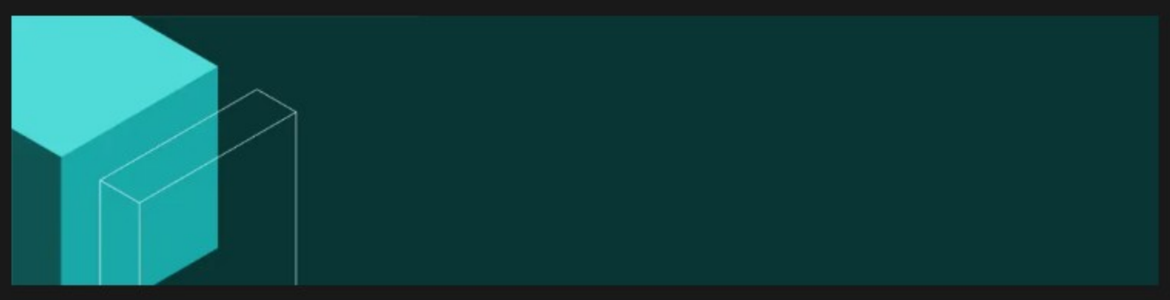
全新升级！TiCDC 新架构试用通道已开启，解锁 TiDB 数据同步新体验

原创 TiCDC 研发团队 PingCAP 2025年02月13日 19:30 北京



导读

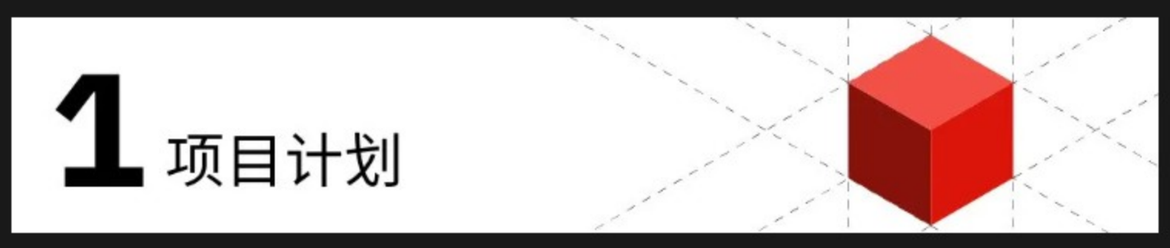
从 TiDB v6.5 开始，TiCDC（https://docs.pingcap.com/zh/tidb/stable/ticdc-overview）就代替了 Binlog 成为了首选的 TiDB 往下游系统的增量同步工具。通过拉取上游 TiKV 的数据变更日志，TiCDC 可以将数据解析为有序的行级变更数据输出到下游。

一直以来，TiCDC 作为 TiDB 生态系统中不可或缺的增量数据同步工具，在数据库灾备、数据集成、流处理与实时分析、业务连续性保障、多写多活等场景都扮演着重要的角色。为了给大家带来更好的产品体验，研发团队在提升 TiCDC 性能、稳定性、扩展性方面不断发力，为 TiCDC 设计了全新的架构。

本文简要介绍了新架构 TiCDC，它解决的问题，以及新老架构的对比，帮助读者快速对新架构有基本的认识，同时也提供了一些新版本的测试建议。

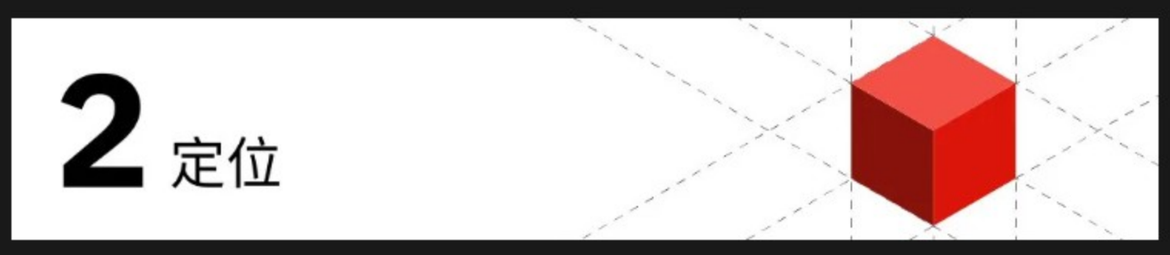
2 月 13 日 – 3 月 14 日，**TiCDC 新架构试用活动正式开启**。扫描下方二维码了解活动详情：

试用过程中你不仅可以**熟悉 TiCDC 新老架构原理与使用方法以提升技术水平**，还能够**参与新架构测试与反馈**，影响其未来优化方向，成为 TiDB 产品发展的参与者与推动者；还能够**获取丰厚奖励**，包括周边礼品及社区荣誉！



新架构 TiCDC 大约从 2024 年 6 月开始开发，目前已经发布了第一个可以给用户测试的版本（https://github.com/pingcap/ticdc/releases），预计在近期的新版本 GA 发布。

可以按照这个文档（https://github.com/pingcap/ticdc?tab=readme-ov-file#quick-start）来在老版本 TiDB 上试用新架构 TiCDC。



1. 作为 CDC 的**下一代架构**，老版本用户可以直接升级，部署和使用方式上和老版本没有任何区别，用户可以**无感升级：**

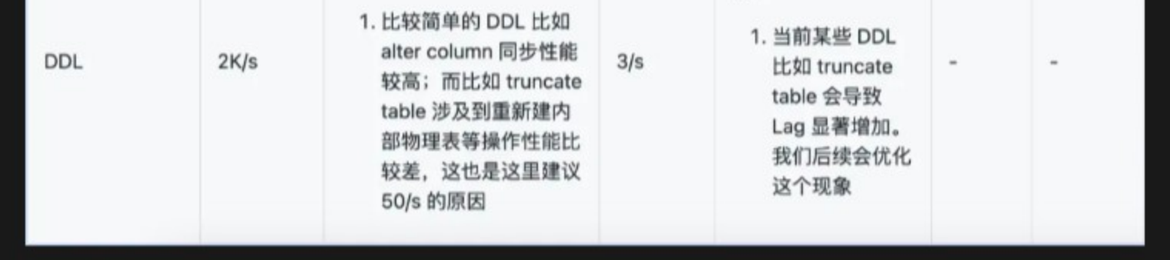
    除了 Grafana 监控做了完全重构，比之前更简单好用

2. 新架构 CDC **实验性**兼容 >= V7.5 版本的 TiDB，所以使用旧版本 TiDB 的用户可以很方便的拿新架构 CDC 来测试：

计划在充分测试并且有用户验证后，将实验性兼容变成正式兼容。由于风险未知，目前还无法估计正式支持的版本，但会作为 Experimental 发布后的高优先级事项推进

3. 为了保证项目顺利推进，新架构 CDC 正式 GA 之前，新架构 CDC **不提供任何新功能**的支持。除了：

支持单表同步任务的动态拆分和合并。目的是能支持更大的单表流量，和保持系统稳定，避免出现热点



下面提供了新架构 TiCDC 的单节点性能参考。在集群增加更多节点时，性能接近**线性增长**。

场景	测试最高性能（单节点）	生产建议最高性能	老架构性能	Lag 延迟（正常运行期间）	CPU	Memory
100 万 张窄表（sysbench）	30MiB/s	20MiB/s	最多支持 10K 表	<8s	4500%	75GiB
1 千张窄表	70MiB/s	50MiB/s	10MiB/s	<5s	2400%	3GiB
256 张宽表（120 columns）	240MiB/s	160MiB/s	96MiB/s	<5s	1000%	3GiB
DDL	2K/s	50/s ~ 1K/s 注： 1. 比较简单的 DDL 比如 alter column 同步性能较高；而比如 truncate table 涉及到重新建内部物理表等操作性能比较差，这也是这里建议 50/s 的原因	3/s	5s ~ 1m 注： 1. 当前某些 DDL 比如 truncate table 会导致 Lag 显著增加。我们后续会优化这个现象	-	-

注意：

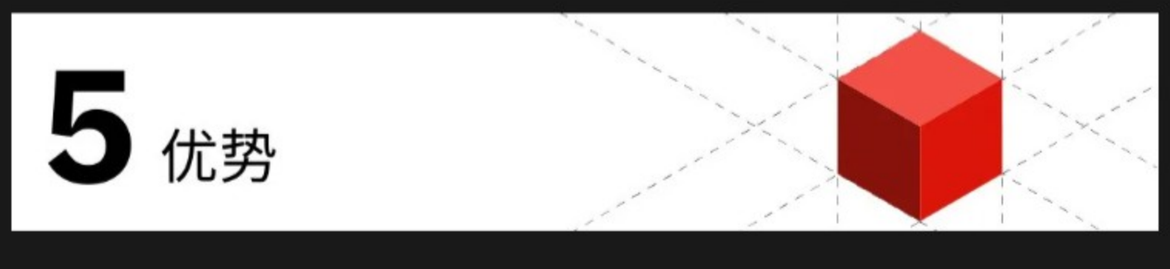
1. TiCDC 的同步流量性能和具体业务情况关系很大，包括表数量、行宽等，所以无法提供一个准确的参考值。图中提供了一些典型场景的参考；

2. 这里的性能是指 TiCDC 本身的同步性能，不包括下游的写入速度。如果下游系统存在性能瓶颈，则同步性能可能会受限，从而达不到这里的性能水准。

新架构 CDC 一开始是为了解决典型用户在规模越来越大的情况下遇到的几个问题：

- 当前 CDC 一个集群只能支持同步大约 10 万张表，400 个 changefeed，但是一些用户希望能支持同步超过 1 百万张表，以及更多的 changefeed；
- 无法稳定扩容超过 10 个 CDC 节点；
- 无法支持超大的流量，比如一个集群同步几个 GB 的流量；
- 无法稳定较高频率的 DDL。实际上当前 CDC 只能支持每个 changefeed 每秒钟大约 3 个 DDL；
- 在集群扩容、缩容期间，或者其他 changefeed 的操作期间，正常运行的 changefeed 会受到影响，导致 lag 增高。

我们深入研究发现，这些问题背后的根本限制来自于当前 CDC 的架构设计，包括一些最基本的抽象和处理逻辑。简单的优化迭代可能有一定效果但是无法彻底解决问题，所以我们启动了新架构的改造。



虽然新架构 CDC 没有功能的变化，但是它在性能、稳定性、扩展性等方面将会有显著提升。以下是目前已经发现的：

1. **显著提升了单节点处理性能：**

i. 一个 CDC 节点最高可以支持 50 万张表（0.5M）

ii. 一个 CDC 节点最多可以同步 200MiB/s 流量（宽表场景）

2. **提供了超强的扩展能力：**

i. 预计可以扩展到 100 个节点以上

a. 一个集群可以支持几 GB 的同步流量

b. 一个集群可以支持同步海量张表

ii. 支持超过 1 万个 changefeed

iii. 一个 changefeed 可以放几百万张表

3. **更高的稳定性：**

i. 在高流量下更稳定的延迟 Lag

ii. 扩容、缩容，添加、删除等操作对其他 changefeed 的影响更小

4. **更省成本，用更少的资源跑同样的流量：**

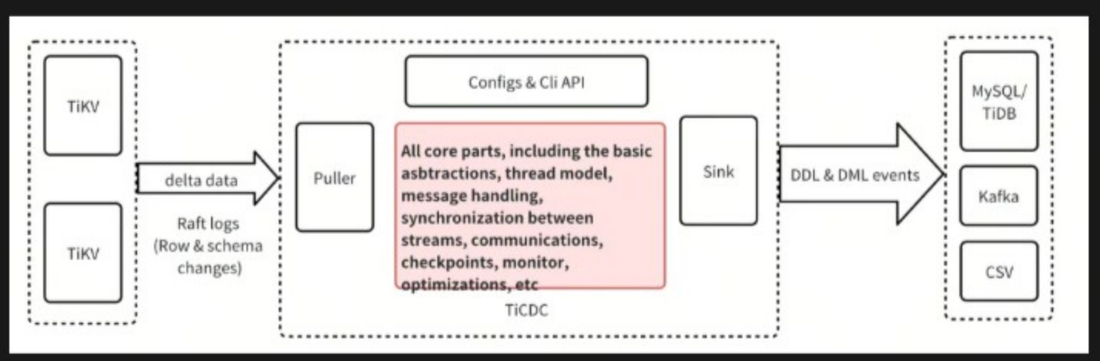
i. CPU、内存效率在典型场景有最多一个数量级的提升

此外，新架构可以显著降低代码复杂性，可以加快后续 TiCDC 的演进。另外新架构充分考虑了后续架构的发展需要，为满足更多场景的需求提供了基础，比如基于 Cloud 的架构，以及 TiCDC 独立部署等。

首先我们来看一下新架构改动了那些模块。

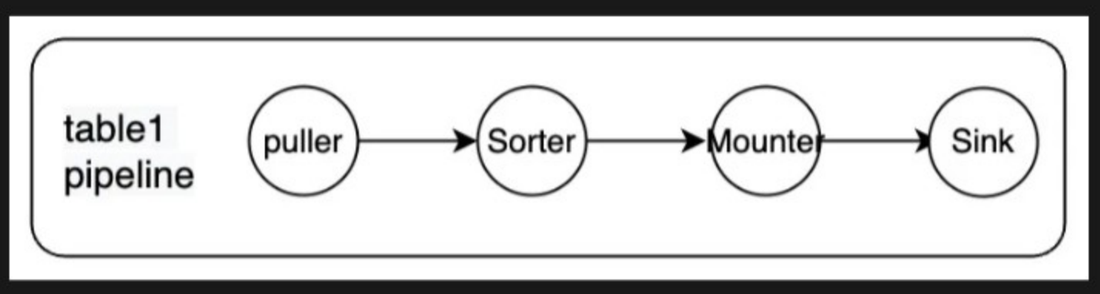
如下图所示，我们保留了三个模块基本没有改动：Puller（从 TiKV 拉取变更），Configs & Cli API（配置和用户操作 API），Sink（往下游系统写入数据的模块，比如 Kafka、MySQL 等）。这保证了新 CDC 与上下游之间有良好的兼容性，用户可以无感升级。

除此之外，CDC 的所有内核内容都进行了重写，包括基本的抽象模型，线程模型，监控等等。另外，我们还将代码仓库从 pingcap/tiflow（ https://github.com/pingcap/tiflow ）移到了 pingcap/ticdc（ https://github.com/pingcap/ticdc ），tiflow 是 TiCDC 和 DM 这两个产品共用的仓库，经过实践证明它们之间的联系非常少，放在一起反而互相影响。所以趁这个机会也让 CDC 有了独立的代码仓库，方便管理。



基本抽象和架构

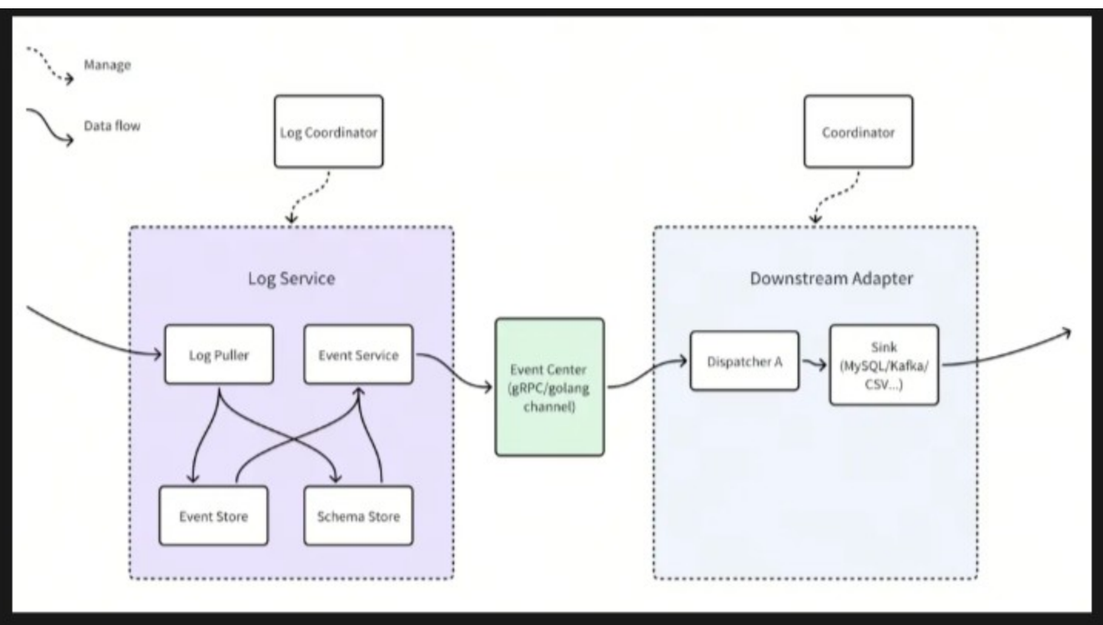
当前 CDC 的内部抽象和 changefeed 是紧密联系在一起的。一个业务层面的 changefeed 也对应了内部的 changefeed 数据结构和相应的资源，包括线程和内存资源。它包含了所有必要的模块，包括从 TiKV 拉数据的 Puller，利用磁盘排序的 Sorter，写下游的 Sink 等等。如此设计的好处是概念容易理解， changefeed 之间相对独立。带来的问题是 changefeed 的任务拆分比较困难，并且有状态和无状态的模块混杂在一起，不利于在节点间做均衡调度。



新架构中，changefeed 并没有直接对应的数据结构和资源，而是作为一个参数，控制内部服务的活动。changefeed 创建后，它会触发创建一个或者多个 dispatcher（包含了 Sink 功能），每个 dispatcher 处理一张表的一段范围的数据变更，这些 dispatcher 负责去上游 Log Service 请求数据，然后写入下游。dispatcher 是无状态的，可以被快速创建和删除。

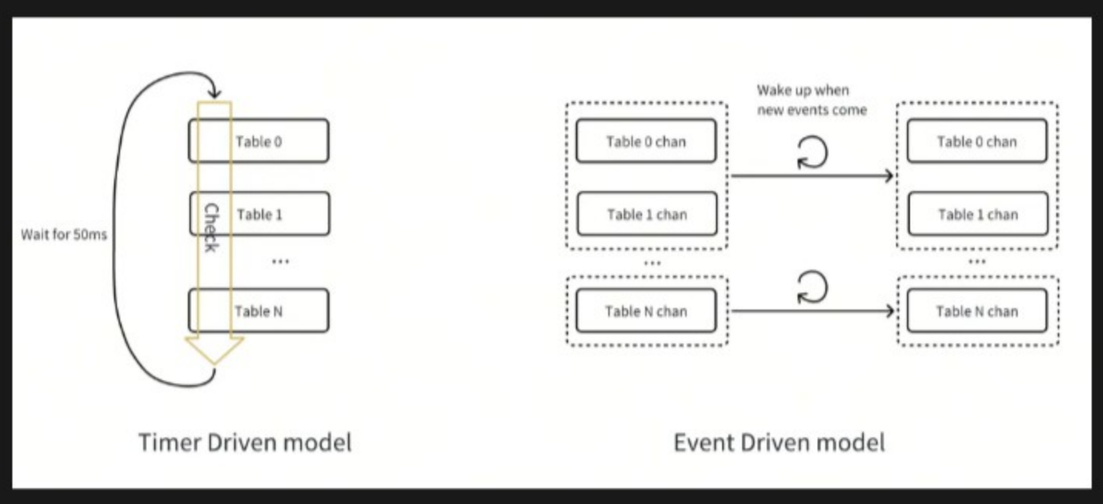
其他的模块被抽象成了独立的服务，包括专门负责从 TiKV 拉数据的 Log Puller 服务，处理数据缓存和排序的 Event Service 服务（原来的 Sorter），这些服务是带有状态的。

显著的变化是，新架构把有状态的服务和无状态的服务做了切分，并且只能通过 Event Center 模块同步消息的方式来通讯。我们用这种方式来帮助开发者仔细地设计 API 接口，避免后续代码迭代过程中的无意识的耦合行为。另外一个显著的好处是，在后续 CDC 的迭代中，可以支持将 Log Service 和 Downstream Adapter 这两个模块放在不同的进程中执行，从而让 CDC 可以根据不同的负载单独扩容必要的模块。同时也用来实现其他的功能，包括 CDC 服务独立部署。



核心处理逻辑

老架构 CDC 使用了 Timer Driven 模式来处理系统内的消息，而新架构使用 Event Driven 模式。

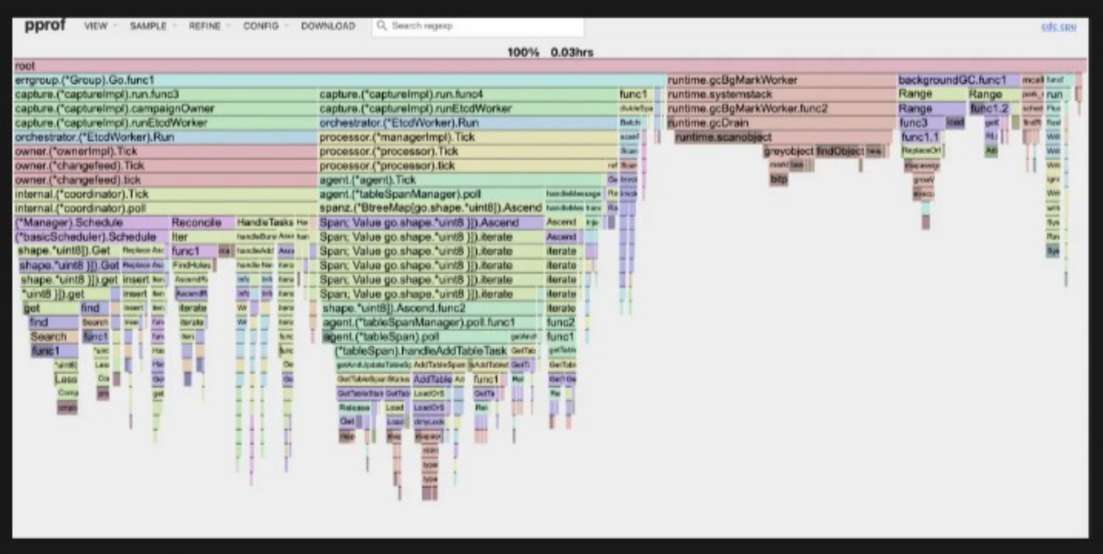


Timer Driven

Timer Driven 模式的基本逻辑是，使用一个定时器触发的大循环来不断推动执行下一批任务。比如，每一个 changefeed 内部都会有一个大循环，每隔 50ms 检查一次所有的 table，看看是否有表存在待处理的任务，有则放到处理队列中让其他线程处理。

这个处理模型的优势是容易理解。这个模式最大的缺陷是处理性能有限，无法扩展，且浪费资源。下面是一些例子

- 50ms 是平衡性能和资源浪费的结果。目前 CDC 需要 3 个步骤才能处理完一个 DDL 操作，并且由于存在中央控制节点（Owner）需要和工作节点通讯（Worker），所以实际需要 6 个循环才能处理完。1s / (3250ms) = 3.33 DDLs per second
- 这种处理模型的复杂度是 O(n)，n 一般等于表数量。当表数量很大的时候，处理性能会显著下降。并且每一次检查由于大量的表是没有新数据要处理的，等于浪费了大量的 CPU 在检查不活跃表上了
- 一个 changefeed 最多只能利用一个逻辑 CPU 来处理循环，意味着无法扩展



Event Driven

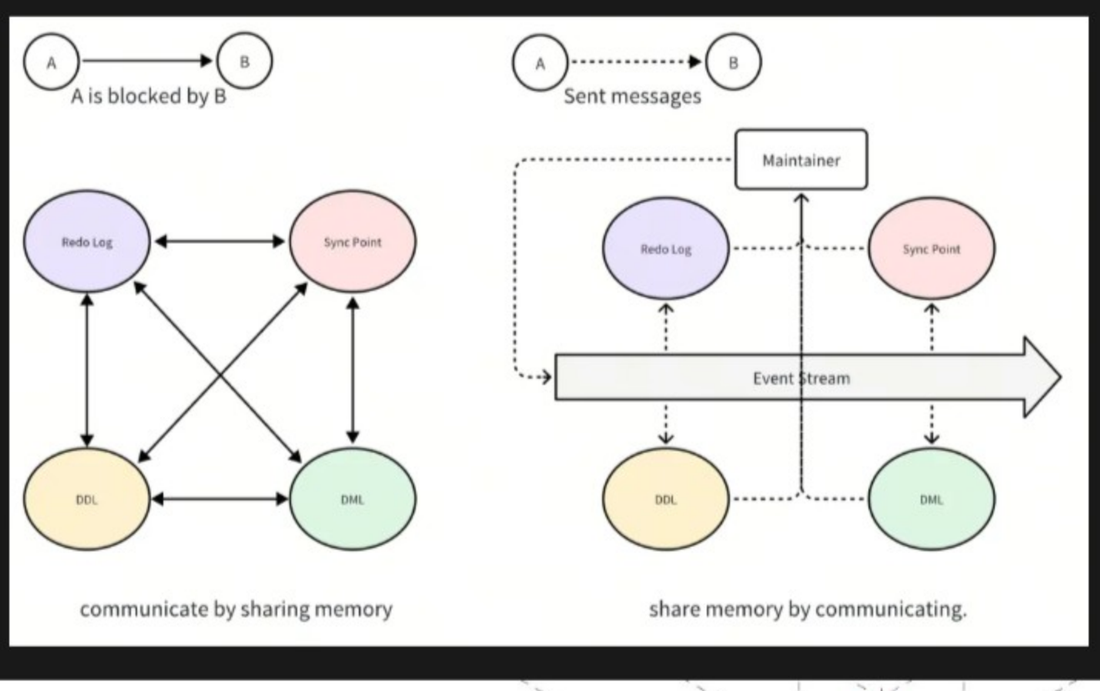
在这个模式中，所有的处理逻辑都由 Event （事件）来驱动，这些事件包括 DML、DDL 变更，changefeed 创建、修改等等。Event 被放到队列中，然后被多个线程并发消费处理。这个模式有这些好处：

- Event 会被尽快处理，不需要等待 50ms
- 提高消费线程的数量，就能提高处理能力
- 每个处理过程的复杂度都是 O(1)，效率更高，且不受 table 数量的影响。这就是新架构能处理海量表的关键

代码复杂度

虽然不同的表之间的变更大部分情况下是互不相关的，可以并行执行。但是有些情况下，表之间的变更是存在依赖的，比如跨表 DDL、Syncpoint 等等。所以在决定一个变更（比如 DML、DDL）是否可以执行，需要复杂的判断。而老架构的处理方式由于没有良好的抽象，导致代码复杂度剧增。比如下图，每个处理模块都需要去考虑其他模块的逻辑，做各种判断，它的复杂度是 N x N。这个问题是旧 CDC 代码复杂性的主要来源。

而在新架构中，我们把这些依赖抽象处理，单独放在 Maintainer 模块中处理，逻辑上每一个 Event 到达后，只需要去问一下 Maintianer 就可以知道它是否可以被马上处理，或者在可以被处理的时候 Maintainer 会主动给对应的模块发送新的 Event。这种架构的复杂度是 N x 1，大大提升了可维护性。



7 测试方向

如果你想测试新架构 TiCDC，可以考虑从这个几个角度入手：

1. 模拟你的正常业务操作，如果遇到任何 bug，请反馈到这里 ticdc/issues（https://github.com/pingcap/ticdc/issues），我们会尽快修复；

2. 观测延迟 lag 是否比之前更稳定，特别是在业务高峰期；

3. 观测资源使用率（CPU、内存）是否显著下降；

4. 测试扩容、缩容 TiCDC 集群，或者对 changefeed 进行创建、删除、pause、resume 等操作，观察 changefeed 的 lag 有什么影响，过程中流量是否平稳，是否存在不均衡的状况等；

5. 如果你之前由于表太多、DDL 太频繁或者流量太大等原因导致需要拆分 changefeed 甚至集群才能完成，可以尝试在新架构 CDC 中只用一个集群或者少数 changefeed 是否能正常同步；

6. 针对你的业务，看看 CDC 还有哪些地方可以改进的。

TiCDC 新架构试用活动现已开启，期待你的参与！

活动详情：https://asktug.com/t/topic/1039027

扫描下方二维码立即报名活动

8 2/26 晚，TiCDC 新架构介绍分享会期待你的参与

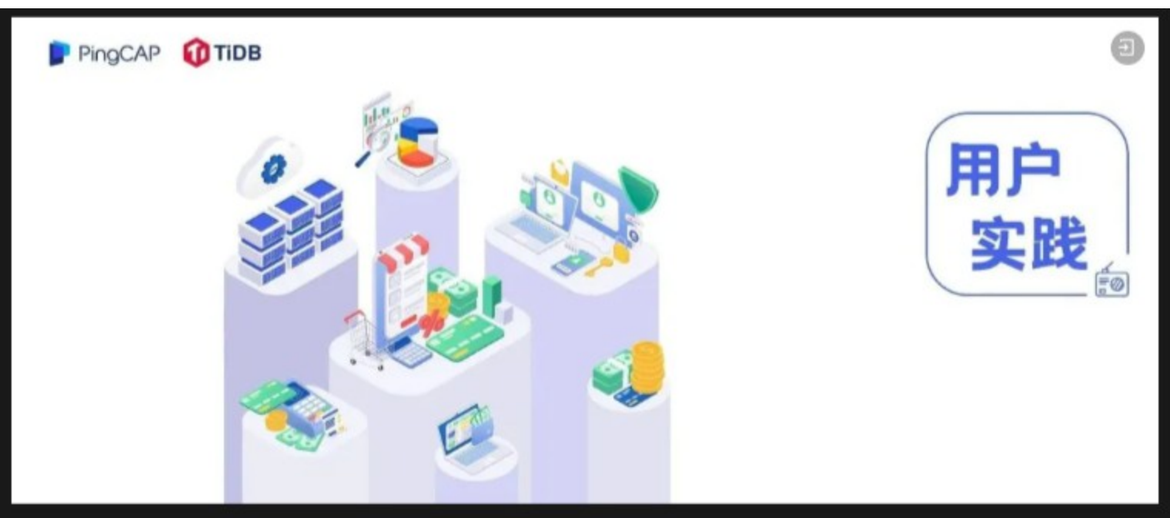
2.26 晚，TiCDC 研发负责人韦万将在线上与大家分享 TiCDC 架构升级的思考和实践：

- **活动时间：**2025 年 2 月 26 日（周三） 19:00–20:00
- **分享嘉宾：**韦万，TiCDC 研发负责人
- **分享主题：**全面了解 TiCDC 新架构带来了哪些优化
- **活动议程：**

- 19:00–19:45 主题分享
- 19:45–20:00 Q&A
- **活动报名 & 进活动群：**扫描图中二维码立即报名！

/ 相关推荐 /

基于 TiDB 资源管控 + TiCDC 实现多业务融合容灾测试



巧用 TiCDC Syncpiont 构建银行实时交易和准实时计算一体化架构

💡 点击文末【阅读原文】，立即参与 TiCDC 新架构试用活动！

阅读原文