从批处理到流式处理：加速 Uber 数据湖中的数据新鲜度

引言（Introduction）

在 Uber，数据湖是支撑全公司分析与机器学习的基础平台。过去，数据湖的摄取主要依赖批处理作业，数据新鲜度通常以小时来衡量。随着业务需求逐步向近实时洞察演进，我们重新架构了数据摄取体系，使其运行在 Apache Flink® 之上，从而实现了更新鲜的数据、更低的成本，以及在 PB 级规模下的可扩展运营能力。

在过去一年中，我们构建并验证了 IngestionNext——一个以 Flink 为核心、基于流式处理的新一代数据摄取系统。我们在 Uber 一些规模最大的数据集上验证了其性能，设计了用于运行成千上万个作业的控制平面，并解决了流式处理特有的一些挑战，例如小文件生成、分区倾斜以及检查点同步问题。本文将介绍 IngestionNext 的设计，并展示其相较于批处理摄取在数据新鲜度和效率方面取得的初步成果。

为什么选择流式处理？（Why Streaming?）

推动我们从批处理转向流式处理的两个关键因素是：数据新鲜度和成本效率。

随着业务节奏不断加快，Uber 内部的 Delivery、Rider、Mobility、Finance 以及 Marketing Analytics 等团队持续提出对更新鲜数据的需求，以支持实时实验和模型开发。批处理摄取通常会带来数小时——甚至在某些情况下长达数天——的延迟，这限制了迭代速度和决策效率。通过将摄取平台迁移至 Flink，我们将数据新鲜度从“小时级”降低到“分钟级”。这一转变直接加快了模型上线速度、实验迭代节奏，并提升了全公司的分析准确性。

从成本效率角度来看，Apache Spark™ 的批处理作业在设计上本身就较为消耗资源。它们以固定的时间间隔调度大规模分布式计算，即使实际负载存在波动。在 Uber 的规模下——成千上万个数据集、数百 PB 的数据——这意味着每天需要运行数十万个 CPU 核心。流式处理消除了频繁批处理调度的开销，使资源能够随着数据流量更平滑、更高效地伸缩。

架构概览（Architecture at a Glance）

IngestionNext 数据摄取系统由多个层次组成。

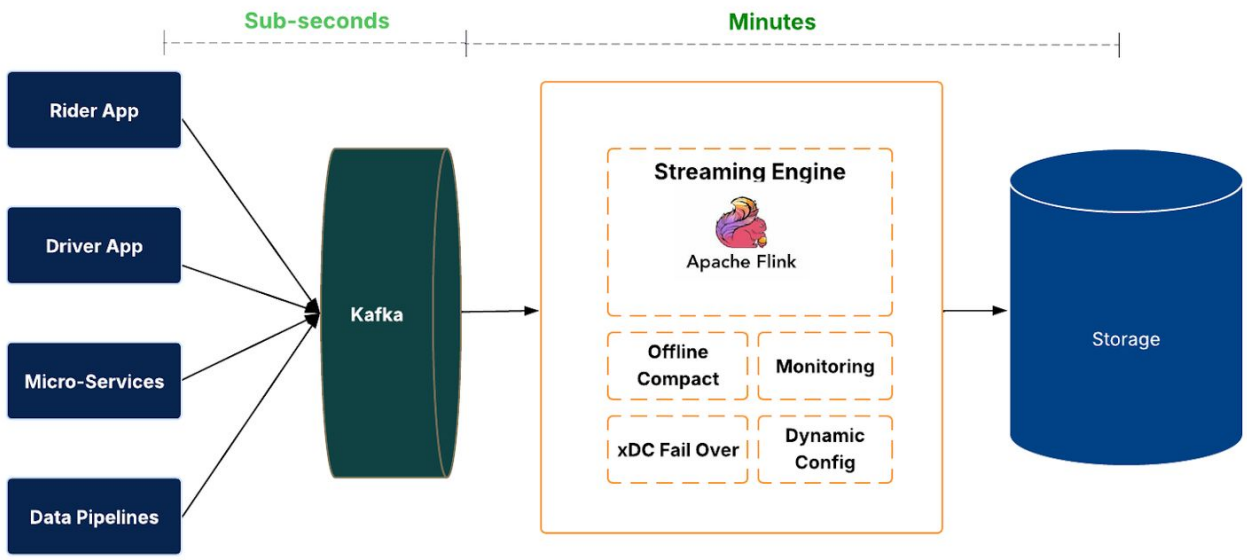

图 1：IngestionNext 架构。

在数据平面（data plane）上，事件首先进入 Apache Kafka®，随后由 Flink 作业进行消费。这些作业以 Apache Hudi™ 格式将数据写入数据湖，从而支持事务性提交、回滚以及时间回溯（time travel）。数据新鲜度和完整性以端到端方式进行衡量，从数据源一直到数据落盘。

要在大规模环境中管理数据摄取，自动化是必不可少的。我们设计了一个控制平面，用于处理作业生命周期（创建、部署、重启、停止、删除）、配置变更以及健康状态校验。这使得我们能够在成千上万个数据集上以一致且安全的方式运行摄取任务。

该系统还针对区域级故障设计了容灾与回退策略，以保障可用性。在发生故障时，摄取作业可以跨区域迁移，或临时切换为批处理模式运行，从而确保数据连续性且不发生数据丢失。

关键挑战与解决方案（Key Challenges and Solutions）

小文件问题（Small Files）

流式摄取通常会生成大量小型 Apache Parquet™ 文件，这会显著降低查询性能，并增加元数据管理和存储开销。这是数据持续到达且需要近实时写入时的一个常见问题。

传统且最常见的合并方法是逐条记录进行合并，这需要对每个 Parquet 文件进行解压、从列式格式解码为行式格式、合并之后再重新编码并压缩。虽然这种方式可行，但由于反复进行编码和解码转换，计算开销巨大且速度缓慢。

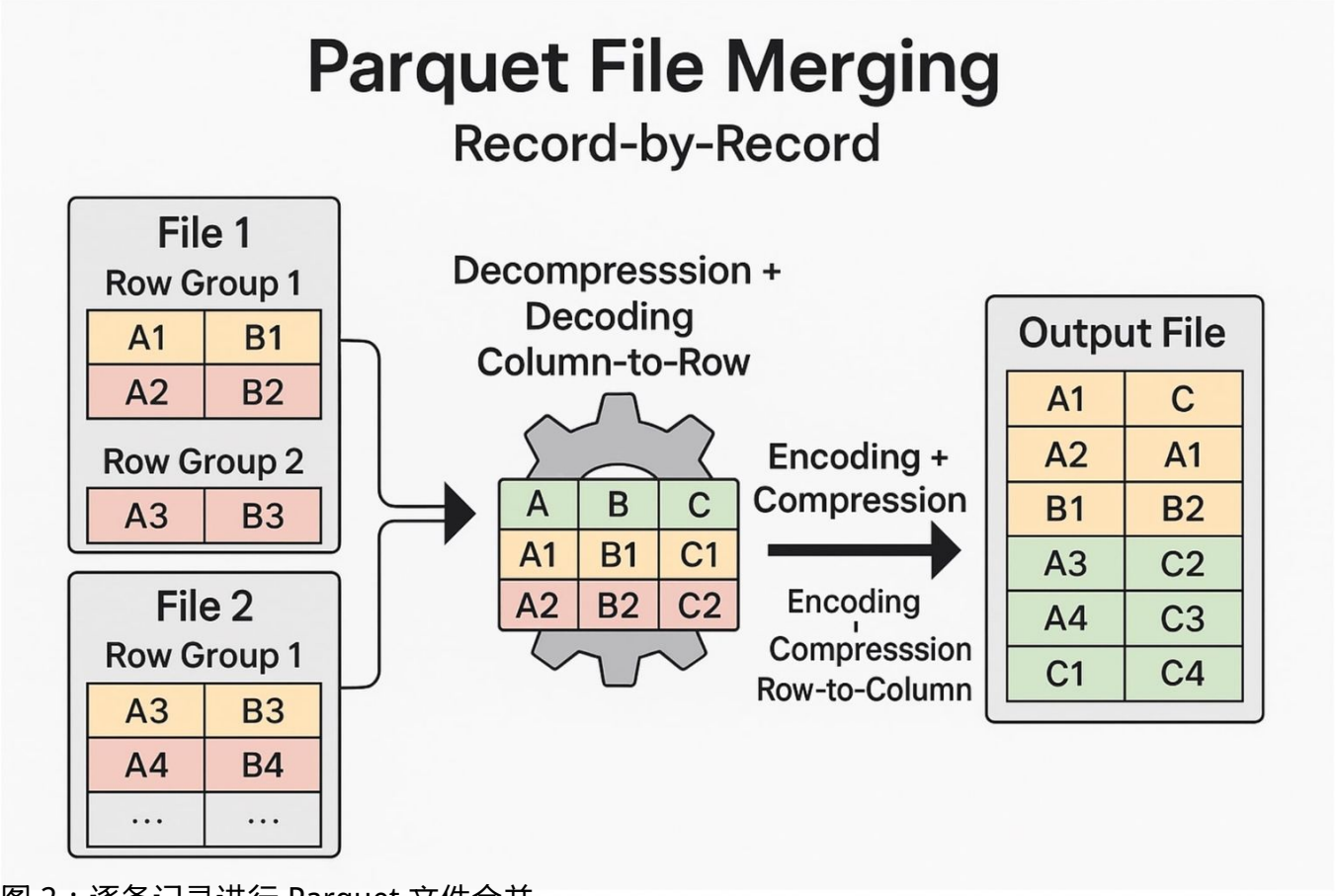

图 2：逐条记录进行 Parquet 文件合并。

为了解决这一问题，我们引入了行组级别（row-group-level）的合并，该方法直接在 Parquet 原生的列式结构上操作。这种设计避免了昂贵的重新压缩过程，使压缩（compaction）速度提升了一个数量级以上（10 倍）。

一些开源工作（例如 Apache Hudi PR #13365）探索了通过填充（padding）和掩码（masking）来对齐不同 schema，从而实现支持 schema 演进的合并。但这种方式带来了显著的实现复杂度和维护风险。

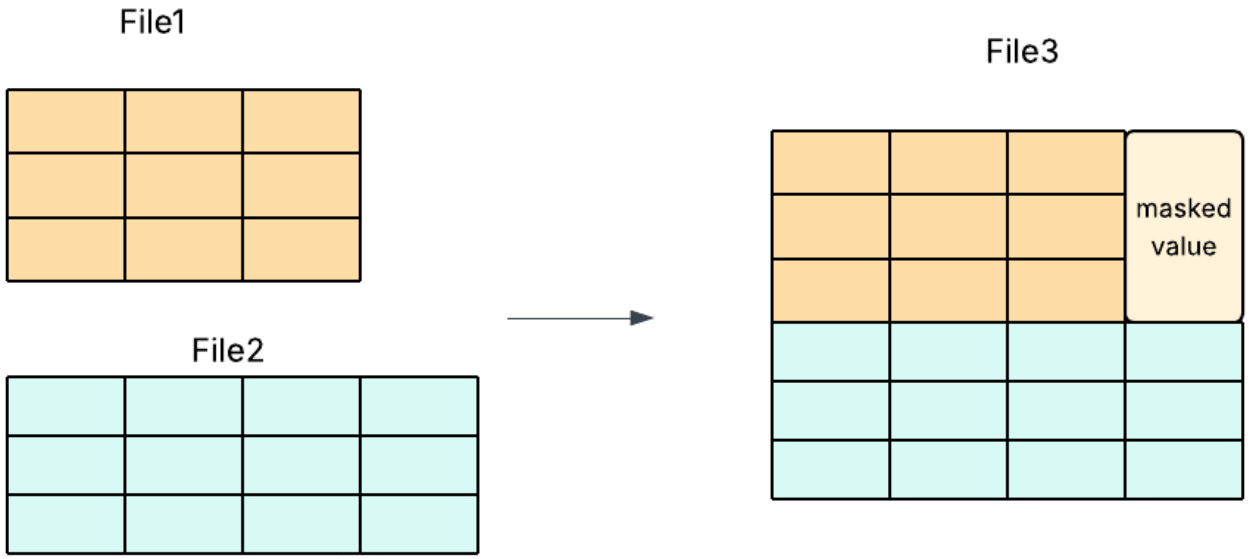

图 3：使用数据掩码的行组合并。

我们的方案通过强制保证 schema 一致性来简化流程——只合并 schema 完全相同的文件。这消除了对掩码或底层代码修改的需求，在降低开发成本的同时，实现了更快、更高效且更可靠的压缩过程。

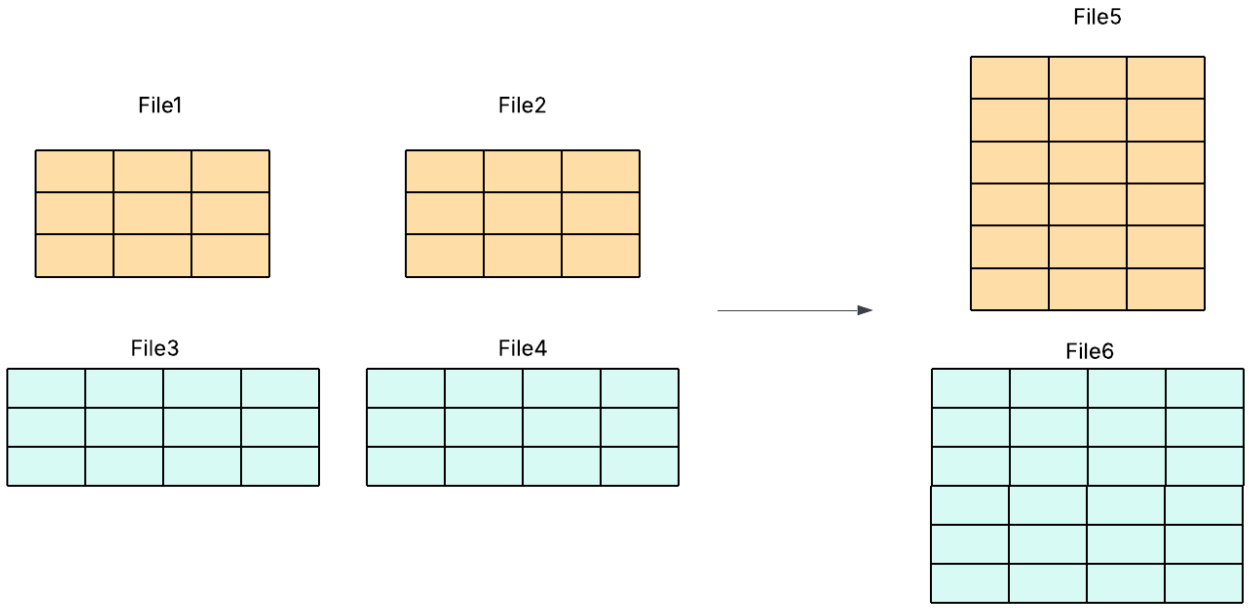

图 4：按 schema 分组的简化行组合并。

分区倾斜（Partition Skew）

我们遇到的另一个问题是：短暂的下游性能下降（例如垃圾回收暂停）可能会导致 Kafka 在 Flink 各个子任务之间的消费不均衡。数据倾斜会降低压缩效率，并拖慢查询速度。

我们通过多种方式解决了这一问题，包括：运维层面的调优（使并行度与分区数量对齐、调整拉取参数）、连接器层面的公平性机制（轮询拉取、对重负载分区进行暂停/恢复、按分区设置配额

），以及更完善的可观测性（按分区的延迟指标、具备倾斜感知能力的自动扩缩容，以及有针对性的告警机制）。

检查点与提交同步（Checkpoint and Commit Synchronization）

我们还发现，Flink 的检查点用于跟踪已消费的 offset，而 Hudi 的提交用于跟踪已写入的数据。如果在故障期间两者发生错位，就可能导致数据被跳过或重复写入。

为了解决这一问题，我们扩展了 Hudi 的提交元数据，将 Flink 的检查点 ID 嵌入其中，从而在回滚或故障切换时实现确定性的恢复。

结果（Results）

我们已将多个数据集接入基于 Flink 的摄取平台，并确认 Flink 摄取在将数据新鲜度提升至分钟级的同时，相较于批处理方式降低了约 25% 的计算资源使用。下面是一个数据新鲜度改进的示例。

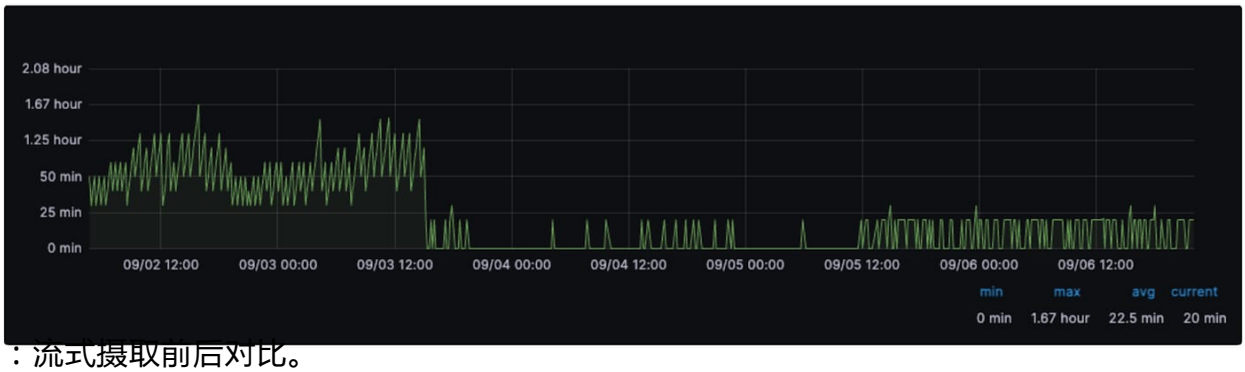

图 5：流式摄取前后对比。

下一步（Next Steps）

通过 IngestionNext，我们已大幅降低了数据摄取延迟，实现了从在线 Kafka 到离线原始数据湖的批处理向流式处理转变。然而，在下游的原始数据转换和分析阶段，数据新鲜度仍然会停滞。要真正加速数据新鲜度，我们必须将实时能力端到端地扩展——从摄取到转换，再到实时洞察与分析。

这一点在当前尤为关键。Uber 的数据湖支撑着 Delivery、Mobility、Machine Learning、Rider、Marketplace、Maps、Finance 以及 Marketing Analytics 等多个团队，因此数据新鲜度已成为这些领域的首要优先事项。大多数数据集始于摄取阶段，但如果下游转换和访问速度不足，数据在决策时仍然是陈旧的。其业务影响涵盖实验、风险检测、个性化以及运营分析等方面——数据滞后会延缓创新、降低响应速度，并限制主动、数据驱动决策的能力。

结论（Conclusion）

从批处理到流式处理的转变，标志着 Uber 数据平台演进过程中的一个重要里程碑。通过基于 Apache Flink 重新架构摄取体系，IngestionNext 为 Uber 的 PB 级数据湖带来了更高的数据新鲜

度、更强的可靠性以及可扩展的效率。系统设计强调自动化的弹性能力和运维简洁性，使工程师能够专注于构建数据驱动的产品，而非管理数据管道本身。

这一方案对工程师而言的吸引力，不仅在于其技术基础——如流式摄取、检查点同步和容错控制平面——更在于一种系统性的思维转变：将“新鲜度”视为数据质量的一等公民。随着 IngestionNext 在生产环境中得到验证，下一个前沿方向是将流式 ETL 和分析能力扩展至整个实时数据闭环，使 Uber 的所有团队都能以更高的信心、更快的速度前进。