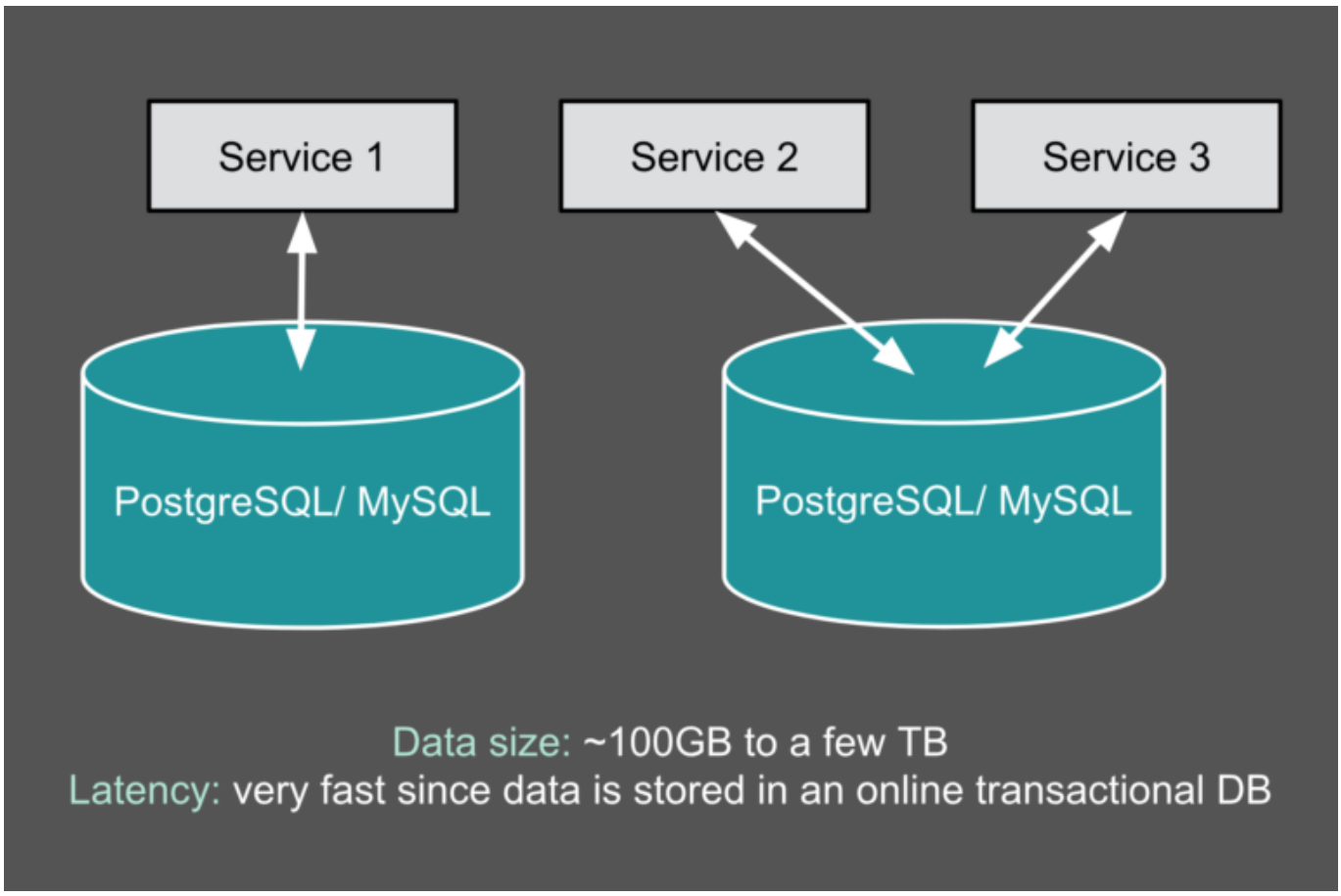
Uber 大数据平台的演进（2014~2019）

Uber 致力于在全球市场上提供更安全，更可靠的运输服务。为了实现这一目标，Uber 在很大程度上依赖于数据驱动的决策，从预测高流量事件期间骑手的需求到识别和解决我们的驾驶员-合作伙伴注册流程中的瓶颈。自2014年以来，Uber
一直致力于开发大数据解决方案，确保数据可靠性，可扩展性和易用性；现在 Uber
正专注于提高他们平台的速度和效率。

本文将介绍 Uber 的大数据平台的演进。

第一代：Uber 大数据平台的开端

在2014年之前，Uber 的数据量非常有限，可以直接存储在在线事务处理（OLTP）数据库（比如，MySQL、PostgreSQL）。 为了利用这些数据，Uber 的工程师必须单独访问每个数据库或表，如果他们需要组合来自不同数据库的数据，则用户需要自己编写一些代码。 那时，Uber
没有全局访问或所有存储数据的全局视图。 实际上，那时候 Uber 的数据分散在不同的 OLTP 数据库中，总数据大小大约为几TB，访问这些数据的延迟非常快（通常是亚分钟）。下图概述了2014年之前的数据架构：

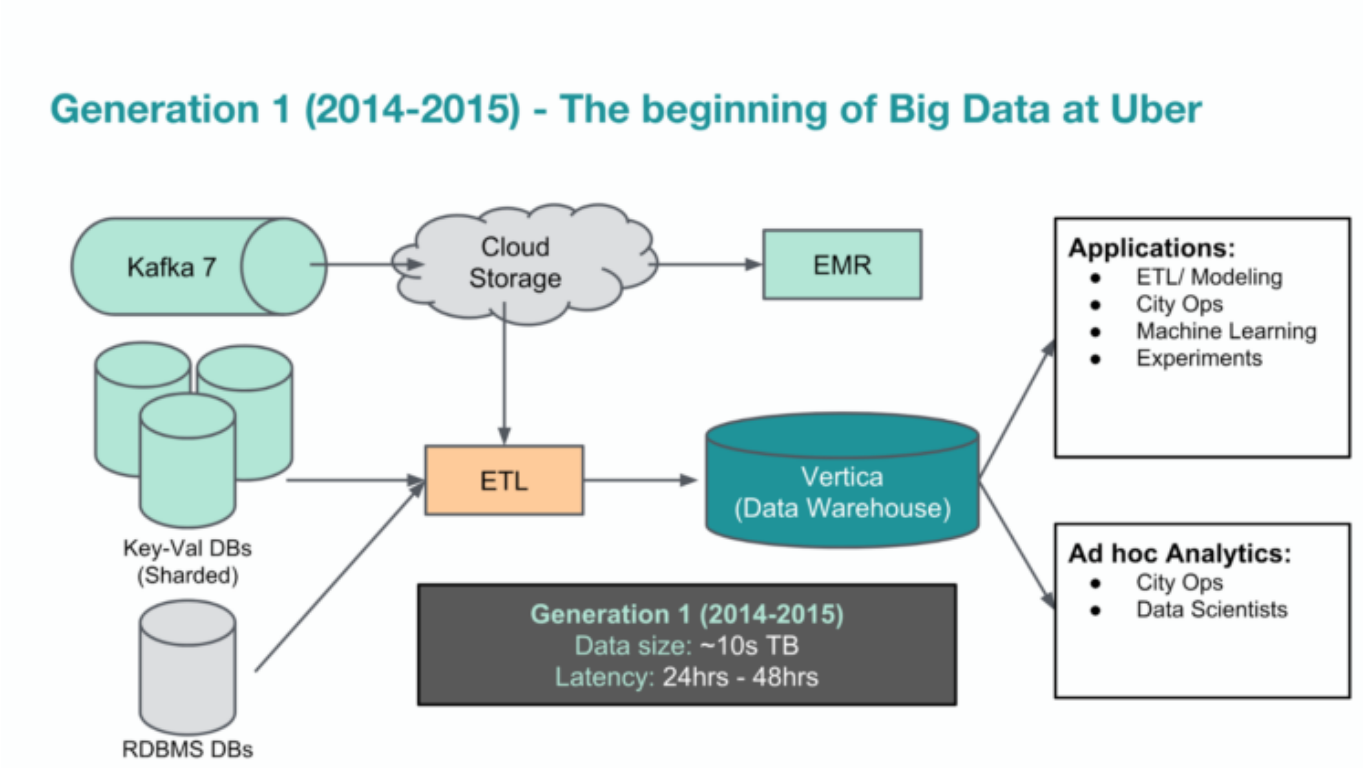


如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

随着 Uber 的业务呈指数级增长（无论是运营的城市/国家数量以及每个城市使用该服务的乘客/司机数量），传入的数据量也增加。为了能够在一个地方访问和分析所有的数据，Uber 建立第一代分析数据仓库。 为了使 Uber 尽可能地受数据驱动，需要确保分析师可以在一个地方访问分析数据。 为实现这一目标，Uber 首先将数据用户分为三大类：

- 城市运营团队（数千名用户）：这些现场工作人员负责管理和扩展 Uber 在每个市场的运输网络。随着我们的业务扩展到新的城市，有成千上万的城市运营团队需要定期访问这些数据，以解决驾驶员和骑手的问题。
- 数据科学家和分析师（数百名用户）：这些分析师和科学家分布在不同的团队，这些团队需要数据为用户提供最佳的运输和交付体验。
- 工程团队（数百名用户）：整个公司的工程师专注于构建自动数据应用程序，例如我们的欺诈检测（Fraud Detection）和司机入职（Driver Onboarding）平台。

Uber 的第一代分析数据仓库专注于将所有数据聚集在一个地方并且简化数据的访问。 对于前者，Uber 使用 Vertica 作为数据仓库软件，因为它具有快速，可扩展和面向列的设计。Vertica 是一款基于列存储的 MPP 架构的数据库。它可以支持存放多至 PB 级别的结构化数据。Vertica 是由关系数据库大师 Michael Stonebraker(2014 年图灵奖获得者)所创建，于2011年被惠普收购并成为其核心大数据平台软件。同时，Uber 还开发了多个临时 ETL（提取，转换和加载）作业，这些作业将来自不同数据源（比如 AWS S3，OLTP 数据库，服务日志等）的数据复制到 Vertica 中。 为了实现后者，Uber 将 SQL 标准化并作为其解决方案接口，然后构建了一个在线查询服务来接受用户查询，最后这些查询将提交给底层查询引擎。 下图描述了这个分析数据仓库的架构：



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

第一代数据仓库服务的发布对整个公司的工程师来说是一个巨大的成功。用户第一次拥有全局视图，可以在一个地方访问所有数据。 这导致大量新团队使用这个服务作为其技术和产品决策的基础。在几个月内，这个服务的分析数据量增长到数十TB，用户数量增加到数百。

使用 SQL
作为简单的标准接口，使城市运营者能够轻松地与数据进行交互，而无需了解底层技术。 此外，不同的工程团队开始构建针对用户需求量身定制的服务和产品，这些服务和产品由这些数据提供信息，并组建了新团队（比如机器学习团队）以更好地使用和提供这些数据。

不足

第一代数据仓库的广泛使用以及传入数据的增长使得这个服务的缺点很快暴露出来。由于数据是通过临时 ETL
作业摄取的，而且缺乏正式的通信机制，因此数据可靠性成为一个问题。另外，大多数数据都是 JSON 格式，可扩展性不强。

另外，随着公司的发展，扩展数据仓库变得越来越昂贵。为了降低成本，不得不开始删除旧的过时数据，以释放空间。 除此之外，Uber 的大部分大数据平台都不能横向扩展，因为之前的主要目标是解决集中数据访问的关键业务需求，并且没有足够的时间来确保所有部件都是水平可扩展的。这个数据仓库实际上被用作成数据湖，里面堆积了所有原始数据，并在其之上进行了数据建模。

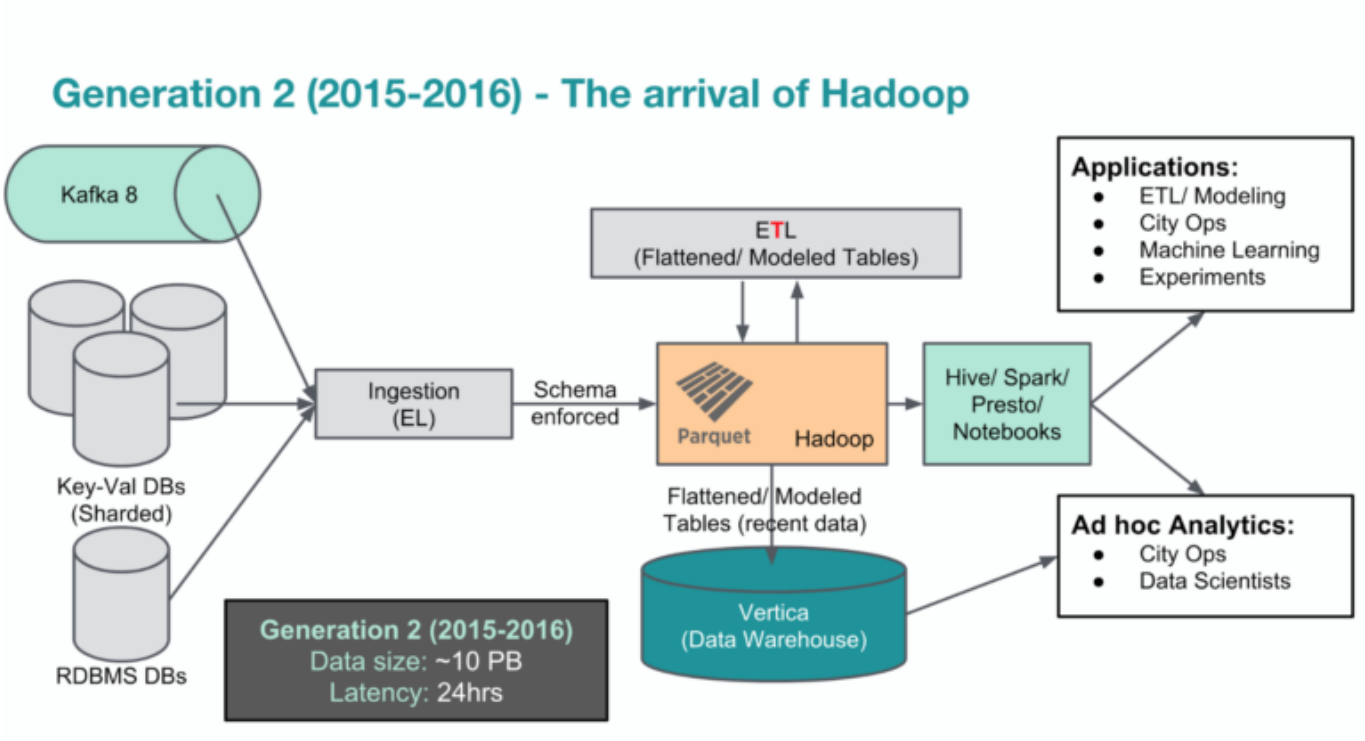
最后，由于生成数据的服务与下游数据消费者之间缺乏通信，将数据写入到数据仓库的 ETL 作业也非常脆弱。如果不同的用户在摄取期间执行了不同的转换，则可能导致相同的数据被摄取多次。这对我们的上游数据源造成了额外的压力，并影响了他们的服务质量。此外，这也导致数据仓库存储了几个相同数据的副本，进一步增加了存储成本。数据的摄取工作缺乏标准化，因此很难摄取任何新的数据集和类型。

第二代：Hadoop 的引入

为了解决这些限制，Uber 开始围绕 Hadoop
生态系统重新构建了新的大数据平台。更具体地说，Uber 引入了一个 Hadoop
数据湖，其中所有原始数据仅从不同的在线数据存储中提取一次，并且在摄取期间不进行转换。这种设计转变明显降低了在线数据存储的压力，使 Uber
能够从临时摄取作业过渡到可扩展的摄取平台。 为了让用户能够访问 Hadoop 中的数据，使用 Presto 来实现交互式查询，使用 Apache Spark 对原始数据进行编程访问，使用 Apache Hive 进行非常大的离线查询。
这些不同的查询引擎允许用户针对其需求进行选择，这使得第二代大数据平台更加灵活。

为了保持平台的可扩展性，我们确保所有数据建模和转换仅在 Hadoop
中进行，从而在出现问题时实现快速恢复。只有最关键的建模表才可以迁移到这个数据仓库。这大大降低了运行庞大数据仓库的运营成本。

我们还利用了 Apache Parquet
的标准列式文件格式，这提高了数据压缩率，从而节省了存储空间。此外，Parquet 与 Apache Spark 的无缝集成使该解决方案成为访问 Hadoop
数据的流行选择。下图总结了我们的第二代大数据平台的架构：



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

除了整合 Hadoop 之外，我们还使该生态系统中的所有数据服务都可以横向扩展，从而提高了我们的大数据平台的效率和稳定性。 特别是，具有这种通用的水平可扩展性可以满足业务需求，使我们能够专注于构建下一代数据平台。

与第一代平台不同，第二次大数据平台允许对所有数据进行模式化，从 JSON 转换为 Parquet 以将模式和数据存储在一起。 为实现这一目标，我们构建了一个集中模式服务来收集，存储和相关客户端库，以便将不同服务数据模式集成到这个中央模式服务中。

随着 Uber 的业务继续以快速的速度扩展，我们很快就拥有了数百PB的数据。
每天都有数十TB的新数据被添加到数据湖中，大数据平台增长到超过10,000个 vcores，每天都有超过100,000个批处理作业。 这导致这个Hadoop数据湖成为所有分析 Uber 数据的集中真实来源。

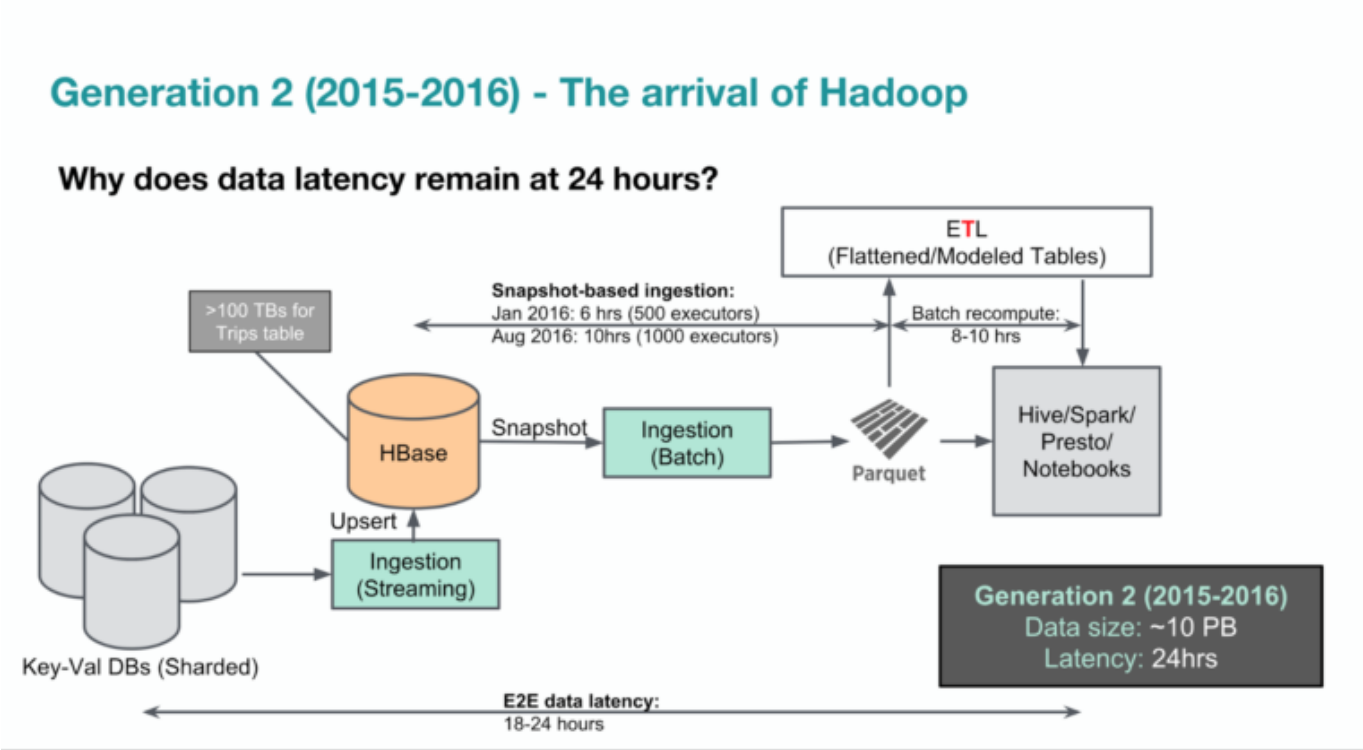
不足

随着公司不断扩展，并且这个数据平台存储了数百PB的数据，我们开始面临着一系列新的挑战。

首先，HDFS 中存储了大量的小文件开始对 NameNode 产生额外的压力。最重要的是，数据延迟仍然远远超出了我们的业务需求。用户只能每隔24小时访问一次新数据，这对于需要做实时决策

的需求来说太慢了。将 ETL 和建模迁移到 Hadoop
使得这个过程更具可扩展性，这些步骤仍然是瓶颈，因为这些 ETL 作业必须在每次运行重新创建整个建模表。除此之外，新数据的摄取和相关派生表的建模都需要创建整个数据集的新快照，并将新表替换旧表以向用户提供对新数据的访问。 摄取作业必须返回源数据存储区，创建新快照，并在每次运行期间将整个数据集转换为列式存储的 Parquet 文件。 随着我们的数据存储量的增长，这些工作可能需要超过20个小时才能运行超过1,000个Spark执行程序。

每项工作的很大一部分涉及将历史数据和新数据进行转换并生成新快照。虽然每张表每天只添加100MB左右的新数据，但每次运行的摄取作业都必须转换该表的整个数据集。对于在每次运行中重新创建新派生表的ETL和建模作业也是如此。从本质上讲，我们的数据包含许多更新操作。由于 HDFS 和 Parquet
不支持数据更新，因此所有的提取作业需要从更新的源数据创建新快照，将新快照写入到 Hadoop 中，并将其转换为 Parquet 格式的文件，然后替换表的数据以查看新数据。下图总结了这些基于快照的数据摄取如何在大数据平台进行的：



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

第三代：从长远角度重新构建大数据平台

到2017年初，我们的大数据平台被整个公司的工程和运营团队使用，使他们能够在一个地方访问新的和历史的数据。用户可以通过单一的 UI 接口轻松访问
Hive，Presto，Spark，Vertica，Notebook 中的数据。我们的计算集群中有超过 100PB
的数据，集群共有 100,000 vcores。每天有超过 100,000 个 Presto 查询、 10,000个Spark
作业、20,000个 Hive 查询。我们的 Hadoop
分析平台遇到了可扩展性问题，许多服务都受到高数据延迟的影响。

幸运的是，由于我们的底层基础架构可以横向扩展以满足当前的业务需求，因此我们有足够的时间研究数据内容，数据访问模式和用户特定需求，以便在构建下一代大数据处理平台之前确定最紧迫的问题。我们总结了四个主要的痛点：

- HDFS 可扩展性限制：许多依赖 HDFS
  扩展其大数据基础架构的公司都面临着这个问题。根据设计，HDFS 受 NameNode 内存容量的限制，因此存储大量小文件会显着影响性能。当数据大小超过10PB这个问题开始出现，如果数据量达到50-100 PB就会成为问题。
  幸运的是，有一些相对简单的解决方案可以将 HDFS 从几十PB扩展到几百PB，例如利用 ViewFS 和 HDFS NameNode Federation；或通过控制小文件的数量并将不同的数据移到单独的集群，这样我们能够减轻 HDFS 的瓶颈。
- Hadoop中的数据需要快速更新：Uber
  的业务是实时计算的，因此，我们的服务需要尽可能的访问到新数据。 因此，对于许多用例而言，T+1的数据更新延迟太长了，这些场景对数据的实时更新需求很高。 我们的第二代大数据平台基于快照的摄取方法效率低下，使得我们难以以较低的延迟摄取数据。
  为了加速数据交付，我们不得不重新构建我们的数据流管道，并仅仅增量摄取更新的和新插入的数据。
- 支持在 Hadoop 和 Parquet 中进行更新和删除操作：Uber 的数据包含大量更新，更新范围包括过去几天到几周甚至几个月。通过基于快照的数据提取，我们每24小时提取一次源数据的新副本。换句话说，我们一次摄取所有更新，每天一次。但是，由于需要更新的数据和增量摄取，我们的解决方案必须能够支持现有数据的更新和删除操作。但是，由于我们的大数据存储在 HDFS 和 Parquet
  中，因此无法直接支持对现有数据的更新操作。另一方面，我们的表通常包含 1,000 个列，并且有五个或更多嵌套级别，而用户的查询通常只触及其中一些列，从而阻止我们使用更加高效的行式存储。
  为了让我们的大数据平台能够实现长期增长，我们必须找到一种方法来解决 HDFS
  文件系统中的这种限制，以便我们也可以支持更新/删除操作。
- 支持更快的 ETL 和建模：与原始数据摄取类似，ETL
  和建模作业也是基于快照方式的，这要求我们的平台在每次运行时重建派生表。
  为减少建模表的数据延迟，ETL 作业也需要以增量方式获取数据。 这要求 ETL 作业逐步从原始源表中提取已更改的数据并更新先前派生的输出表，而不是每隔几小时重建整个输出表。

Hudi 介绍

考虑到上述要求，我们构建了 Hudi（Hadoop Upserts anD Incremental），这是一个开源的 Spark 库，在 HDFS 和 Parquet 之上提供抽象层，以支持所需的更新和删除操作。Hudi
可以在任何 Spark 作业中使用，可以横向扩展，并且运行时只依赖于
HDFS。因此，任何需要支持历史数据更新/删除操作的大数据平台都可以利用 Hudi。

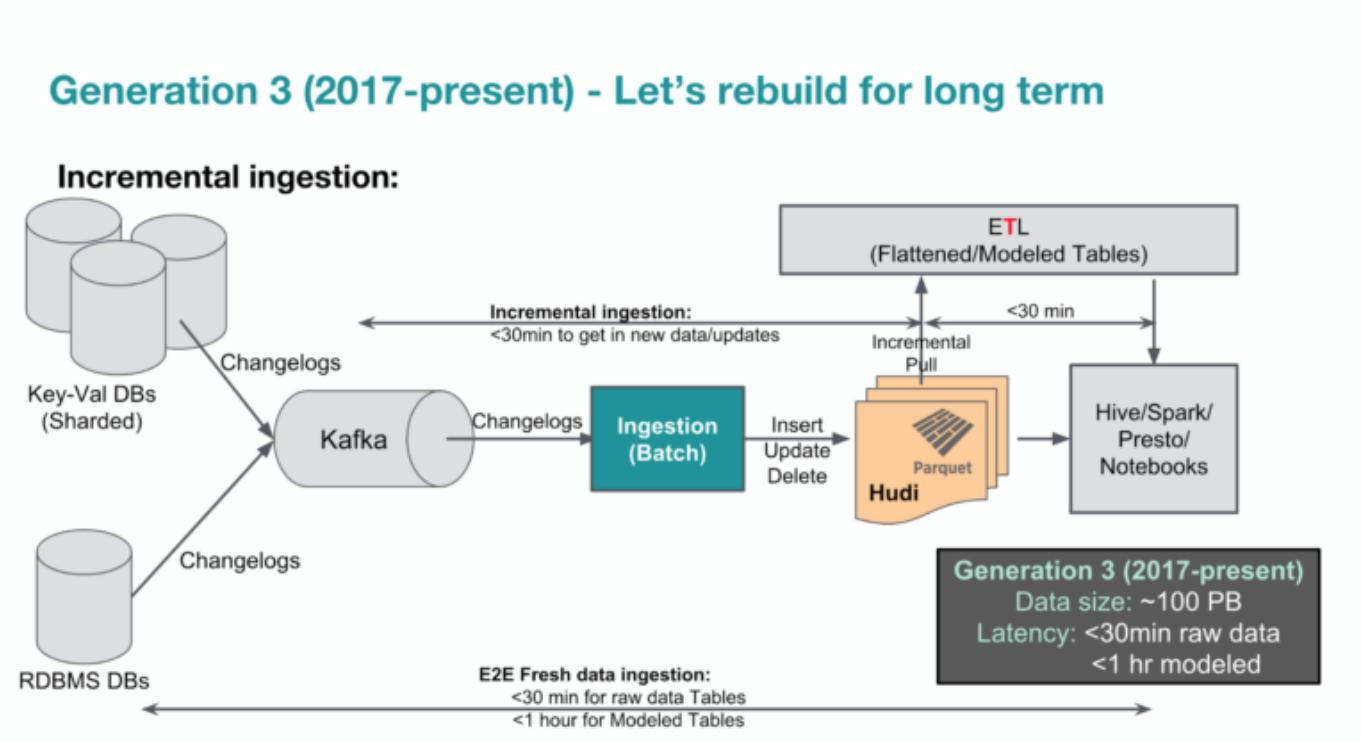
Hudi 使我们能够在 Hadoop 中更新，插入和删除现有的 Parquet 数据。 此外，Hudi
允许用户获取仅更改的数据，显着提高查询效率并允许派生建模表的增量更新。

我们的 Hadoop 生态系统中的原始数据是根据时间分区的，任何旧分区都可能在以后被更新。
因此，对于依赖于这些原始数据表的用户或 ETL
作业，了解哪个分区被更新的唯一方法就是扫描整个源表并根据一些条件过滤不要的数据。

这导致查询的计算成本非常高，需要对源表进行全量扫描。

通过引入 Hudi，用户可以传递最后一次 checkpoint
时间戳并检索所有已更新的记录，无论这些更新是添加到新分区还是旧分区，无需扫描整个表。

通过 Hudi 库，我们能够从基于快照的原始数据提取转移到增量提取模型，使我们能够将数据延迟从24小时减少到不到一小时。下图描述了引入 Hudi 后我们的大数据平台架构：



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

备注：其实 Uber 的 Hudi 和前段时间 databricks 开源的 Delta
功能很类似，这个系统目前也开源了，具体参见：https://hudi.apache.org

第四代：展望

自2017年推出第三代大数据平台以来，整个公司的用户可以快速可靠地访问 Hadoop
中的数据，但是这个平台还是有很大的跟进空间。 下面总结了我们增强 Uber
的大数据平台的努力，以提高数据质量，数据延迟，效率，可扩展性和可靠性。

数据质量

为了加强数据质量，我们确定了两个改进方向。首先，当某些上游数据存储在存储之前没有强制执行或检查数据模式时，我们希望避免非符合模式的数据。因为这个会导致不规范的数据进入我们的 Hadoop 生态系统，从而影响所有依赖此数据的下游用户。为了防止脏数据流入，我们正在对所有上游数据存储的数据模式进行强制检查，并在数据存在任何问题时拒绝这些数据的写入。

我们发现的第二个改进点是数据内容的质量。
虽然使用模式检查能够确保数据包含正确的数据类型，但它们不检查实际数据值。
为了提高数据质量，我们正在扩展架构服务以支持语义检查。
这些语义检查允许我们在基本结构类型检查之外添加对实际数据内容的额外约束。

数据延迟

我们的目标是将 Hadoop 中的原始数据延迟减少到五分钟，将建模表的数据延迟减少到十分钟。这将允许更多用例从流处理转向使用 Hudi 增量数据拉取的更有效的小批量处理。

我们还在扩展我们的 Hudi
项目以支持视图模式，其中包括现有的读取优化视图，以及显示延迟仅几分钟的数据实时视图。这个实时视图依赖于 Merge-On-Read 或 Hudi 2.0。

数据效率

为了提高数据效率，我们打算不再依赖专用硬件来实现任何服务和服务容器化。此外，我们统一了 Hadoop 生态系统内部和跨 Hadoop 生态系统的所有资源调度程序，以弥合整个公司的 Hadoop 和非数据服务之间的差距。 这允许所有作业和服务以统一的方式进行调度。随着 Uber 的发展，数据位置将成为 Hadoop
应用程序的一大关注点，成功的统一资源管理器可以将所有现有的调度程序集中在一起。

扩展性和可靠性

为了确保无论数据从哪里来的都能统一进行数据摄取，我们与 Uber
数据存储团队合作启动了一个项目，以统一所有上游数据源的更改日志的内容，格式和元数据。

本文翻译自：Uber’s Big Data Platform: 100+ Petabytes with Minute Latency

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）