Presto 在 Pinterest 的实践

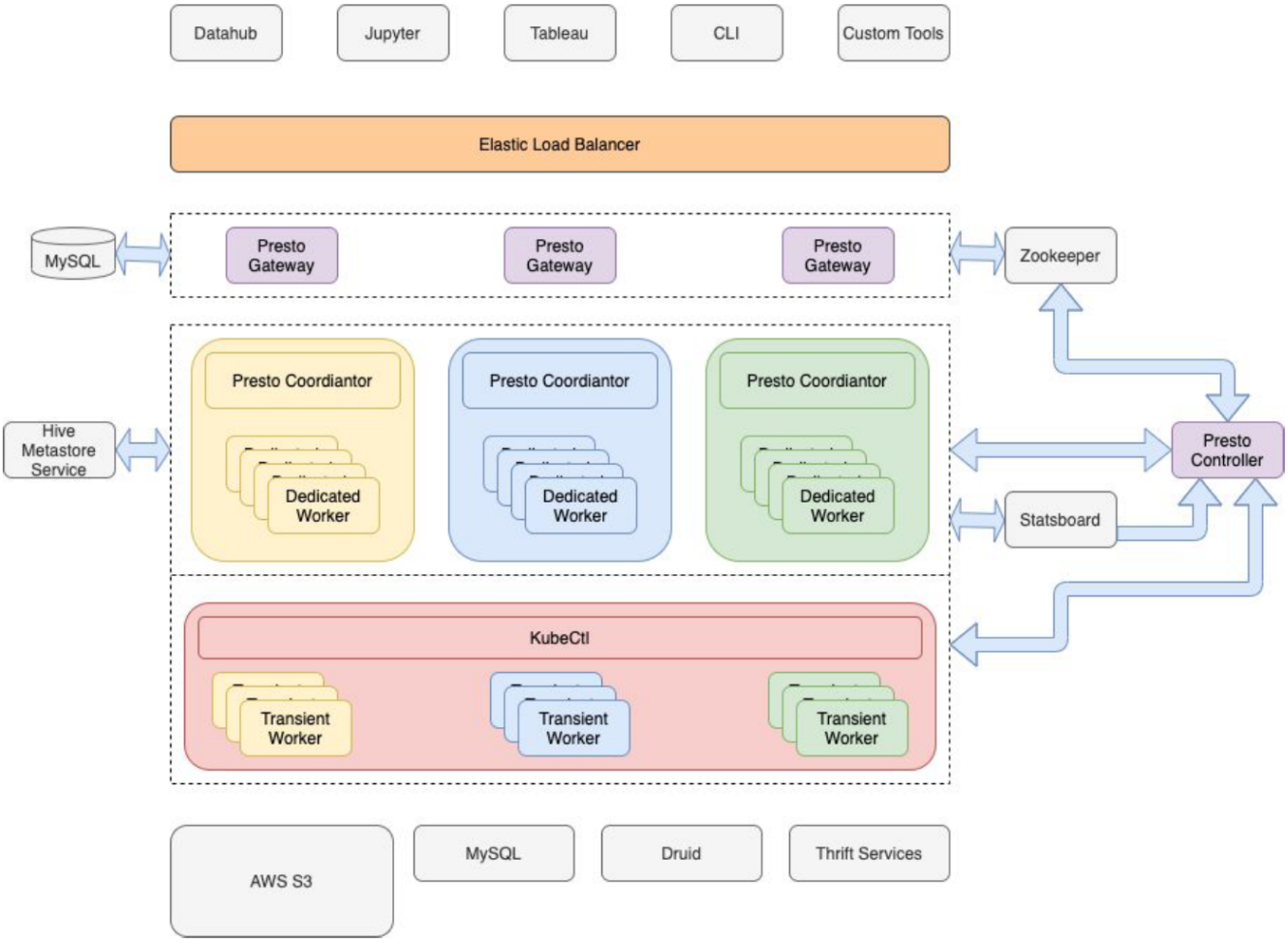
作为一家数据驱动型公司，Pinterest 的许多关键商业决策都是基于数据分析做出的。分析平台是由大数据平台团队提供的，它使公司内部的其他人能够处理 PB
级的数据，以得到他们需要的结果。

数据分析是 Pinterest 的一个关键功能，不仅可以回答商业问题，还可以解决工程问题，对功能进行优先排序，识别用户面临的最常见问题，并看到使用趋势。因此，在
Pinterest，工程师和非工程师都同样需要这些分析能力。SQL 及其变体已被证明可以为员工提供一个有效表达其计算需求或分析的平台。它还提供了用户代码/查询和底层计算基础设施之间的巨大抽象，使基础设施能够在不影响用户的情况下发展。

为了向员工提供交互式查询的关键需求，我们多年来一直与 Presto 社区合作。在 Pinterest 的规模下使用 Presto 需要解决相当多的挑战。在这篇文章中，我们将介绍 Presto 在 Pinterest 的使用情况。

概述

下图展示了 Pinterest 的 Presto 部署。我们的基础架构建立在 Amazon Web Services (AWS) EC2 之上，我们主要利用 AWS S3
来存储数据。这可以将计算层和存储层进行分离，并允许多个计算集群共享 S3
上的数据。我们有多个处理不同用例的 Presto
集群。这些集群可以是长期的，也可以是短期的。两个主要的集群分别是 ad-hoc 和 scheduled 集群：前者主要用于解决 ad-hoc 查询；后者服务于调度查询。将 ad-hoc
查询与调度查询进行分离使我们能够为调度查询提供更好的 SLA
保障，并且还为调度集群的资源需求带来了更多的可预测性。



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：过往记忆大数据

Presto 在 Pinterest 的部署情况

目前 Pinterest 是基于 PrestoSQL 320 修改的版本，数据源包括 Hive（主要），MySQL、Druid 以及 Thrift。我们在 S3 上有 400多PB 的数据和数万张 Hive 表。我们的 Presto 集群有超过 500 个 EC2 实例组成。Presto 集群总共有超过 100TB 的内存和14K的 vcpu 内核。在 Pinterest，我们有近1000个月活跃用户使用 Presto，他们每月在这些集群上运行约40万次查询。ad-hoc 和 scheduled 等集群规模如下（其中 ad-hoc 集群是在 K8S 上运行的）：

用户场景	集群数量	集群大小	Coordinator 配置	Worker 配置
adhoc	2	200	64 core, 488G	43 ~ 48 core, 340 ~ 384G
scheduled	1	165		
Pii and others	2	30 ~ 100		

Presto 以其能够从各种系统中查询数据的能力而闻名。Pinterest 目前主要使用 Hive 数据源。Hive 和 Presto 共享相同的 Hive Metastore 服务。我们的用户通常使用 Hive 来写数据，而使用 Presto 来进行只读分析。此外，由于以下原因，我们最近开始允许 Presto

创建表和插入数据。

- 运行大查询的能力：我们通过对查询的运行时间和它们在 Presto 上处理的数据来限制查询。写入支持提供了一种通过将大查询分解为较小查询来运行大查询的替代方法。每个小查询都可以从先前查询的输出中读取并写入中间表，然后由下一个查询使用。这是处理大型查询的一种更好的方法，因为它提供了易于调试、模块化、共享和检查点。如果一个子查询失败，只需要重新运行该子查询和后续的子查询，而不需要重新运行整个大查询，这样可以节省时间和资源/金钱。
- 支持工作流：用户对 Presto 的处理速度印象深刻，一直要求支持在 Presto 之上定义工作流。 由于只有读取功能，Presto 要么只能在工作流的末尾提供最终输出，要么将 Presto 输出引入工作流系统的内存中，然后传递给下一个作业/执行。这两种方法都有很大的局限性。由于 Presto 支持写，它可以很容易地在工作流中使用。

Pinterest 的每个 Presto 集群在专用 AWS EC2 实例和 Kubernetes pod 都有 Worker 节点。Pinterest 上的 Presto 部署看起来应该与任何大规模 Presto 部署非常相似。 有几个内部组件，即 Presto Controller 和 Presto Gateway，我们将在下一小节中讨论。

Presto Controller

Presto 控制器是一种内部构建的服务，对我们的 Presto 部署至关重要。以下是当前控制器提供的一些主要功能。

- 健康检查（Health check）
- 慢节点检测（Slow worker detection）
- 大查询检测（Heavy query detection）
- Presto 集群滚动重启
- 集群扩容

Presto Gateway

Presto 网关是位于客户端和 Presto 集群之间的服务。它本质上是智能 HTTP 代理服务器。我们是使用 Lyft 开源的 Presto-Gateway (https://github.com/lyft/presto-gateway)。并且在此基础上增加了许多功能，我们计划把这些功能贡献给 Lyft 的版本。该服务使客户端不知道特定的 Presto集群，并支持以下用法。其中一些功能正在积极开发中，我们正在缓慢地将所有客户端转移到 Presto Gateway 上。

- 基于规则的查询路由
- 资源使用限制和当前资源使用情况
- Presto集群整体运行状况可见性

监控和警告

每个提交到 Presto 集群的查询都通过 Singer 记录到 Kafka 相关主题中。Singer 是 Pinterest 开发的日志代理。每个查询在提交和完成时都被记录下来。当 Presto 集群崩溃时，我们将有查询

提交事件，而没有相应的查询完成事件。这些事件使我们能够捕获随着时间的推移集群崩溃的影响。JMX 和操作系统指标通过在所有 Pinterest 主机上运行的 tcollector 记录到 OpenTSDB 中。然后通过 Statsboard（Pinterest 的指标监控 UI）可以实时从 OpenTSDB 中获取监控数据，这对于调试服务问题非常方便。Statsboard 还具有与 PagerDuty 相关的警报机制。

客户端

用户可以通过几个途径与 Presto 进行交互。 最常见的是 DataHub（一种内部 Web UI 工具）、Jupyter 和 Tableau。 但是，有很多自定义工具是通过 Presto 提供支持的。

分析

为了扩大 Pinterest 的 Presto 使用，我们利用从 Presto 集群和 Presto 查询日志收集的数据来推导出一些有用的指标，比如以下几个：

- 哪些表在读的时候很慢？
- 哪些查询一起运行会导致集群崩溃或停止？
- 哪个用户/团队在运行大查询？
- 配置的好阈值是多少？
- P90 和 P99 查询时间是多少？
- 查询成功率是多少？

挑战和解决方案

深度嵌套和巨大的 thrift schemas

Presto 集群中的 Coordinator 对整个集群的运行非常重要。 而且它存在单点故障。直到2018年年中，我们的 Presto 版本还是基于开源 Presto 0.182 版本。 从那时起，我们对 Coordinator 进行了许多错误修复和改进，以更好地应对其关键职责。然而，即使进行了改进，我们发现我们的 Presto 集群的 Coordinator 会因内存不足 (OOM) 卡住甚至崩溃。

崩溃的最常见原因之一是非常大且嵌套非常深的 thrift schemas，这在我们的 Hive 表中非常常见。例如，一个流行且常用的大型 Thrift schema 有超过 1200 万个原子类型和 41 层的嵌套深度。当把这个 Thrift schema 序列化为字符串时，将占用超过 282 MB 内存。另外，我们有近 500 张超过10万个列的 hive 表。

在 Presto 中，coordinator 负责从 Hive Metastore 为 Hive catalog 获取表的模式，然后在它发送给 worker 的每个任务请求中序列化该模式。这种设计使 Hive Metastore 服务不会同时受到来自 Presto workers 的数百个请求的轰炸。但是，当表的模式非常大时，这种设计会对 coordinator 内存和网络产生不利影响。

幸运的是，我们大而深度嵌套的模式问题仅限于使用 Thrift schemas 的表。在我们的部署中，创建了一个 Thrift 模式 Java 归档 (jar) 文件并将其放入 coordinator 和 Presto

集群的每个 worker 的类路径中，并在服务启动时加载。
在日常服务重新启动期间，将创建并重新加载具有更新架构的新 jar。
这使我们能够从任务的请求中完全摆脱 Thrift 模式：相反，只有 Thrift
类名作为请求的一部分被传递，然后 Worker 收到类名之后利用 thrift schema jar 来构建 table schema，这在很大程度上有助于 Presto coordinator 的稳定性。

Slow or stuck workers

Presto 的一部分效率和速度得益于它始终启动 JVM 并准备好开始在 worker 上运行任务。 Presto worker 上的多个查询的多个任务（tasks）共享一个 JVM。
这种共享通常会导致繁重的查询减慢集群上的所有其他查询。使用资源组（resource groups）强制执行内存限制，强制限制查询在给定时间在集群上可以消耗的内存量，在多租户的集群中解决这些问题大有帮助。然而，我们仍然经常看到集群陷入停滞，查询会卡住，worker
并行度会降到零并在那里停留很长时间，通信错误开始出现，查询开始超时。

Presto 使用多级反馈队列（multilevel feedback queue）来确保慢速任务（slow tasks）不会减慢 worker 的所有任务。这可能会导致 worker
随着时间的推移积累大量慢速的任务，因为快速任务将被优先考虑并很快完成；缓慢的 IO 任务（slow IO tasks）也可能在 worker 上累积。 如前所述，我们所有的数据都位于 AWS S3 中，如果 prefix is being hit hard（不太好翻译），S3
可以限制请求，这会进一步减慢任务的运行速度。如果 worker
处于缓慢或卡住（stuck），缓慢会逐渐通过 Presto 集群传播。 等待来自慢速 worker 的上的页面（page）的其他 workers 也会变慢，这进一步导致更多的 worker 变慢。

解决这个问题需要良好的检测和公平的解决机制。我们采取了以下检查来检测 workers 变慢：

- 检查 worker 的 CPU 利用率是否低于集群的平均 CPU 利用率但超过某个阈值，并且这种差异会持续一段时间。
- 检查一些查询是否因内部错误而失败，这表明在一段时间内与超过阈值的 worker 通信时失败；
- 检查 worker 打开的文件描述符数是否高于阈值并超过一段时间。

一旦 worker 符合上述任何条件，Presto Controller 就会将该 worker
标记为关闭状态（shutdown）。 首先尝试正常关闭（graceful shutdown），但是在几次尝试中未能正常关闭 worker 将导致 controller 强行终止这个 worker 所在的 EC2 实例或关闭托管该 worker 的 Kubernetes pod。

多集群资源不均衡

如最上的图所示，我们在 Pinterest 有多个 Presto 集群。 为了有效利用所有 Presto 集群中的可用资源，应将新查询发送到未充分利用的集群，或者必须将未充分利用的集群中的资源移动到将要运行查询的集群。 前者会更容易，但是在 Pinterest，不同的 Presto 集群具有不同的访问模式和不同的特征。某些集群针对在其上运行的非常特定类型的查询/用例进行了调整。例如，在计划集群（scheduled cluster）上运行即席查询，将干扰计划集群的分析任务，并且还会对集群上的查询产生不利影响。查询之间的这种影响使我们更喜欢将资源从未充分利用的集群转移到过度使

用的集群的原因。

将专用 EC2 实例从一个集群移动到另一个集群需要我们终止并重新配置该实例。
这个过程很容易花费接近或超过十分钟。 在 Presto 世界中，10 分钟是很长的时间，我们的 P90 查询延迟不到 5 分钟。 相反，Kubernetes 平台为我们提供了在 Presto 集群中快速添加和删除 worker 的能力。 在 Kubernetes 上启动新 worker 的最佳延迟时间不到一分钟。 但是，当 Kubernetes 集群本身资源不足并需要扩展时，最多可能需要十分钟。 在 Kubernetes 平台上部署的其他一些优势是我们的 Presto 部署变得与云供应商、实例类型、操作系统等无关。

Presto 控制器服务负责在 Kubernetes 上添加/删除 workers。 我们今天在 Kubernetes 上为每个集群统计了一个静态的 workers 数量。
但是，我们计划很快根据当前需求以及这些集群需求的历史趋势自动扩展集群。

集群不正常关闭（Ungraceful cluster shut down）

每天晚上，我们都会重新启动所有 Presto 集群以加载更新的配置参数、Thrift schemas、自定义 Hive 序列化器/反序列化器 (SerDe) 和用户自定义函数 (UDF)。

在不影响任何正在运行的任务的情况下关闭服务的能力是服务的一个重要方面（通常称为优雅关闭（graceful shutdown））。 在开源 Presto 中，无法通过优雅关闭来重启集群。
业界其他公司的 Presto 管理员通过控制到集群的流量来做到优雅关闭。我们也开始在 Pinterest 上使用 Presto Gateway 做同样的事情。 然而，目前，有一些客户端与特定的 Presto 集群通信，并受到集群不正常关闭的影响。即使使用 Presto Gateway，我们仍然有一些客户端会继续与特定的 Presto 集群通信，而无需通过 Presto Gateway，这可能是出于安全原因，或者是因为只有一个集群服务于特定用例。

在 Presto 中，可以优雅关闭 worker。但是，仅凭这一点还不足以确保整个集群正常关闭。我们为 Presto coordinator 添加了优雅关闭的功能，以执行整个集群的正常关闭。当启动集群优雅关闭时，将向集群的 coordinator 发送关闭请求。在收优雅关闭请求时，类似于 Presto Workers，coordinator 将其状态更改为 SHUTTING_DOWN。在这种状态下，Presto coordinator 不接受任何新查询，并在关闭自身之前等待所有现有查询完成。在这种状态下，coordinator 对任何新查询都返回一个错误，通知客户端集群正在关闭，并通知它们在一段时间内再重试，通常大约是允许的最大查询运行时间。这种快速失败且只包含信息消息的行为比在客户端之前看到突然失败的行为要好得多，提示他们简单地重试查询只是为了再次看到这些故障。将来，我们计划实现无需重新启动 coordinator 即可重新加载 jar 的功能，并使一些配置参数动态化以避免频繁重新启动集群的需要。

总结

Presto 在 Pinterest 使用非常广泛，并且在分析中发挥了关键作用。 作为非常流行的交互式 SQL 查询引擎之一，Presto 发展非常迅速。 Presto 的最新版本在稳定性和可扩展性方面有很多改进。 但是，对于 Pinterest 规模，我们必须解决一些问题才能成功操作和使用 Presto，例如优雅的集群关闭、大型深度嵌套的 thrift 模式的处理、LDAP

身份验证器中的模拟支持、慢 worker 检测。
其中一些也可以使社区受益，我们计划对这些进行开源。

未来，我们希望继续提高可靠性、可扩展性、可用性和性能，例如滚动重启（rolling restarts）、无需重启集群即可重新加载 jar 以及为用户提供集群资源利用率的可见性。
我们也对任务的按需检查点非常感兴趣，以实现 Amazon EC2 Spot
实例的无缝使用，并使我们的用户能够在不等待查询完成的情况下获得查询运行时估计。

本文翻译自：Presto at Pinterest