Presto 在车好多的实践

本文作者：车好多大数据 OLAP 团队-王培，由车好多大数据 OLAP 团队相关同事投稿。

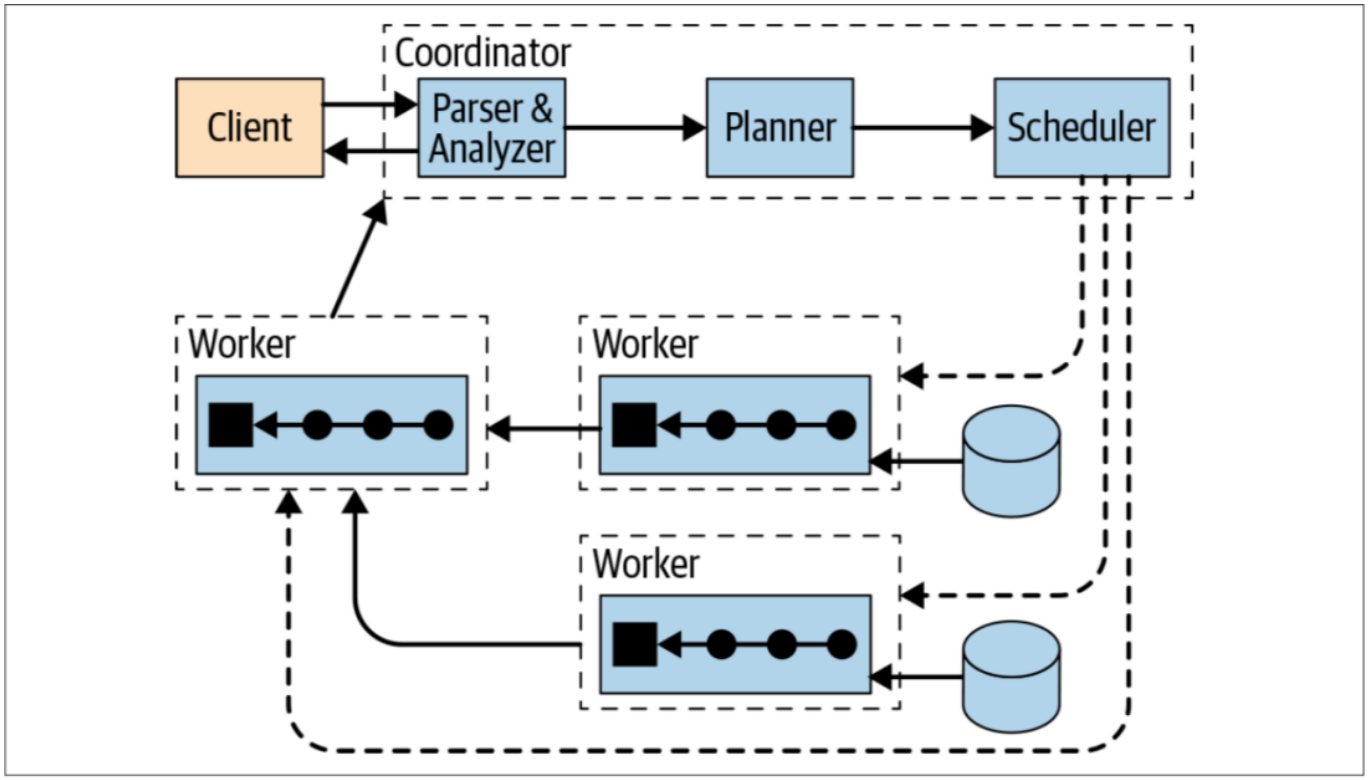
Presto 简介

简介

Presto 最初是由 Facebook 开发的一个分布式 SQL 执行引擎，
它被设计为用来专门进行高速、实时的数据分析，以弥补 Hive
在速度和对接多种数据源上的短板。发展历史如下：

- 2012年秋季，Facebook启动Presto项目
- 2013年冬季，Presto开源
- 2017年11月，11888 commits，203 releases，198 contributors
- 2019年1月，Presto分家，目前有PrestoDB和PrestoSQL两个社区

架构



Presto 是典型的 MPP 架构，由一个 Coordinator 和多个 Worker 组成，其中 Coordinator 负责
SQL 的解析和调度，Worker 负责任务的具体执行。可配置多个不同类型的

Catalog，实现对多个数据源的访问。

Presto 在车好多的落地

Presto 大概在 2017 年底 2018 年初左右开始在车好多落地使用，主要是为满足集团的 Adhoc 查询和报表而服务。落地三年左右，经过了数次的版本升级和一次大的架构升级，迭代如下：

目前有专门提供 Ahoc 查询的大集群，以及一些业务专有小集群相互配合提供服务，满足集团不同 SLA 的查询需求，总体使用情况：

用户数/天	600+
查询数/天	150k+
扫描数据量/天	300+ TB
客户端	cli/jdbc/python/go/php

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

初期落地

初期选择的版本是 0.153，当时根据情况有以下几个需求：

- 隐藏 Coordinator 真实地址
- 接入有规范或者管控
- 简化管理员运维

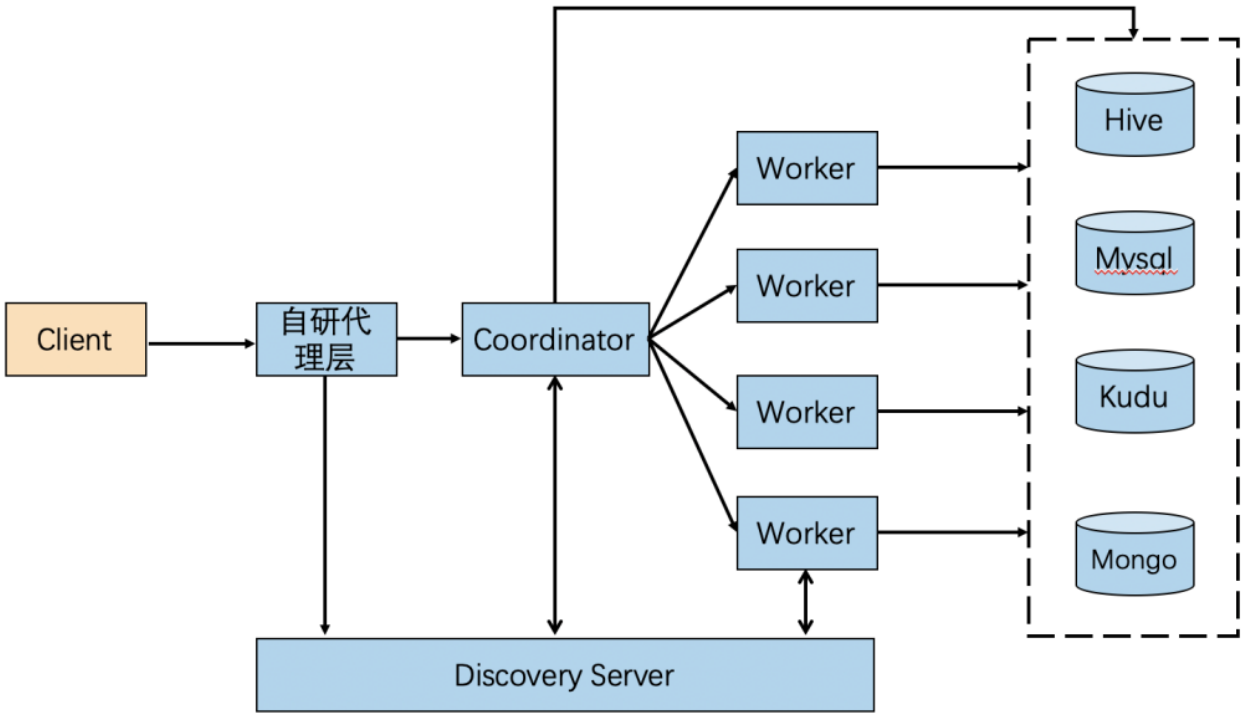
开源社区版本直接暴露 Coordinator 地址给客户端提供服务，重启 Coordinator 会 Fail Query。为了满足以上需求，我们实现了以下关键功能点：

- 客户端和服务端之间加一层代理
  代理层的作用不仅隐藏了 Coordinator 真实地址，而且可以根据需求设置一些客户端接入规范，以便能区分接入方式/类型等。我们还在代理层附加了下面两个主要功能：在每一个 Query 结束时，会记录其所有信息并发送到 Kafka，最终落入到

Hive，即日志审计，方便管理员后续分析/治理；监控一些 Query 指标，在超出阈值时主动 kill Query，提高集群稳定性。

- 发现服务单独部署
  发现服务没有采用内嵌在 Coordinator 中的方式，而是采用单独部署方式，不仅有助于代理层灵活的获取集群地址，不会受限于某个 Coordinator，而且在管理员运维时发挥很大的作用：在集群中启动第二个 Coordinator 角色，代理层会自动把流量切换到新启动的 Coordinator，待旧 Coordinator 原有的 Query 运行结束再切换回来，达到用户无感知重启 Coordinator 的目的，再加上本身 Worker 节点 支持优雅下线，那么整个集群的无感知运维就可以轻松实现，为管理员运维带来极大的便利。

整体架构大致如下：



根据实际的场景需求，除了 Hive 之外，Mysql 是接入最多的数据源，后续又接入了 Kudu（版本升级后才接入）、Mongo、PostgreSQL 等数据源，方便用户利用 Presto 进行跨数据源的关联查询。这也是我们当时选择 Presto 组件的主要原因。

一开始采用了和 Hadoop 集群混合部署的模式，但是考虑到资源竞争，很快切换到物理机单独部署：

- Coordinator 节点不作为计算节点，只作为协调节点；
- 每台物理机只部署一个 Presto 节点，无其他任何竞争服务；

- JVM 配置为 G1 回收器、最大堆内存为物理内存的 75%；

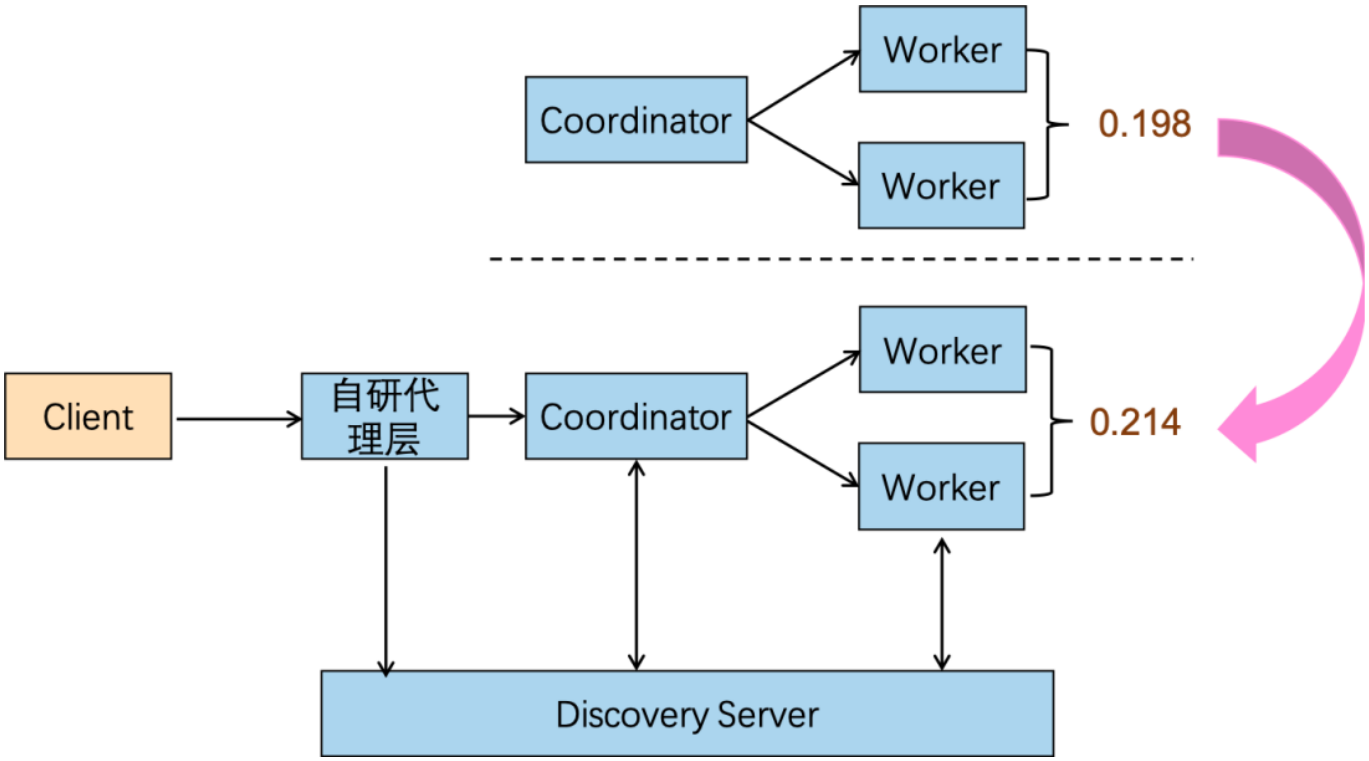
以为 Presto
集群的稳定性会大大提高，但是服务上线后，我们就遇到了一个不小的挑战：服务经常 OOM，很不稳定。经过调研，我们采取以下措施来优化 OOM 问题：

- 设置堆外内存最大使用量 MaxDirectMemorySize
- 设置 glibc 的参数 export MALLOC_ARENA_MAX=1

通过以上主要优化，我们 Presto 集群的内存使用值常年比较平稳，OOM 问题大大缓解。

中期迭代

经过初期的稳定阶段以后，为了跟进社区，开始着手做版本升级的事情。基于单独部署的发现服务和代理层切换流量的功能以及客户端的向后兼容，我们成功实现了用户无感知升级。



- Gracefully 停掉一半 Worker
- 升级一半 Worker 到新版本
- 启动一个新版本的 Coordinator
- 等待老版本 Coordinator 查询执行完成
- 关闭老版本 Coordinator 以及剩余老版本 worker
- 升级剩余 Worker 到新版本

在上述的方案中，重启 Coordinator 或者升级版本的过程，会出现一个集群中同时存在多个

Coordinator 的情况，日志会出现 com.facebook.presto.execution.SqlTaskManager Switching coordinator affinity from xxx to yyy 类似的警告,这种状况长时间内是有资源调度死锁风险的，然而在我们的状况中，不论是重启还是升级都是在短时间内（分钟级别），所以稳定性还是可以保证的。随着用户和任务的增多，Presto 在车好多作为 Adhoc
查询引擎慢慢流行开来，但随之几个核心问题暴露出来：

2.1 无权限管控

背景

Presto 接入的底层数据源种类多，而且数据量大，覆盖车好多集团相当一部分业务线的业务数据。没有权限管控的机制，任何一个用户都可以通过 Presto
访问底层数据源的全部数据，这对数据安全来说是一个很大的隐患。

解决方案&效果

由于底层对接的数据源种类不统一，比如 Hive、Mysql、Mongo
等，在数据源层做权限当时有以下几方面限制：

- 数据源层面，有些数据源开启权限验证，而有些没有开启；
- 不同类型数据源支持权限的策略不一样，无法统一；
- 在 Presto 里不是所有的 Connector 都支持 Impersonate[1]；
- 基于以上限制，最快速、最适合的方案就是在代理层做权限管控的逻辑。
- 改造 Presto 不同类型（cli、jdbc、python、go
  等）的客户端，支持公司内部账号体系，完成认证过程；
- 基于公司权限/流程系统，改造一套适合 Presto 的权限管理系统；
- 在代理层实现鉴权逻辑；

这个权限管理方案实现简单，落地后比较符合公司的使用需求和场景，结合代理层的日志审计功能，这样管理员对 Presto 集群的所有用户以及 Query
执行情况都有了全面详细的了解。这个为后续的任务治理提供了非常宝贵的数据支持。

2.2 新增 Catalog 频繁，运维压力大

背景

车好多集团关于车有多个业务线，比如收车、卖车、车后、金融等方面，每个业务线都有自己的业务数据，有些时候需要跨业务线的OLTP数据库（Mysql，Mongo、PostgreSQL等的只读从库）进行关联查询，需要新增对应的 Catalog。Presto 对于新增 Catalog
是需要重启集群的，所以这对于管理员来说有很大的运维压力。

当然从长远来看，还是要将多数据源统一入Hive，有HiveMetaStore服务统一管理所有元数据，运维和管理都会方便很多。

解决方案&效果

我们修改了部分源码，Presto-Server 对外提供 Restful 接口可在线添加新的 Catalog。对于更新和删除 Catalog 的情况，比较低频，为了稳定，还是采用重启集群的方式。这个功能的实现大大减轻了管理员的运维压力，也减少了上线带来的稳定性风险。

2.3 棘手的排队问题出现

背景

经过了一年多的迭代，Presto 在车好多集团内部成为了提供 Adhoc 查询的核心组件，数十个业务线的数百名用户都重度依赖 Presto 来实现他们的分析需求或者报表结果，基本上集群每天有 600+用户（数据分析师、运营、市场、产品等），高峰期每秒提交数目最大能达到百级别。在这样的一个情况下，高峰期任务排队的情况就会出现并且越来越严重，严重影响了用户的使用体验。

解决方案&效果

首先的想到的是任务治理

- 大查询限制：导致集群排队的主要原因是大查询（耗费计算资源多的 Query）长时间占用集群资源不释放，集群最大运行 Query 数目被打满，后续提交的 Query 只能排队。为了限制大查询，我们下调单个 Query 的最大运行时间、最大扫描分区数目、内存使用最大值、stage 数目等，让集群资源快速流转起来；
- 单个用户 Query 数目限制：我们下调单个用户的最大运行数目以及最大排队数目，防止单个用户提交过多查询占满集群资源，其他用户没有机会提交；
- 优化 SQL：我们根据一些规则，给出 SQL 优化的建议，比如：避免笛卡尔积、distinct 滥用、非等值 join 等情况，并推动用户优化 SQL；
- 推动上层 BI 工具缓存结果：为了方便用户使用，有一些 BI 工具来对接 Presto，有多个用户会查看同一张报表，基于这样的情况，没有必要每次查看都要发起一次查询，工具层缓存这个结果，对底层 Presto 的压力会大大缓解；
- 推动中间表的建设，优化查源表的情况，减少计算资源的浪费；
- 每周统计出各个部门的资源使用账单&资源消耗排名 Top N 的用户，并通知，这是推动用户优化任务重要的数据来源；

其次，增加资源，这也是必然要尝试的一个方法。然而由于一些客观原因，比如：成本、机房初始容量规划等，无法给集群进行提供充足的资源，只能小规模有限扩容。

通过以上两个方面的优化，尤其是任务治理，排队情况得到缓解，然而总会有一些新用户会提交一些不合理的任务，因此任务治理是一项长期持续的工作。资源方面，没有条件新增，那么就只能在存量资源上想办法。

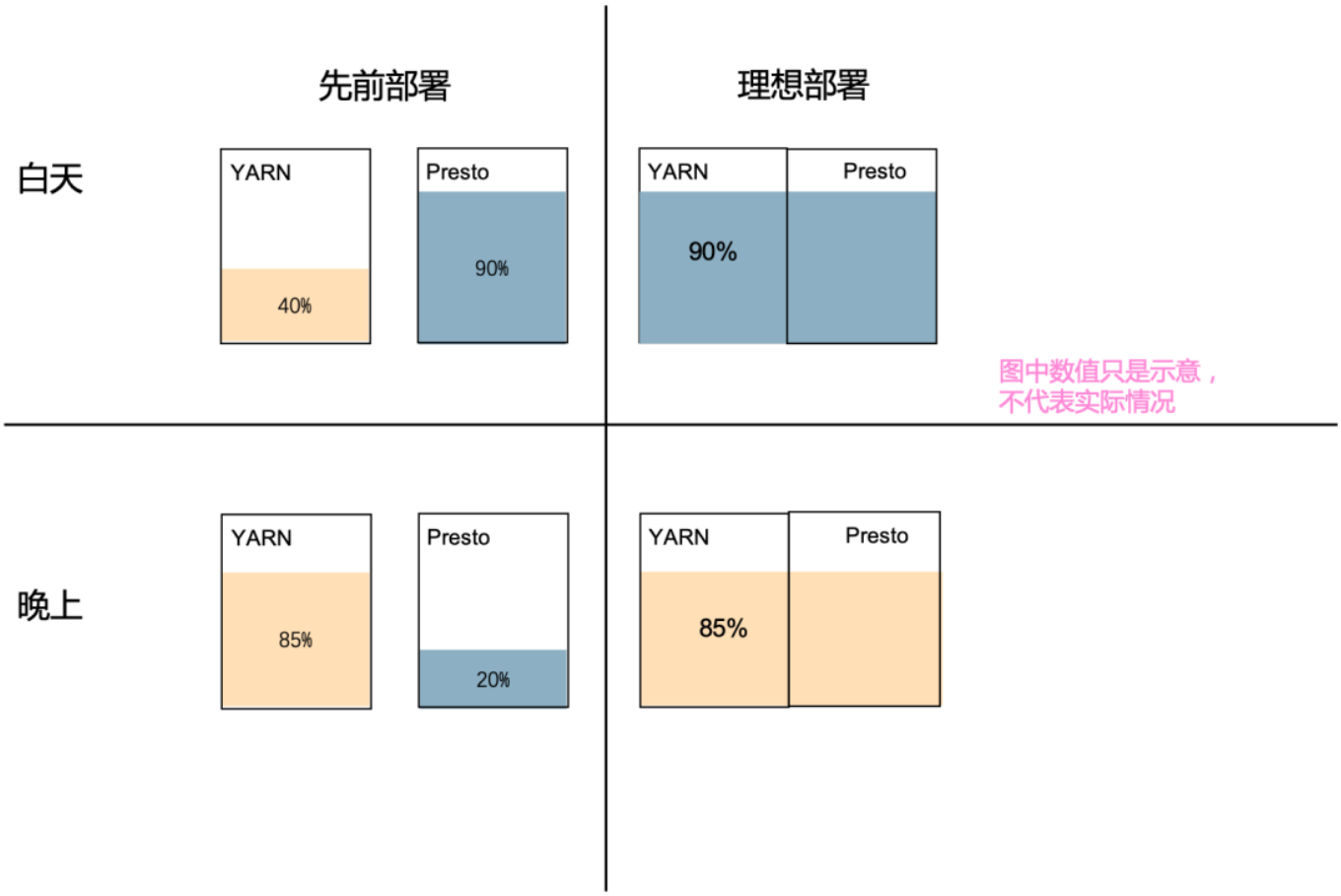
Presto 在车好多的架构升级

弹性Presto 方案（Presto on YARN）

我们调研弹性Presto方案主要基于以下2点：

- 中期任务治理后，排队问题依然严重，希望有更多的资源能提供给Presto，但由于Adhoc查询场景的特殊性，白天资源利用率高，晚上这部分资源又会闲置；
- 一键快速的拉起、删除集群，以及一键快速的扩容、缩容能力，对管理员来说是刚需，能极大提高管理员的工作效率；

在当前大数据架构的概览下，我们发现 Hadoop 中 YARN 集群的夜间批处理任务和 Presto 集群白天的查询任务是完全错峰的，有典型的潮汐现象。



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

所以我们开始考虑 Presto on YARN 的弹性技术方案，总体来说，收益很多，总结如下：

- 可以为用户快速搭建专有集群，达到资源隔离，提升服务质量的效果；
- Presto 集群可以利用 YARN 集群白天空闲的资源，大大缓解资源紧张的问题；
- YARN集群也能利用晚上Presto闲置的资源来扩充批处理任务的资源
- 大数据整体机器全天的资源利用率会大大提高，节省成本；

由于 Hadoop 集群整体版本是 2.7.x，经过调研，需要使用 Slider 这个已经组件来实现 Presto on YARN，总体来说调研过程比较顺利。由于 Slider
项目已经不维护，资料相对较少，过程中请教了吴彪前辈一些问题，这里衷心感谢！

Presto on YARN 方案有以下注意点：

- 如果 YARN 集群不支持 label 功能，可以采用动态端口的方式解决单个 NodeManager 上调度多个 PrestoServer 节点的端口冲突问题；
- YARN集群要开启CGroup，否则CPU、MEM不受控制；
- appConfig 中可设置"site.global.data_dir": "${AGENT_LOG_ROOT}"来 解决一台 NodeManager 上两个 PrestoServer 目录冲突的问题；
- 单个 PrestoServer 的资源受限于 YARN 集群中 Container 最大资源的限制；

在使用过程我们也发现了一些 Slider 的问题：

- 某些情况下节点短时间无法自动拉起。在队列资源比较紧张的情况下，节点会因资源被抢占而被 kill，Slider 会把当前 NodeManager
  加进黑名单，如果重试次数足够多直到把所有 NodeManager 都遍历一遍，那么所有 NodeManager 都会被 Slider
  加进黑名单，虽然黑名单有超时机制，但是在黑名单失效前节点是无法被拉起的。
- Slider 把 YARN 的优先级和节点亲和性揉在一起，造成重启后实际节点优先级倒置；
- Slider 上报给 YARN 的应用诊断信息过长，可能导致无法写入 zk，将 RM 阻塞在 zk 写操作，最终搞挂 RM；

将以上问题都解决以后，Presto on YARN 的方案达到了可用、稳定的状态。

代理层 Presto-gateway

有了 Presto on YARN 方案以后，结合 Presto
集群晚间有一些定时任务、架构演进稳定性以及后续规划的考量，考虑采用 物理和 on YARN 的多集群模式来改善资源状况。如果采用多集群的架构，有一个重要的点需要考虑：Presto中，一个Query执行周期内需要客户端和服务端进行多次的HTTP请求，在多集群模式下，如何保证同一个Query的请求都分发到同一个集群呢？

针对上述问题，经过调研，发现普通现有的Nginx算法比如IP
Hash[2]等无法满足需求，还是需要在代理层进行改造。这个代理层需要满足以下功能：

- 保存每个Query和后端集群地址映射状态；
- 任务分发；
- 灵活控制每个集群的激活状态；

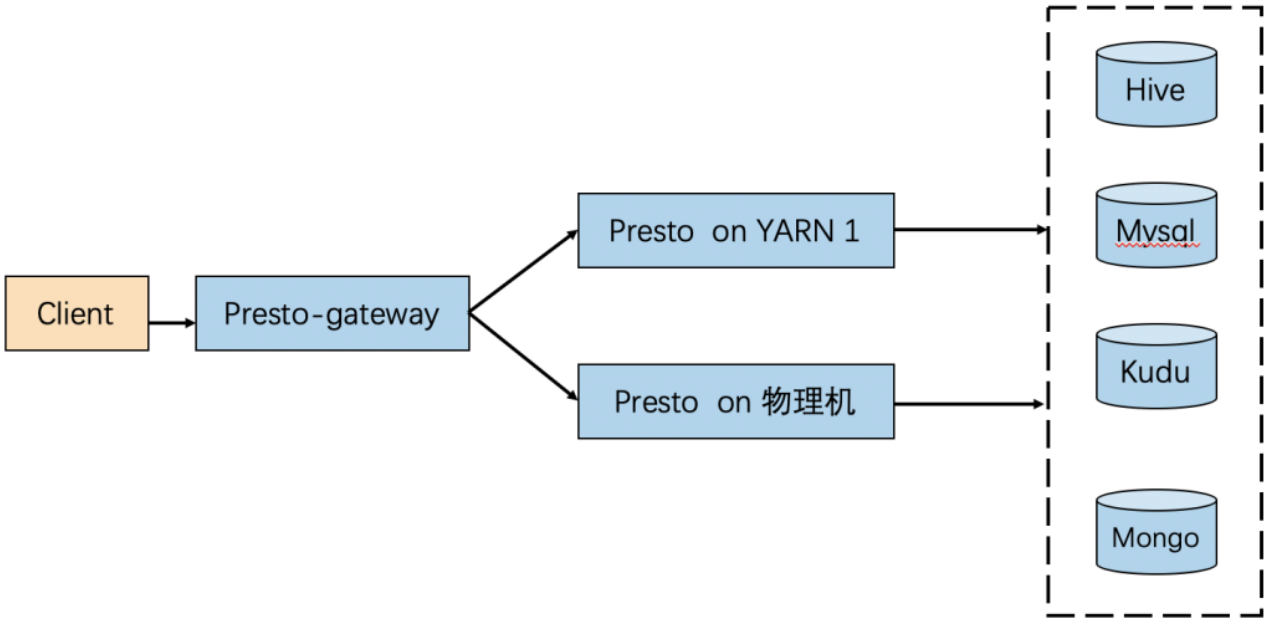
调研过程中发现有个开源的Presto-gateway[3]专门做了上述的事情，从前面的架构已经知道，我们有一个现成的代理层，但是现有代理层没有覆盖上述功能。经过工作量、架构扩展性等方面的评估，决定用Presto-gateway替换自研代理层，并做一些落地改造：

- 原有代理层权限、监控相关功能的添加；
- 每个查询和后端集群地址的映射关系由原来的 Guava Cache 修改到 Redis 中，Presto-gateway 彻底无状态，可多实例部署保证 HA；
- 增加后端探活功能，检测某个集群功能异常，从分发列表中移除；
- 增加分发策略，在原来的随机策略基础上增加了平滑加权轮询、指标动态策略；

后续也会考虑把一些公共的功能，比如多实例HA、探活、分发策略等回馈给Presto-gateway社区

多集群部署

多集群方案全部准备好以后，我们首先为一些需要专属集群保障 Query 不受其他查询影响的用户试用这个方案。经过试用以后，这些用户反馈良好，开始着手改造公司的大Adhoc集群，采用了原有物理集群和 Presto on YARN 新集群同时提供服务的模式，其中 Presto on YARN 集群只在白天提供服务，晚上下线为批处理任务让出资源。结构如下：



上述架构目前已经稳定上线运行，白天集群排队情况大大缓解，用户体验也大大提高。后续会逐渐 All in Presto on YARN，把物理机集群的资源添加到 YARN 中。

总结与展望

Presto 在车好多落地将近 3 年的时间，期间对于源码的改造一直保持克制谨慎的态度，主要基于升级方面的考量。经过 Presto on YARN 架构升级，Presto 组件资源有了很大的灵活性，也具备了快速部署独立集群提供高 SLA 服务的能力，让 Presto 在车好多更好的发挥作用。

未来将从以下方面继续推进 Presto 的建设：

- 版本升级到 PrestoSQL，近期底层 HDFS 已经升级到 3.X 版本[4]，会根据情况开启 EC

功能，作为上层计算组件，Presto 需支持读 EC；
- 根据资源指标实时对 Presto on YARN
集群动态扩缩容，资源管理将更加精细，用户体验也会大大提高；
- Presto on Hudi 等新技术的探索，为引入数据湖技术做储备；
- 有机会的话，针对特定需求范畴场景的BI，用Presto on Alluxio加速；

References:

- [1] https://docs.starburstdata.com/latest/connector/starburst-connectors.html
- [2] https://www.nginx.com/products/nginx/load-balancing
- [3] https://github.com/lyft/presto-gateway
- [4] HDFS 2.x 升级 3.x 在车好多的实践