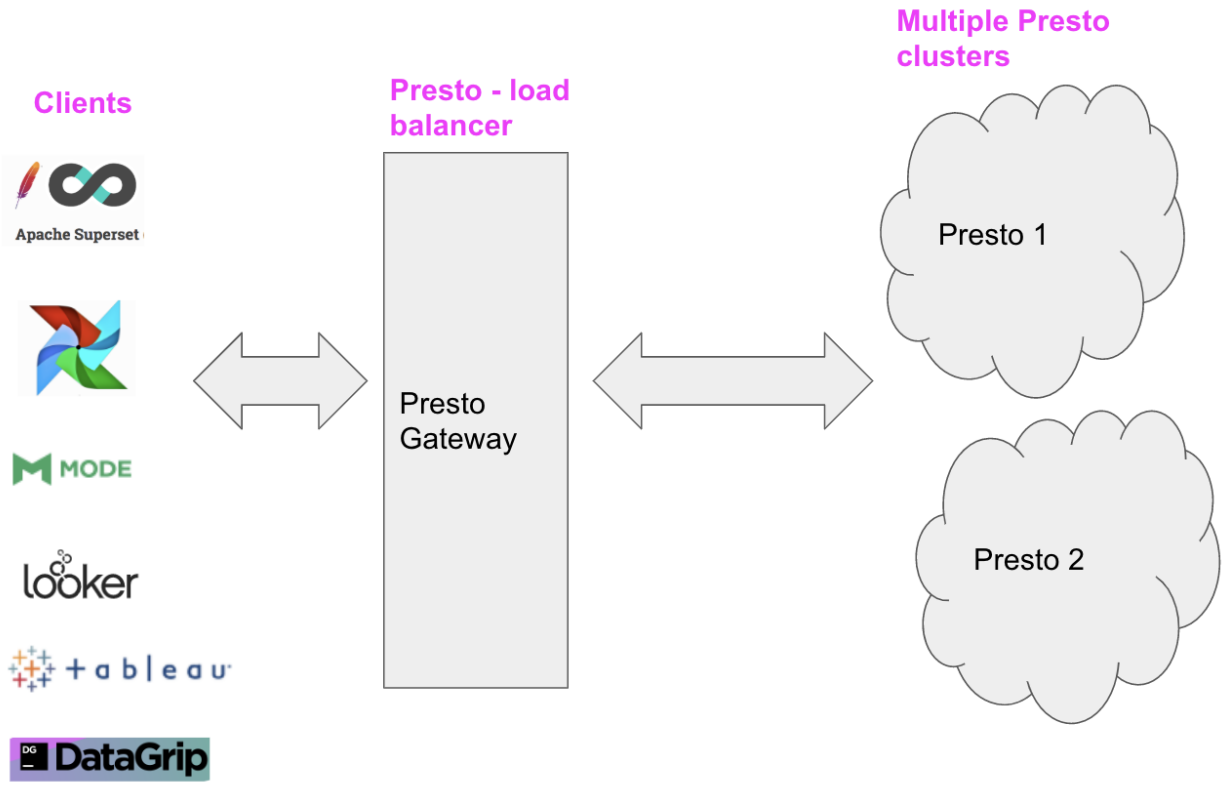
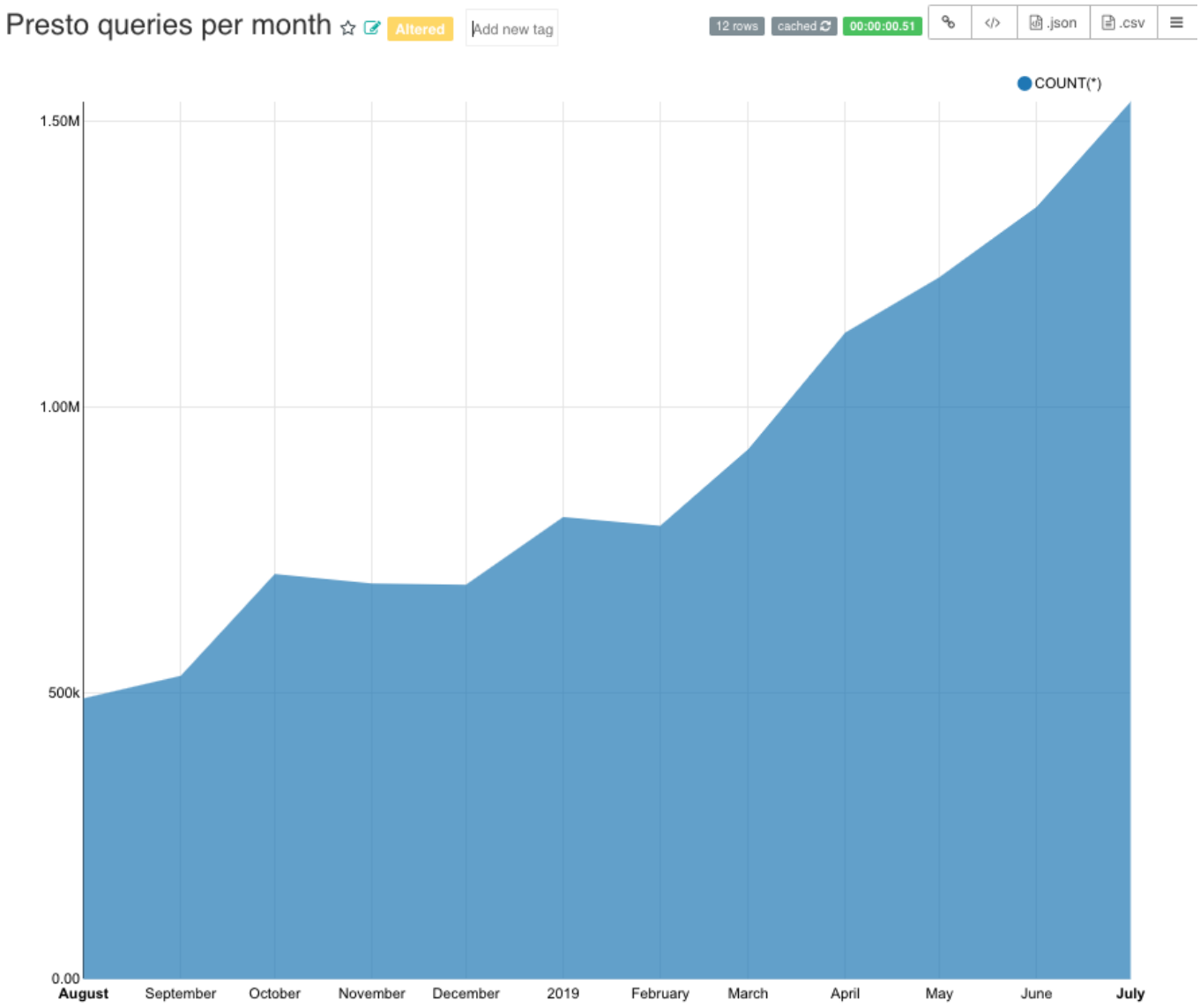
Presto 在 Lyft 的实践

2017 年初，我们开始探索 Presto 来解决 OLAP
用例，我们意识到了这个惊人的查询引擎的潜力。与 Apache Hive
相比，它最初是一种临时查询工具，供数据工程师和分析师以更快的方式运行 SQL
来构建查询原型。 当时很多内部仪表板都由 AWS-Redshift 提供支持，并将数据存储和计算耦合在一起。我们的数据呈指数级增长（每隔几天翻一番），这也需要频繁的存储扩展。由于存储与计算相结合，任何维护、升级都需要停机时间，并且扩展节点使查询变得极其缓慢（因为大量数据在节点间移动），我们需要一个存储和计算分离的系统，这就是 Presto
非常适合我们的用例的地方。我们已经设置了以 Parquet 格式存储事件数据的管道，并且通过 Hive 来访问数据，我们很容易地将 Presto 添加到这个架构中。

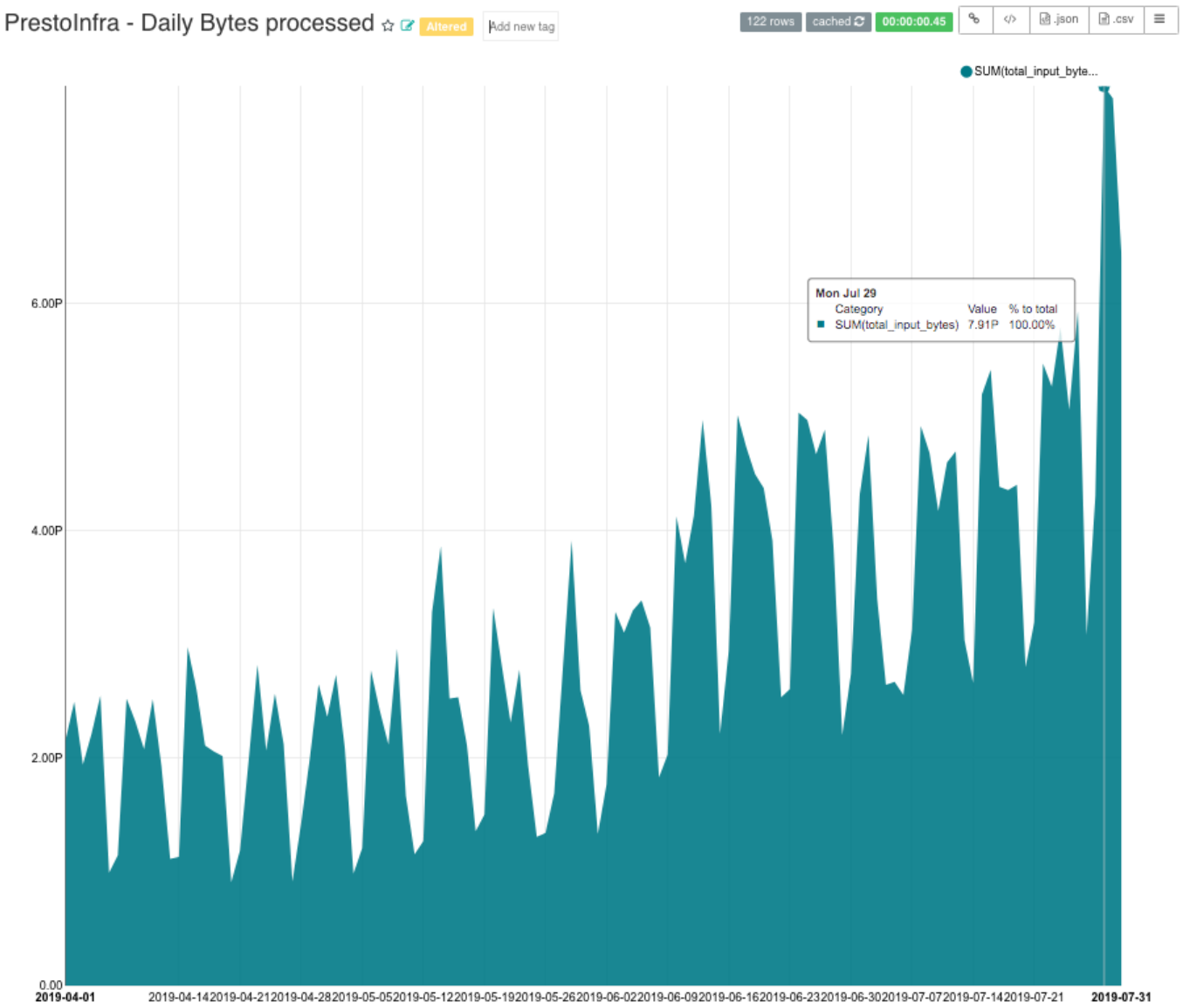


如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

现在，数千个仪表板（dashboards）由 Presto 提供支持，每周约有 1500
名活跃用户在这个平台上运行数百万次查询。截至今天，我们有 60 PB
的可查询事件数据存储在基于 S3 的数据湖中，并且每天使用 Presto 扫描大约 10 PB
的原始数据。以下图表显示了 presto 使用增长的时间线。



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



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

从上图可以看出，在过去 4 个月中，每日原始数据扫描量增长了 4 倍。

Presto 客户端

Lyft 用户使用查询工具（例如 Apache Superset、Mode、Looker、Tableau、Jupyter notebooks）和一些机器学习模型的内部 ML 工具来使用数据。所有这些工具都通过 Presto Gateway 连接到 presto，这些客户端发送的查询在多个 Presto
集群之间统一进行负载均衡。通过 Presto
Gateway，我们可以实现零停机升级，对通过这些客户端查询的用户/应用程序透明。

启动：构建、发布和部署

我们在 lyft/presto 仓库下面 fork 了 prestosql/presto 仓库的代码，并创建了 lyft-stable 分支（如果需要，可以额外的分支）。我们有一个私有存储库，我们使用 saltstack 和 aws-orca 脚本将服务部署到我们的环境。在这个私有分支中，我们添加了特定于我们环境的额外依赖项，并添加了在每次更新或拉取请求时运行的集成测试。我们已经对这个 repo 进行了 docker 化，并使用 Docker 容器作为开发环境。在每次有新的提交时，我们都会通过 Jenkins 触发针对开发环境的集成测试。

以下是我们在将 Presto 投入生产时使用的各种组件：

- Lyft 的 presto 查询日志插件：我们添加了一个基于 Presto-Event 侦听器框架的查询日志和阻塞插件。 当新查询到达时，我们使用 EventListener 进行拦截，并阻止一些我们认为对系统有害的查询类型（例如，一些工具主动缓存列名和查询 system.jdbc.columns 或 catalog.information_schema.columns 并导致额外的系统负载）。我们在 queryCreated 和 queryCompleted 事件中记录查询统计信息，这有助于我们分析成功率、延迟、各种类型的失败及其根本原因等。
- Presto UDF：如果 Presto 内置函数无法解决用户的场景，我们会让用户根据他们的需要添加自定义 UDF。一些用户根据他们的用例添加了自定义地理函数。
- 基于 Python 的统计信息收集：我们使用 datadog-agent 库添加了多项检查来收集系统/jvm/进程 统计信息指标并推送到 statsd。 我们有 statsd 收集器，可以收集指标并将其进一步推送到 Wavefront。 我们设置了各种警报，并在出现问题时进行报警。
- 测试套件：在将新的 presto 版本投入生产之前，我们会运行多种类型的测试来确保候选版本的质量。 以下是我们为确保每个版本的质量而运行的测试类型。
    - 集成测试套件——针对每个 build 版本运行 Table-CRUD 和 UDF 测试，在 Jenkins 构建任务中启动 devbox docker 容器。
    - Replayer 测试套件——在将发布候选版移到到 pre-prod 之后，我们启动了为期一天的测试，其中包含之前有价值的查询，以与过去执行查询相同的方式进行执行。我们使用我们的事件日志管道提供查询日志，并将性能与之前记录的数据进行比较。
    - Verifier 测试套件——与 replayer 类似，我们使用 presto-verifier 在同一份静态数据集上运行旧版本和新版本的代码。如果性能结果没有下降，我们将使用此版本。
- PrestoInfra：配置和脚本的集合，用于构建和发布可部署的 artifacts 和 saltstack 脚本以推出特定于环境的部署。
- PrestoProxy：具有 lyft 特定路由规则和覆盖的自定义 presto-gateway 。

Presto 生产环境和配置

我们正在运行多个 presto 集群，通过 presto-gateway 共享负载，每个集群有 100 多个工作节点。 节点规格如下：

Presto type	AWS Node type	Number of nodes
Presto Proxy/Gateway	c5.2xlarge	1
Coordinator	r5.12xlarge	1
Worker	m5.12xlarge	100 - 150

Presto 配置

在每个集群上，我们设置了 25 个最大查询并发和 200 个最大查询排队，每个运行最多 4 个查询，每个用户最多有 20 个查询排队。每个查询最多运行 30 分钟，每个查询的 execution 的时间为 10 分钟。 我们每天轮换整个集群一次，以避免长时间的老一代 GC 堆积。我们让每个查询最多扫描 1 TB 的数据，每个工作节点最多加载 10 GB 数据。
以下是实现这些设置的确切配置。

Config file	Name	Value	Notes
etc/config.properties	query.max-memory	1 TB	Max allowed input data that's loaded into presto per query.
	query.max-memory-per-node	10 GB	Each worker node can load up to this much data per task.
	query.max-run-time	30 mins	Max toal run (queue + exec) time
	query.max-execution-time	10 mins	Max allowed execution time

在 Presto gateway 协调器中，可以为每个协调器分配一个路由组，设置 X-Presto-Routing-Group header 会将请求路由到该路由组下的集群之一。
我们有一个集群，它以扩展的内存限制的方式运行，分配到 nolimit

路由组。用户必须在查询中添加注释 `-highlimit` 作为提示，以指示资源繁重的查询，然后 presto-gateway 将会把那个查询路由到更高资源限制的集群。

JVM 配置

我们在 presto 节点上运行 Java 11。 Java 11 为 old gen 阶段提供了并行 GC，显着减少了垃圾回收时间。

以下是我们运行 presto 进程的 JVM 配置。随着 Java 11 的升级，我们能够在几秒钟内降低 Old gen GC 暂停，而在 Java 8 中，这个过程需要 400 秒，每隔一段时间就会破坏服务。

```
-Djdk.attach.allowAttachSelf=true
-Xlog:gc:{{ presto_data_dir }}/var/log/gc.log
-XX:+UseG1GC
-XX:+AggressiveOpts
-XX:ReservedCodeCacheSize=150M
-XX:ErrorFile={{ presto_data_dir }}/var/log/java_error.log
-Xmx{{ max_heap_size_mb }}m
-XX:MaxGCPauseMillis=2000
-XX:InitiatingHeapOccupancyPercent=30
-XX:OnOutOfMemoryError="kill -9 %p"
{% if enable_jvm_debugging -%}
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005
{% endif -%}
{% if optimize_gc -%}
-XX:ParallelGCThreads=30
-XX:ConcGCThreads=8
{% endif %}
```


max_heap_size_mb 的值为 (0.6 * node_memory_mb)，在 worker 节点上为 114 GB。

Presto 节点回收

尽管进行了所有优化，但 presto 会随着时间的推移消耗主机资源，并且根据我们的经验，我们了解到服务运行的时间越长，它就会变得越慢。随着越来越多的查询在集群上执行，老一代的 GC 暂停时间随着时间的推移而增加，我们也注意到，与旧集群相比，新集群产生更好的查询性能。牢记这一点，我们将基础架构设计为每 24 小时回收一次每个集群中的所有节点。

我们使用 PrestoGateway 的 deactivate API 在计划关闭时间前 30 分钟禁用集群，并在计划启动时间后 30 分钟通过提供无停机维护的 cron 激活集群。由于我们为每个查询设置了 30 分钟的最大运行时间，以确保在这些操作期间不会丢失查询。我们正在使用 aws-orca 脚本来触发 AWS 的 ScheduledActions，以按照给定的时间表启动或关闭整个 presto 集群。

这是用于在下午 5:20 关闭集群并在太平洋时间晚上 7 点重新启动的 salt 脚本：

```
{% elif cluster_label == 'sch0' %}
- scheduled_actions:
    scale_down_at_evening_everyday:
      desired_capacity: 0
      min_size: 0
      max_size: 0
      # This is 5:20 PM everyday Pacific, cron is in UTC
      recurrence: "20 0 * * *"

    scale_up_at_evening_everyday:
      desired_capacity: {{ asg_details.min_size }}
      min_size: {{ asg_details.min_size }}
      max_size: {{ asg_details.max_size }}
      # This is 7:00 PM everyday Pacific, cron is in UTC
      recurrence: "0 2 * * *"
{% else %}
```


Presto Gateway

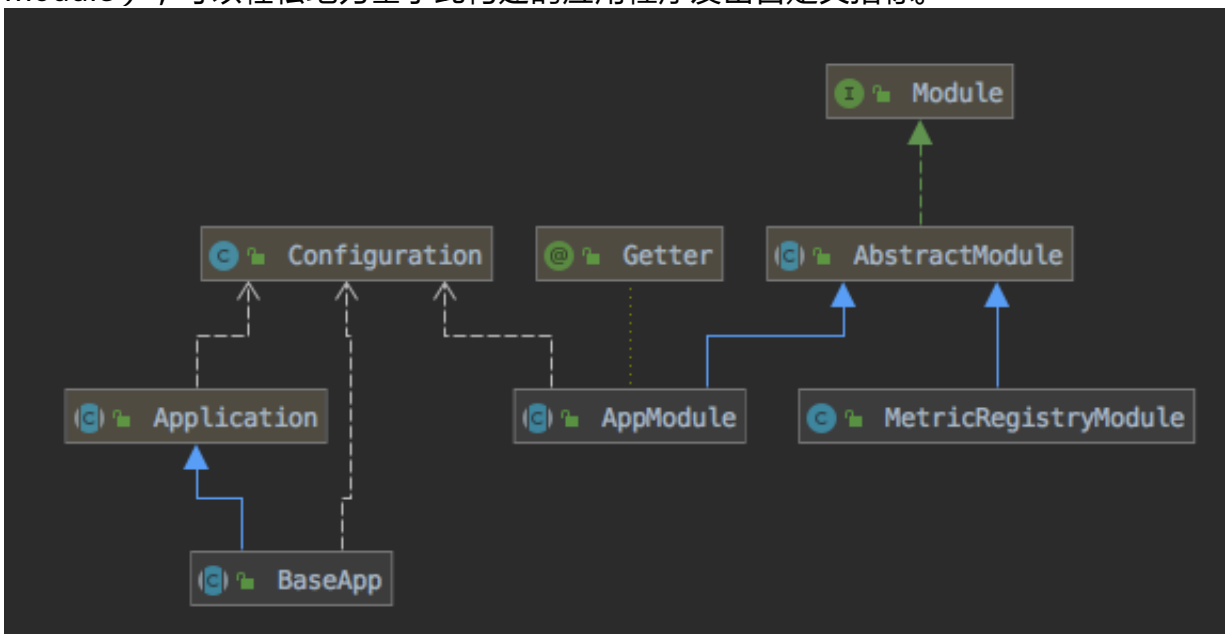
Presto-Gateway 是用于多个 presto 集群的有状态负载均衡器、代理和路由器，它提供对底层 presto 后端的透明访问，而无需更改协议。它最初是一个代理服务项目，以一种安全的方式将 presto coordinators 暴露给外部 BI 工具，如 Mode 和 Looker。随着查询量的增加，我们面临着更频繁的中断，因为单个集群无法处理负载。我们很快意识到我们需要多集群设置。因此，我们为每个这样的查询工具专门分配了一个集群。尽管它降低了中断和事故的频率，但我们仍然没有实现零停机维护，并注意到一个集群相对空闲，而另一个集群则有大量的作业在排队。那时我们决定实现一个真正的代理负载均衡路由器。由于 BI
工具是外部的，它们通过部署在我们网络中的代理访问 presto 集群，这些代理无法遵循 HTTP 重定向，因此我们需要实现真正的代理路由器和负载平衡器。

我们使用了 JettyProxy，它是一个代理服务器，允许绑定自定义过滤器和插入代理处理程序（proxy-handler）。 我们在 Proxy Handler 中实现了路由规则，它允许我们拦截 HTTP
请求和响应，从而允许检查查询、源、headers
等，并基于此我们可以选择后端主机来服务查询请求。客户端发送的请求有两种类型
1、新的查询提交请求；2、对先前提交的查询的后续请求。 PrestoGateway proxy handler 将查询 id 缓存到后端映射，以便它可以将后续请求转发到运行原始请求的集群上。

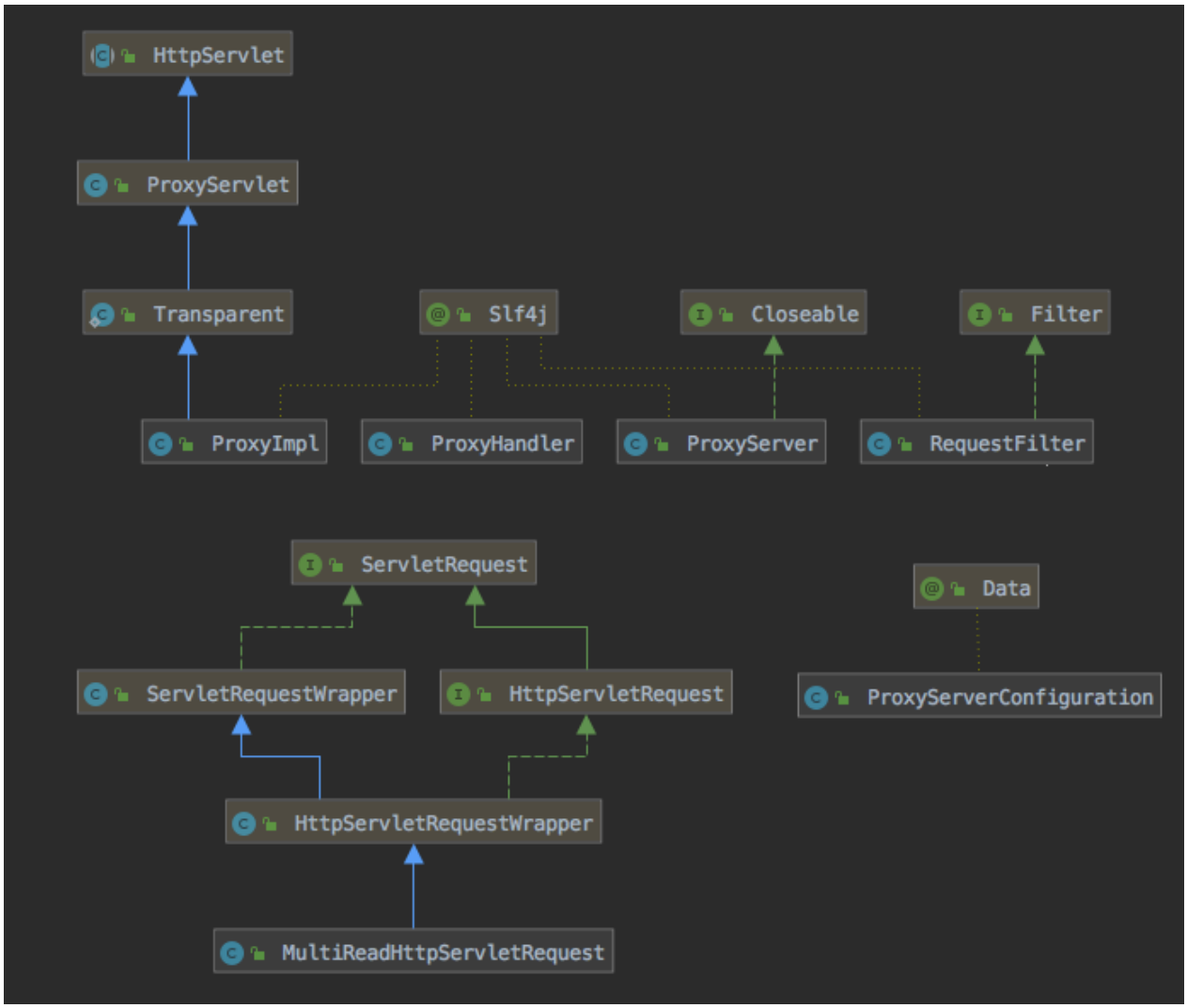
Presto Gateway 有三个组件：

- BaseApp - 它提供样板代码以通过 yaml

配置添加/删除可插拔组件，并具有内置的指标注册表模块（metrics registry module），可以轻松地为基于此构建的应用程序发出自定义指标。

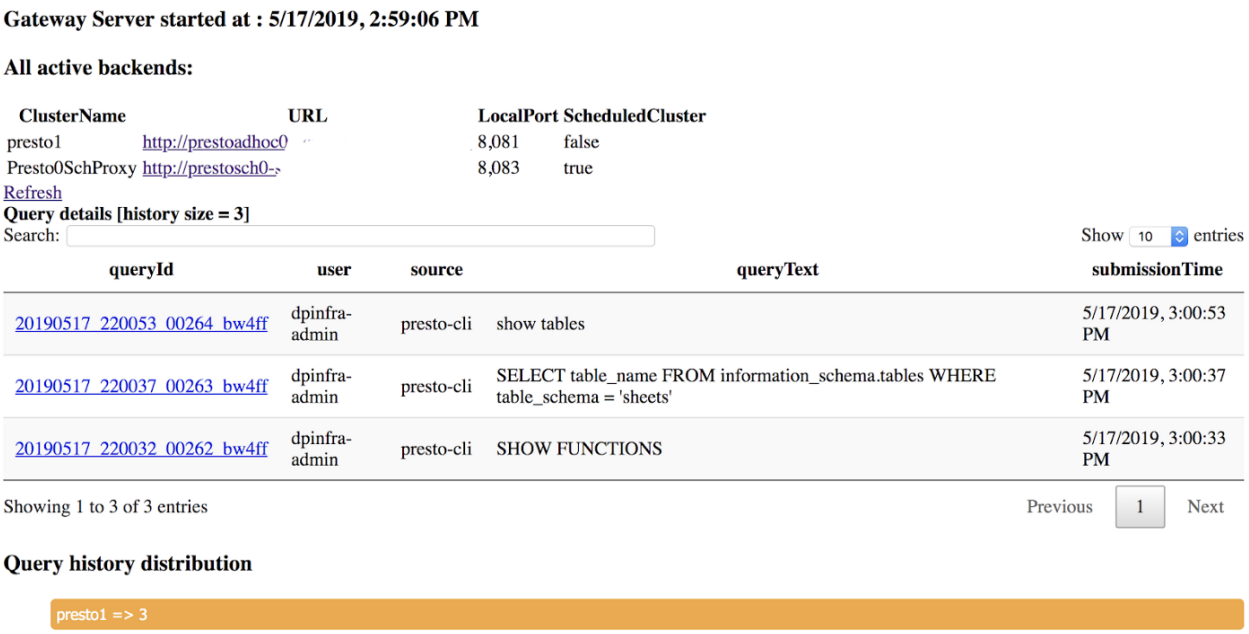


- ProxyServer——它是一个建立在 jetty 代理之上的库，它提供了一个带有可插拔 proxy-handler 的代理服务器实现。



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

- Gateway ——该组件充当代理服务器的容器，并插入 ProxyHanders 以提供代理、路由和负载平衡功能。 它还公开了一些 endpoint 和 UI 来激活/停用最近提交的查询的后端和查询历史作业的功能。

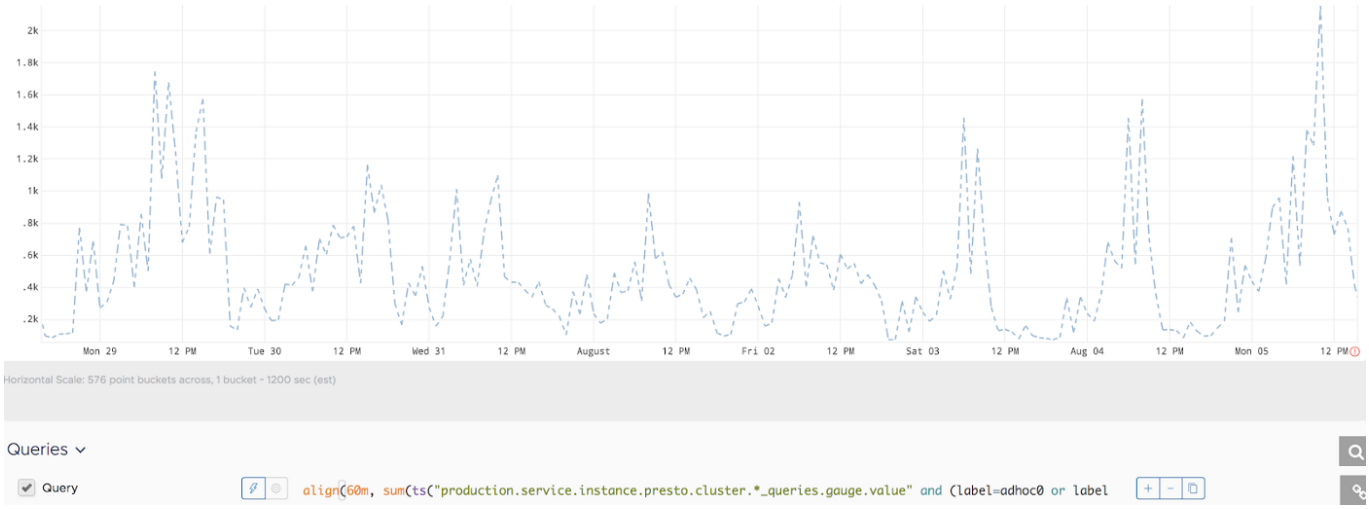


如果想及
时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

我们正在使用 lombok 来减少大量样板代码，例如 getters/setters/logger/equals 等，以加快开发过程，并且该应用程序是使用 dropwizard 框架构建的。

成本意识扩展以满足查询需求

我们从一个 Presto 集群开始，随着使用量的增长，我们不断添加更多的 worker 节点来支持更高的计算需求。随着我们添加更多 worker 节点，为了充分利用集群的潜力，必须提高查询并发设置，并且需要重新启动 presto-coordinator 导致停机时间。 引入 presto-gateway 后，我们采用 gateway 作为负载均衡器的多集群模式。整个 presto 基础设施的使用在一天中并不统一，而且我们的工作负载的性质是突发的，所以我们配置了很多基础设施，如果有大量的查询，也不至于没资源来处理。



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

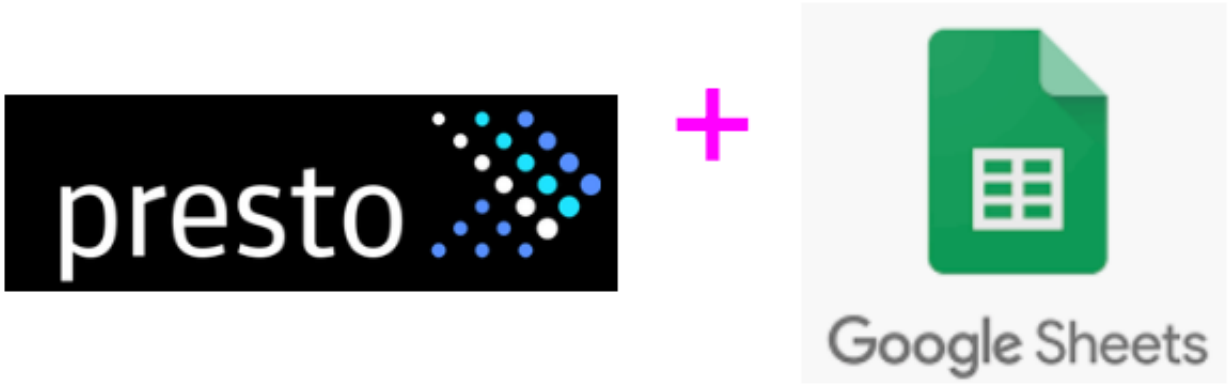
为了优化成本，我们实现了集群的动态扩缩容功能。
我们查看了查询请求数，并注意到在工作时间的使用率通常更高。在 gateway
实现后端激活/停用 API 后，我们已经能够执行无停机升级、部署和维护。
我们将此提升到了一个新的水平，并在非工作时间为一半数量的集群添加了计划停机时间。
在触发关闭前 30 分钟，我们使用 gateway API 禁用集群，因为我们为任何查询设置了 30 分钟的最大运行时间，以确保在此过程中没有查询受到影响。下图显示了节点数量如何随时间变化：

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

在非工作时间削减 50% 的基础设施，总体上节省了 30% 的成本。

Google sheets connector 插件

这个数据源允许 Presto 从 Google sheets 中读取数据，以便可以将小维度数据添加到查询中。



我们在 Presto summit 2019 中宣布了此功能，并将其贡献给社区。

限制

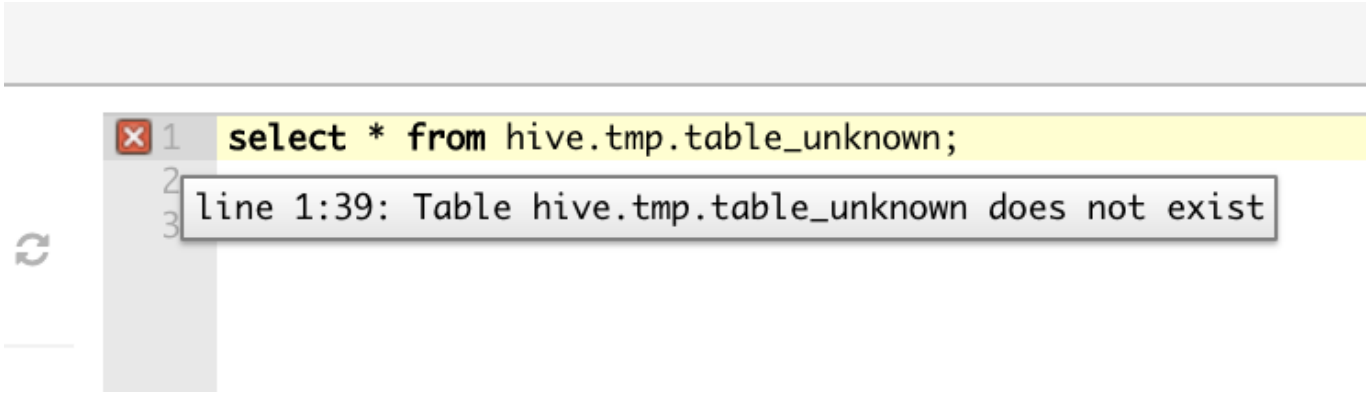
目前 Google sheets 连接器插件有以下限制。

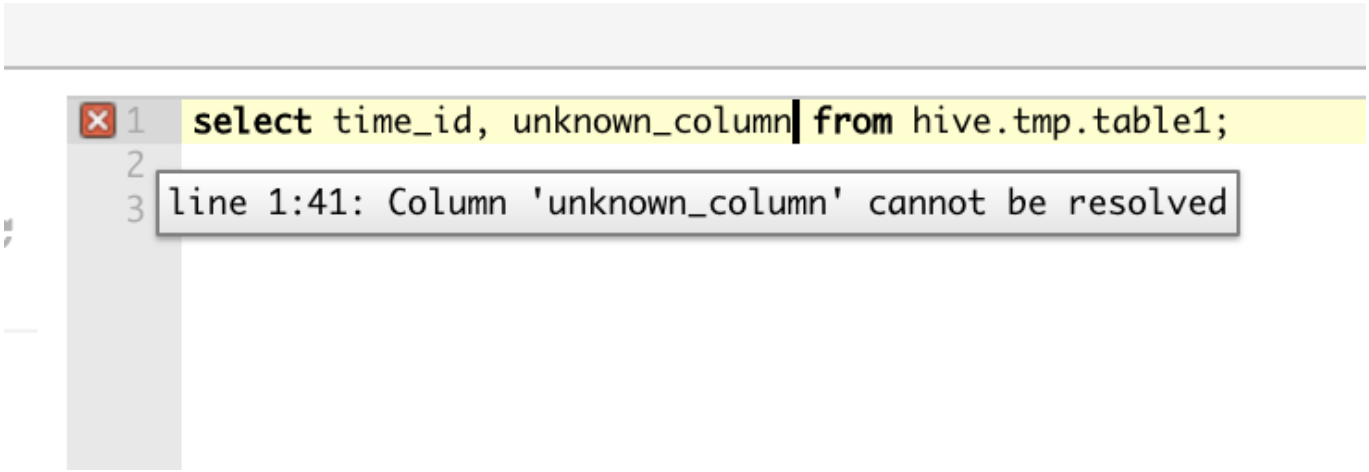
- 所有 sheets 都必须与服务帐户用户共享，至少具有查看权限。
- sheets 的第一行始终被视为所有列的列名；
- 所有列都使用 VARCHAR 类型解析；
- Google sheets API 的速率限制 - 如果 google
  项目帐户未启用计费，则调用速度每100秒进行100个调用（一天无限制）。
  为了避免这种情况，用户可以为缓存配置属性选择更高的值——在 presto-gsheets
  数据源中配置 sheets-data-expire-after-write 属性。

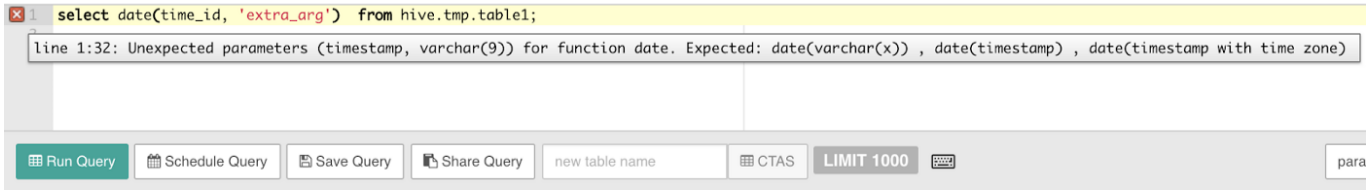
更多关于 Google Sheets connector 的介绍，请参见
https://trino.io/docs/current/connector/googlesheets.html。

Superset Presto 集成改进

用户在开发基于 SQL 的工作流或管道时面临许多挑战。
查询浏览器提供错误和建议以在执行查询后修复语法错误（错误的 SQL 或错误的 UDF 使用）。
用户需要花费一些时间通过多次迭代来找出和解决此类错误，并降低用户体验。 我们在 Presto
中实现了 Explain Type (Validate)
查询，并在实际查询执行之前将这些解释查询作为用户类型发送到 Superset 中的 sqlLab
中，返回的解释计划捕获语法错误以及列、表和 UDF
签名的有效性。这可以在不实际执行整个查询的情况下执行深度查询验证，通过消除编写复杂
SQL 查询所涉及的调试时间来改善整体查询体验。







Apache Superset — 执行前深度查询验证。我们已将这些功能贡献回开源。

总结

Lyft 的所有团队都必须做出数据驱动的决策，数据平台团队的使命是让数据成为 Lyft 所有决策的核心。 Presto 是我们实现这一使命的关键组成部分。 在过去的几年里，我们主要专注于扩展基础设施、减少数据到达延迟、改善用户查询体验，同时提高成本效益。

非常感谢 Data Platform Infra 团队的其他成员，他们帮助改进、扩展和支持 Lyft 的 Presto 基础设施。

另外，感谢 PrestoSQL 开源社区的其他成员在开发过程中帮助和支持我们。

本文翻译自：Presto Infrastructure at Lyft