Cilantro: Performance-Aware Resource Allocation for General Objectives via Online Feedback

研究背景

用于在竞争作业中分配有限资源的传统系统有以下两种方式，但忽略了作业运行的真实性能

- 基于用户指定的资源需求下确保公平性
- 通过一些替代的指标如CPU利用率来估计这些需求

作者认为当前的资源分配系统应该直接考虑作业的真实性能和用户的不同分配目标。基于此提出了Cilantro [sɪˈlæn.trəʊ]。

用户真正关心的是：影响业务的真实性能指标（real-world metrics）：服务工作的 P99 延迟或吞吐量）。除了最近的一些例外情况[10,18,35,64]，资源分配系统传统上关注的是作业请求的资源，而不是作业使用这些资源的实际性能（以下简称性能）。

service level objective (SLO)：服务性能指标

文中举了一个集群调度的例子，若采样基于CPU的分配算法会导致其中一个用户的SLQ，若已知用户的资源吞吐量曲线则可以同时使得两个用的的SLQ均得到满足。

研究问题

性能感知的资源调度策略的难点

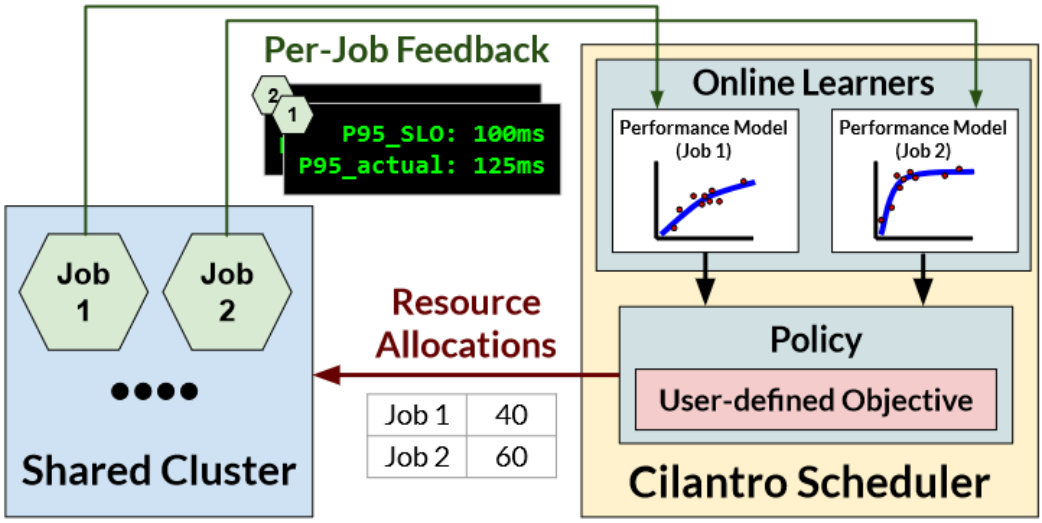
1. **从资源到性能的映射在实践中可靠性低**

- 离线分析的资源到性能映射可能无法可靠地反映生产环境中作业的性能，因为它可能无法捕获来自其他作业的干扰和服务器的性能变化
- 由于负载变化（例如，外部查询的到达率），作业的资源需求随着时间的推移而变化，而profiling通常无法解释这些变化
- 从资源到性能的映射分析对于用户来说是负担，对于组织来说是昂贵的，因为它需要大量资源来详尽地分析广泛的资源分配

2. **希望可以支持不同用户定义的不同调度目标**，文中举了两个例子

- 最大化集群的吞吐量而不是满足每个用户的SLO
- 向微服务提供资源时对端到端的性能目标更感兴趣，如应用程序延迟

基于此，作者提出Cilantro，在竞争作业中对单一可替代资源类型（例如 CPU、容器）进行性能感知分配的框架



在 Cilantro 中，终端用户首先声明调度目标。然后 Cilantro 通过以下措施克服上述提到的两个难点

- 为了实现从资源到性能的映射，Cilantro 使用一组**性能学习者和负载预测者分析来自作业的实时反馈并学习模型以估计每个作业的资源性能曲线和负载变化**
- 为了支持不同用户定义的不同调度目标，Cilantro 的调度策略是根据用户的目标自动得出的，利用这些第一步得到的模型来**计算每个作业的分配。**

随着学习的模型随着时间的推移变得准确，Cilantro 最终能够实现用户的目标，这就消除了估计给定性能目标所需的资源分配的离线模型的需求，并允许Cilantro优化自定义目标，如各种公平性或性能准则。这与忽视性能的策略截然不同，后者基于不可靠的代理度量指标，如CPU利用率和队列长度，以及其他基于启发式的策略（使用代理或性能度量），这些策略都是为了非常特定的调度目标而设计的。

Cilantro在两个场景下实现了面向性能感知策略的实施

- 独立作业的多租户资源分配
- 应用程序内部互相关联的作业（微服务）的资源分配

Cilantro的一些细节：

- 在数据不足的早期阶段，在线估计资源到性能的映射可能非常困难。为了在没有准确估计的情况下进行操作，Cilantro 会通过**其估计的置信区间**来通知调度策略
- Cilantro 通过**将学习机制与分配策略分离来实现支持自定义目标的通用性。**这种解耦使得能够单独考虑每个作业的性能和负载变化对目标的影响。

作者将 Cilantro 实现了 **Kubernetes 核心调度程序的开源扩展**， https://github.com/romilbhardwaj/cilantro 。

相关研究

忽略性能的方法

- 资源公平原则，最简单
- 根据策略进行调度，如 Kubernetes 、Mesos 和 YARN ，需要用户估计其自身工作的资源需求，这可能很困难

基于代理指标的方法

- 依赖于代理指标（例如 CPU 利用率、工作队列长度），最大化集群利用率，没有直接考虑用户的性能目标并针对单一目标进行优化

基于离线分析的方法

- 离线分析的资源到性能映射可能无法可靠地反映生产环境中作业的性能，因为它可能无法捕获来自其他作业的干扰和服务器的性能变化

基于在线反馈的方法

- 使用在线反馈的方法：在相关工作中，一些反馈驱动的系统考虑了资源分配中的性能指标和SLO。Jockey [18] 专注于通过对内部作业依赖关系进行建模来动态重新配置资源来满足单个作业的延迟 SLO。 Henge [35] 为流处理工作负载定义了新的效用函数，旨在最大化所有作业效用总和的单一目标。 [48]使用应用程序提示来预取操作系统内核中的磁盘块。 Gavel [44] 是一个调度程序，用于在具有不同目标的异构环境中进行 ML 训练工作负载。由于 Gavel 专注于 ML 训练，因此它的策略是针对吞吐量而设计的，并且贪婪优化器会计算每轮的最佳分配。最后，在视频流应用中，Minerva [45] 研究了资源分配方法，以便所有最终用户拥有相同的服务质量。
- **上述工作中使用的高度定制的策略虽然足以满足作者提出的分配目标，但不适用于我们的目标不同的集群目标。**Cilantro 支持用户指定的任何指标，并采用在线学习最终收敛到最佳分配。

可变的资源量

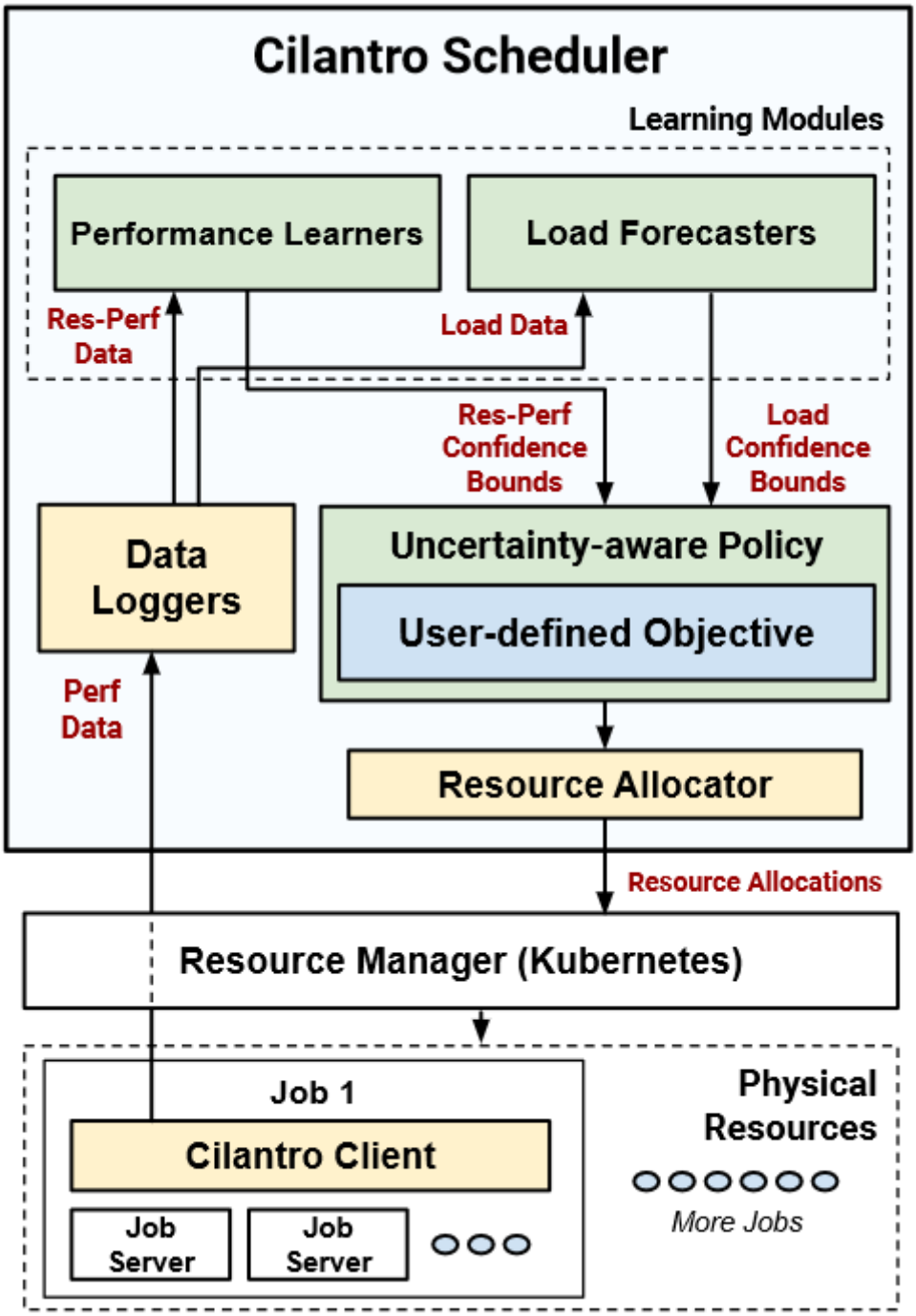
- 在其他相关工作中，PARTIES [10] 将资源分配给同一服务器内的作业，同时始终满足 SLO。如果无法满足所有作业的 SLO，则会将其中一个作业驱逐到另一台服务器；这种情况不适用于 Cilantro 的环境，**因为其考虑的资源数量是固定的，而且不可能驱逐作业。**
- Sinan [64]、DS2 [34]、Autopilot [51] 和 FIRM [49] **在资源可用性存在弹性时考虑使用在线反馈进行性能感知资源分配**，例如云端。由于这些工作可以扩展到比最初配置的资源更多的资源，因此它们不能直接与在固定集群设置中运行的 Cilantro 相比较。虽然云是一个新兴的用例，但出于隐私和成本原因，**传统的固定资源集群管理仍然具有相关性。**此外，上述工作侧重于具体目标，并非旨在处理一般分配目标。例如，FIRM [49] 专注于为部署为微服务的单个应用程序自动扩展资源，以最大程度地减少端到端 SLO 违规
- Cilantro 的操作方式不同，**根据用户指定的目标重新分配固定数量的资源**，其中可以包括公平性考虑。

- 云端资源可用性：如虚拟机、存储空间和网络带宽，而无需直接拥有或管理物理设备

研究内容

Cilantro 考虑的是一种传统的固定资源集群管理下的资源分配调度

系统框架



Cilantro 是一种**性能感知调度框架**，可以针对各种调度目标进行优化，而不需要任何工作负载的资源性能映射的先验知识。

Cilantro 的设计基于以下两个关键见解

- **资源性能的离线分析还不够**，需要通过在线学习以适应如服务器、应用程序性能的变化
- **学习机制和策略的解耦可以实现不同的调度目标。**由于不同的调度策略优化不同的标准，因此调度框架普遍支持不同的策略类型可能具有挑战性。先前关于反馈驱动的资源分配的工作[34,49,64]使用**端到端模型**来为固定目标（例如总效用或成本）分配资源。针对这些系统中的不同目标进行优化可能需要对系统和策略进行彻底的重新设计，或者至少需要对其模型进行昂贵的重新训练。**将学习机制与策略分离允许模型学习一次并应用于多个分配目标。**这种解耦还提高了调度程序做出的分配决策的透明度并有利于调试。

Cilantro 由两个关键组件组成

- 集中式 Cilantro 调度程序，负责生成资源分配，采用在线学习来创建越来越准确的作业绩效和负载模型，通过轮询这些模型的资源性能估计来优化用户定义的目标

- Cilantro 客户端，与每个作业位于同一位置的轻量级 sidecar，获取作业的性能指标并将其发送到 Cilantro 调度程序

Cilantro 调度程序调度器的模块，

- 数据记录器。从 Cilantro 客户端推送的**应用程序指标**存储在内存支持的表中。他们将这些指标**转发给性能学习器和负载预测者。**
- **性能学习器。性能学习器根据资源分配和负载使用相关模型，学习作业性能与这些因素之间的关系。**它定期轮询数据记录器以获取新数据并更新模型。学习器的更新频率仅受模型更新速度的限制。每个应用程序维护一个性能学习器实例。
- 负载预测器。负载预测器通过数据记录器的轮询定期更新。
- 不确定性感知策略。该策略通过计算资源分配来优化用户指定的调度目标。在在线环境中，直接使用性能的估计可能会失败，因为它没有反映模型中的不确定性。因此，**Cilantro的策略利用这些估计的置信区间**，以一种有原则的方式考虑不确定性，从而在进行资源分配决策时进行调整。
- 资源分配器。资源分配器负责通过与底层集群管理器进行交互来执行资源分配。该模块通过分配到期事件来驱动，一旦发生该事件，它就调用策略的compute-alloc方法并分配资源。分配到期事件是基于超时引发的，从而导致新一轮的资源分配。**在实践中，分配周期的持续时间受环境的敏捷性限制。**由于扩展作业需要时间，在过于频繁地更改资源分配之前可能会导致作业频繁启停（在有机会利用新资源之前需要缩减）。

Cilantro 客户端：Cilantro 客户端是一个轻量级的 side-car 容器，其目的是轮询作业以获取其当前性能、对其进行处理并将其发布到调度程序的数据记录器。客户端的主要任务是**从分配的工作中提取指标。**

策略设计

共享集群中的资源分配

性能：资源/负载到性能的映射，将性能那个映射表述为接受资源分配 a_j 和负载 l_j 的函数，性能可能是在 100 毫秒内完成的查询的比例，负载可能是指查询的外部到达率。

需求：如果一项作业有明确定义的 SLO，我们将需求 d_j 定义为实现该 SLO 所需的最小资源量。需求取决于作业的性能映射曲线 p_j、SLO 和负载 l_j

效用/性价比 Utility：作业的效用 u_j 是由于其性能而产生的实际价值。通常情况下，u_j 是性能的非递减函数，我们可以将其写作 $u_j(a_j, l_j) = u_i'(p_j(a_j, l_j))$，其中$u_j'$是一个非递减函数。

Cilantro 的在线学习策略

- 基于福利的在线策略
  - 最大化平均效用
  - 最大化最小效用
- 基于需求的在线策略：使用性能学习器和负载预测器的置信区间，得到对作业j需求的保守建议d(r)j。然后，通过调用相同的基于需求的策略，并使用推荐需求{d(r)1, ..., d(r)n}而不是真实需求，计算本轮的资源分配a(r)
  - Cilantro-NJC

微服务资源分配

希望优化由多个相互依赖的微服务（作业）组成的应用程序的端到端性能指标 p。

Cilantro中的微服务资源分配是指在一个由多个相互依赖的微服务（作业）组成的应用中，优化端到端性能度量p的使用场景。整个固定的资源集可供应用程序使用，并需要在微服务之间进行资源分配，以最大化性能度量p。

与多租户设置相比，这种设置引入了新的挑战，因为应用程序中的微服务可能具有复杂的依赖关系图，并且它们的性能可能是相互依赖的。此外，虽然应用程序的性能明显与单个微服务的性能相关，但无法将其明确地写成资源分配的函数，这与多租户设置不同。

为了解决这些挑战，Cilantro 将端到端性能 p 建模为每个微服务分配和应用程序面临的外部负载的直接函数,采用 OFU 原理，选择一个分配向量，该向量最大化从性能学习器获得的性能的置信上限：

$$a^{(r)} = \underset{a \in \mathcal{A}^{(r)}}{\operatorname{argmax}} \, \widehat{p}(a, \ell).$$

Cilantro 不足

- Cilantro目前仅支持分配单一资源类型。虽然可以将多个资源类型捆绑成组合单元，例如具有固定CPU、内存和GPU比例的虚拟机SKU，然后由Cilantro进行调度，但这种捆绑并不总是可行的，特别是当不同的作业具有不同的资源需求时。
- Cilantro无法处理非可互换的资源类型。
- 尽管Cilantro的不确定性感知设计在面对不可预测的变化时具有一定的弹性，但持续的极端负载波动可能会对Cilantro的性能产生负面影响。为了避免在重新分配资源时出现迟滞，未来的工作可以探索在动态大小的窗口上对负载进行平均或包含规则以临时覆盖Cilantro的策略。
- Cilantro在多租户设置中也不支持基于市场的资源分配策略的在线学习版本。