Boreas - A Service Scheduler for Optimal Kubernetes Deployment

李剑明

摘要

为了保持高效调度，Kubernetes 的默认调度器在设计之初优先考虑速度，而不是调度的最优性；而在系统动态变化不大且资源有限的场景中，更倾向于更精确和资源利用率更高的调度。实验表明，由于缺乏对调度的布局计算能力，默认调度器在某些资源足够的场景下无法完成对所有 Pod 的合理调度。Boreas 通过集成 Zephyrus2 这一配置优化器，将 Kubernetes 中各种资源的约束条件转化，提交至 Zephyrus2 得到最优化调度方案，然后通过 Kubernetes API 完成 Pod 的调度。而 Boreas 实现的只是在一个调度周期内的局部最优化，本选题通过引入二次调度，允许对已部署 Pod 进行驱逐，再捕获并重新调度由上层 ReplicaSet 创建的新 Pod，实现了全局的最优化调度。

关键词：Kubernetes；调度；最优化

1 引言

以 Docker 为代表的容器虚拟化技术实现了轻量化的环境部署，推动着云计算的发展，现如今许多组织都采用容器来部署应用。而 Kubernetes 作为容器编排领域的事实标准，能够管理运行在不同主机节点上的分布式容器，基于内置的策略将容器调度到合适的节点上运行，并且管理容器的生命周期。

调度器 (Scheduler) 是 Kubernetes 的工作组件，负责整个集群资源的调度。Kubernetes 默认使用的调度器能够确保在大多数场景下做出合理的调度，满足我们绝大多数的需求，但却不能覆盖所有的场景；考虑到实际环境的各种复杂情况，Kubernetes 的调度器采用插件化的形式实现，方便用户进行定制和二次开发。默认的调度器在设计之初优先考虑速度而不是调度的最优性，以便保持高效调度，而在系统动态变化不大且资源有限的场景中，更倾向于更精确和资源利用率更高的调度，在这种场景下，需要设计一种调度器，其能够根据收集到的资源利用情况做出最优化的部署，提高整个集群的资源利用率，实现更精确的调度。

本选题在最优化调度器 Boreas 的基础上进行。Boreas 通过集成 Zephyrus2 这一配置优化器，在每个调度周期收集 Kubernetes 中各种资源的约束条件，并将其转化为 Zephyrus2 可接受的不等式组合后，提交给 Zephyrus2，进而得到当前最优的调度方案。但 Boreas 只是在一个调度周期内的局部最优调度，本选题通过引入二次调度，允许对已部署容器进行驱逐，实现全局的最优化部署，在资源有限的集群环境中能够进一步提高资源的利用率。

2 相关工作

2.1 调度器实现方式

Kubernetes 中的组件都被实现为服务集合，每个服务都是单独开发和部署的，具备扩展性；其中调度器作为核心部件，默认的调度器 kube-scheduler 虽然能满足绝大多数需求，但在特定场景下需要

进行定制开发，以在某些方向上有所优化。Kubernetes 扩展调度器的方法主要有三种：调度器扩展程序、调度框架和 Kubernetes API。

调度器扩展程序本质是一个可配置的 Webhook，其中包含过滤器 (Filter) 和优先级 (Prioritize) 两个端点，分别对应调度周期中的过滤阶段和打分阶段。在具体实现过程中需要定义 Extender 策略文件，对于 Filter 需要实现对节点的过滤逻辑，判断节点是否可以进入下一阶段打分；对于 Prioritize 需要实现分数计算逻辑。调度器扩展程序在一些情况下可以实现对调度器的扩展，但由于扩展点有限，只能在 Filter 和 Prioritize 的末尾阶段发挥作用，局限性较大。

调度框架 (Scheduler Framework) 是官方推荐的主流扩展方式，其将调度的详细流程划分为调度过程和绑定过程两部分，并且在这两部分中设置多达 12 个可供注册的扩展点，以便可以执行更复杂的有状态任务。在调度框架下，要实现一个插件以完善调度器的功能，只需实现对应的扩展点，然后把插件注册到调度器中即可。

Kubernetes 提供多种语言的 API 以供调用，运行代码等价于用户通过 kubectl 手动管理集群，能够实现自动化管理的效果。其中，以 Go 语言呈现的 Client-Go 和以 Python 语言呈现的 kubernetes-client/python 是被应用最为广泛的 API 库，通过封装好的 API 程序员可以监控集群状态、收集信息并完成调度、驱逐等操作。基于此实现的调度器虽然在兼容度上没有调度框架高，但由于 API 的丰富性，所以可以实现更多样化的功能。

2.2 调度器优化方向

调度器是 Kubernetes 的核心组件，由于默认调度器在设计之初是为了在大部分应用场景上保持高效的调度，所以其通用性很强；但兼顾各个场景意味着要进行取舍，场景间各个特性的相互制约使得默认调度器可能无法适应特定的场合，为此需要对调度器进行定制化修改。Kubernetes 本身就提供多种方式自定义调度器，为的就是根据需求执行更好的调度策略。

近几年来，针对 Kubernetes 调度器相关的研究也在不断开展。在调度方面，部分研究人员旨在提高 Kubernetes 的调度性能。Tarek Menouer[1]基于 TOPSIS 多标准决策分析算法提出了 KCSS，在调度时同时考虑六个关键指标：(1) 节点 CPU 利用率；(2) 节点内存利用率；(3) 节点磁盘利用率；(4) 节点功耗；(5) 节点运行容器的数量；(6) 传输容器对应镜像所需的时间；将所有标准聚合到一个等级中，在云基础设施和用户需求相关的混合标准之间进行折衷，最终提升不同场景下的性能表现。Yuqi Fu[2]等人提出了一种基于进度的容器放置方案 ProCon，在调度时，ProCon 不仅考虑即时资源利用率，而且考虑了对未来资源使用情况的估计。实验表明 ProCon 可缩短特定作业的完成时间达 53.3% 并提高整体性能 23.0%，与 kube-scheduler 相比，ProCon 的完成时间提高了 37.4%。Ghofrane El Haj Ahmed[3]等人描述了一个动态调度平台 KubCG，该平台实施的调度程序通过考虑 Pod 的时间线和有关容器执行的历史信息来优化新容器的部署。在不同实验中，KubCG 能将不同任务的完成时间减少到原本的 64%。

除此之外，许多研究人员致力于推广 Kubernetes 到其它计算环境，根据特定环境对 Kubernetes 的调度做定制优化。Lukasz Wojciechowski[4]等人通过 Istio 服务网格收集动态的网络指标，提出了新的调度算法 NetMARKS，该算法能够将应用程序的响应时间减少 37%，并节省多达 50% 的节点间带宽，使得 Kubernetes 在 5G 应用场景中保持高吞吐量，尤其是多接入边缘计算和机器对机器通信场景。Yiwen Han[5]等人认为 Kubernetes 具有合并分布式边缘和云的潜力，但缺乏专门针对边缘云系统的调度框架，

为此设计了基于学习的调度框架 KaiS，它能作用于边缘云系统，提高请求处理的长期吞吐率。越来越多的开发者和企业选择将人工智能应用部署在 Kubernetes 集群上，但 Kubernetes 集群并不是主要针对深度学习而设计的，因此需要对深度学习这个特定的领域做定制优化。陈培[6]等人在优化调度器的过程中，将 CPU 的“本地的”和“非本地的”概念扩展到 GPU，在调度的打分阶段把 GPU 的连接情况纳入到计算范围中，使得调度策略满足深度学习场景。

Kubernetes 在调度的优选阶段只根据节点的 CPU 和内存利用率来决定节点的分值，这只能保证单节点的资源利用率，无法保证集群资源的负载均衡。为此，胡程鹏[7]等人提出了一种基于遗传算法的资源调度算法，在该算法中加入网络带宽和磁盘 IO 两项指标，并给指标赋予不同的权重，以及引入校验字典修复遗传算法生成过程中不符合配置的个体。实验表明，与 Kubernetes 默认调度策略相比，该算法能提高集群的负载均衡能力。相似的，T Lebesbye[8]等人通过引入配置优化器 Zephyrus2[9]，设计了 Kubernetes 最优化调度器 Boreas，通过收集待调度 Pod 的资源需求与集群空闲资源容量，转换约束请求后提交给 Zephyrus2 得到最优化部署方案，进而实现最优化调度，在特定场景下能完成 Kubernetes 默认调度器无法完成的调度。

3 本文方法

3.1 本文方法概述

本文在 Boreas 调度器的基础上进行改进，通过完善代码引入二次调度，使得 Boreas 的调度功能从原本的局部最优扩展到全局最优，进而实现在特定场景下对资源的充分利用，摆脱由于调度周期而存在的局限性。

3.2 Boreas 局部最优调度

Boreas 的结构如图 1所示：

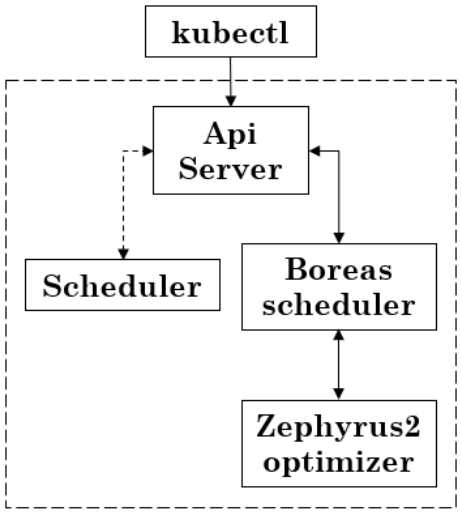

图 1: Boreas 结构图

在 T Lebesbye 等人的工作中，整个调度器作为 Controller Plane 的一个 Pod 存在，其中包含两个容器：Boreas Scheduler 和 Zephyrus2，由于属于同一个 Pod 下的容器，双方可以直接通过 HTTP 交流。Boreas Scheduler 基于 Kubernetes 提供的 API 监控集群的状况，在每个调度周期中，收集一批待调度的 Pod，将这些 Pod 所需要的 CPU 和内存资源转换为约束条件，连同各个节点此刻的可用资源量一起，提交给配置优化器 Zephyrus2。

Zephyrus2 是一种配置优化器，它通过 SMT 或 CP 求解器来探索搜索空间，进而计算应用程序的最佳配置以部署在云或其它集群上。Zephyrus2 最早被应用来计算应用程序在虚拟机上的最佳位置，而在 Boreas 中，将各个节点抽象为位置 (Location)、各个待部署的 Pod 就是需要放置的应用程序，Zephyrus2 将自动找到一组位置上分布组件的配置，使得：(1) 满足用户需求的约束；(2) 提供已部署组件的所有功能；(3) 各个位置上的可用资源足以覆盖在其上部署的所有组件的资源需求；(4) 最小化用户自定义的目标函数值；默认情况下的目标函数是以较低的成本获取最终配置，在“平局”的情况下选择组件数量最少的配置。

Boreas Scheduler 将集群状况转换为 Zephyrus2 的语言后，交由 Zephyrus2 计算得到最优化部署方案，基于此方案通过 Kubernetes API 实现调度，最终完成最优化调度。在特定场景中，可能会出现 Kubernetes 默认调度器无法完成所有 Pod 的调度、但 Boreas 可以完成的情况。

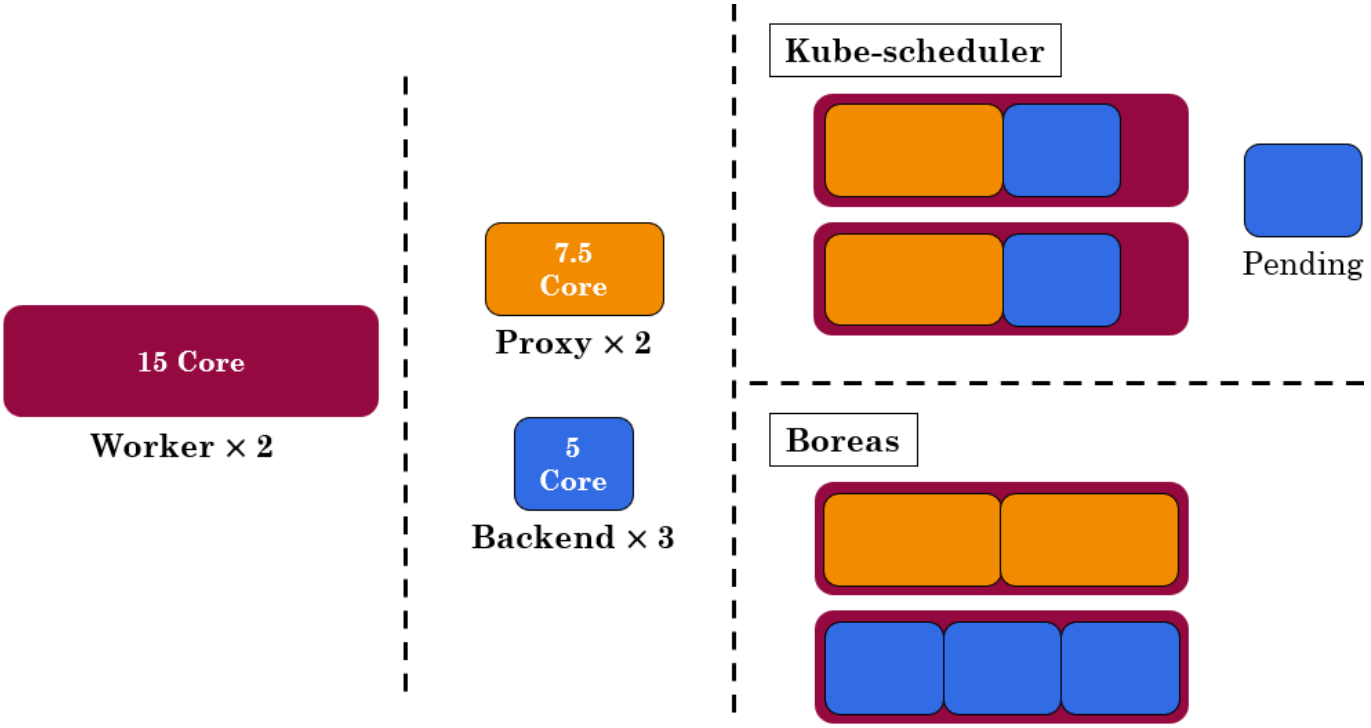

图 2: Boreas 与 Kubernetes 默认调度器的实验对比

如图 2所示，存在两个含有 15 核 CPU 资源的工作节点，待调度的 Pod 有：两个需要 7.5 核 CPU 的 Proxy 应用、三个需要 5 核 CPU 的 Backend 应用。经过实际测试，Kubernetes 由于将 Proxy 应用和 Backend 应用调度到同一节点，导致每个节点剩余 2.5 核 CPU 资源无法被利用，而一个 Proxy 应用处于资源不足无法被调度的 Pending 状态，最终呈现为非最优化调度。而 Boreas 由于经由 Zephyrus2 计算后得到最优化调度方案，因此成功将所有 Pod 都调度到节点上运行。

3.3 Boreas 全局最优调度

Boreas 虽然是最优调度，但却只是局部最优调度。究其原因是 Boreas 存在“调度周期”的限制，在实际代码实现中，Boreas 会每隔 30 秒提交一次期间收集的待调度 Pod 给 Zephyrus2，如果在这 30 秒内提前收集到 99 个 Pod 的数量上限则会提前提交，因此，Boreas 实际是基于每个周期收集到的集群状态信息做出的局部最优调度。

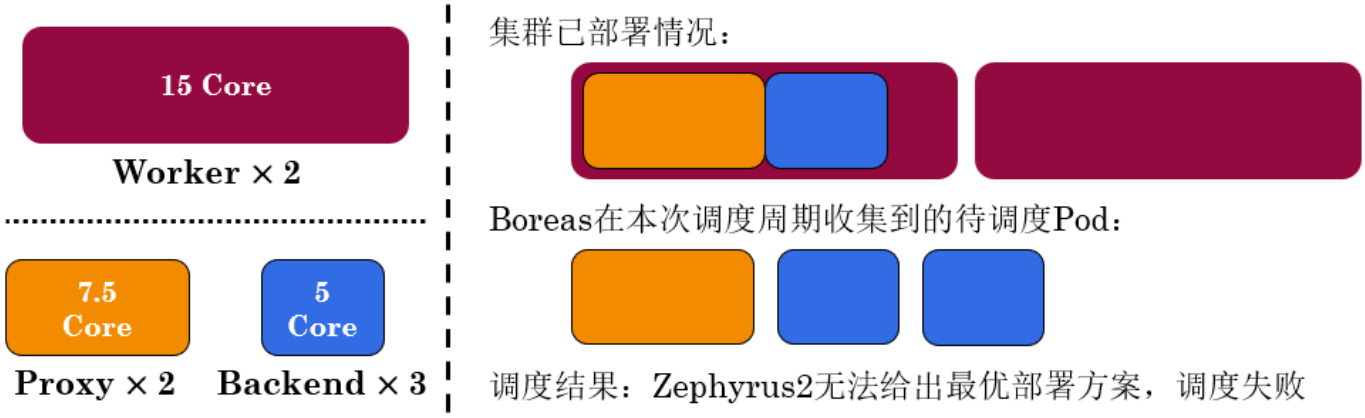

图 3: 实验证明 Boreas 存在的缺陷

如图 3所示，在相同的环境中进行实验，存在两个含有 15 核 CPU 资源的工作节点，待调度的 Pod 有：两个需要 7.5 核 CPU 的 Proxy 应用、三个需要 5 核 CPU 的 Backend 应用。在上一个调度周期中，Boreas 收集到需要调度的有一个 Proxy 应用和一个 Backend 应用，由于 Zephyrus2 会最小化用户的自定义目标函数，而默认情况下的目标函数是以较低成本获取最终配置，因此这两个应用都会被调度到同一个节点上。在下一个调度周期，Boreas 再次收集到需要调度的 Pod 有：一个 Proxy 应用和两个 Backend 应用。由于此刻不存在最优化调度方案，因此会使得 Boreas 无法调度这三个 Pod。

对 Boreas 的改进方向是令其支持二次调度。在 Kubernetes 中，Pod 被分为两类：一类是直接通过 Pod 配置文件创建的普通 Pod；一类是通过 ReplicaSet 间接创建、能够进行生命周期管理的特殊 Pod。Pod 在其生命周期中可能会出现异常而导致运行结束，普通 Pod 一旦结束就只能手动恢复，但特殊 Pod 的状态会被 ReplicaSet 持续监测，一旦因为异常而结束，ReplicaSet 就会删除该 Pod 并立刻创建新的 Pod，以恢复该 Pod 提供的服务。二次调度首先需要将已在节点上运行的 Pod 驱逐，随后捕获到上层 ReplicaSet 新创建的 Pod，再将该 Pod 纳入到调度规划中。为此，改进后的 Boreas 只适合对由 ReplicaSet 创建的特殊 Pod 进行全局最优调度；事实上，Kubernetes 本身就推荐通过 Deployment、DaemonSet 等方式间接创建 Pod，通过创建一个上层的 Pod 控制器管理其生命周期，保证服务的高可用性。

秉承 Kubernetes 组件插件化的设计方式，对 Boreas 的改进方式是：如果首次调度失败后，才转入二次调度。二次调度的流程为：

1. 通过 Kubernetes API 获取节点的资源容量。
2. 转换约束条件后，提交至 Zephyrus2。
3. 对已部署的 Pod 进行驱逐。
4. 基于上层 ReplicaSet，捕获新创建的 Pod。
5. 延时部署，避免驱逐操作未完成导致调度失败。

改进后的 Boreas 调度算法对应的流程图如图 4所示。

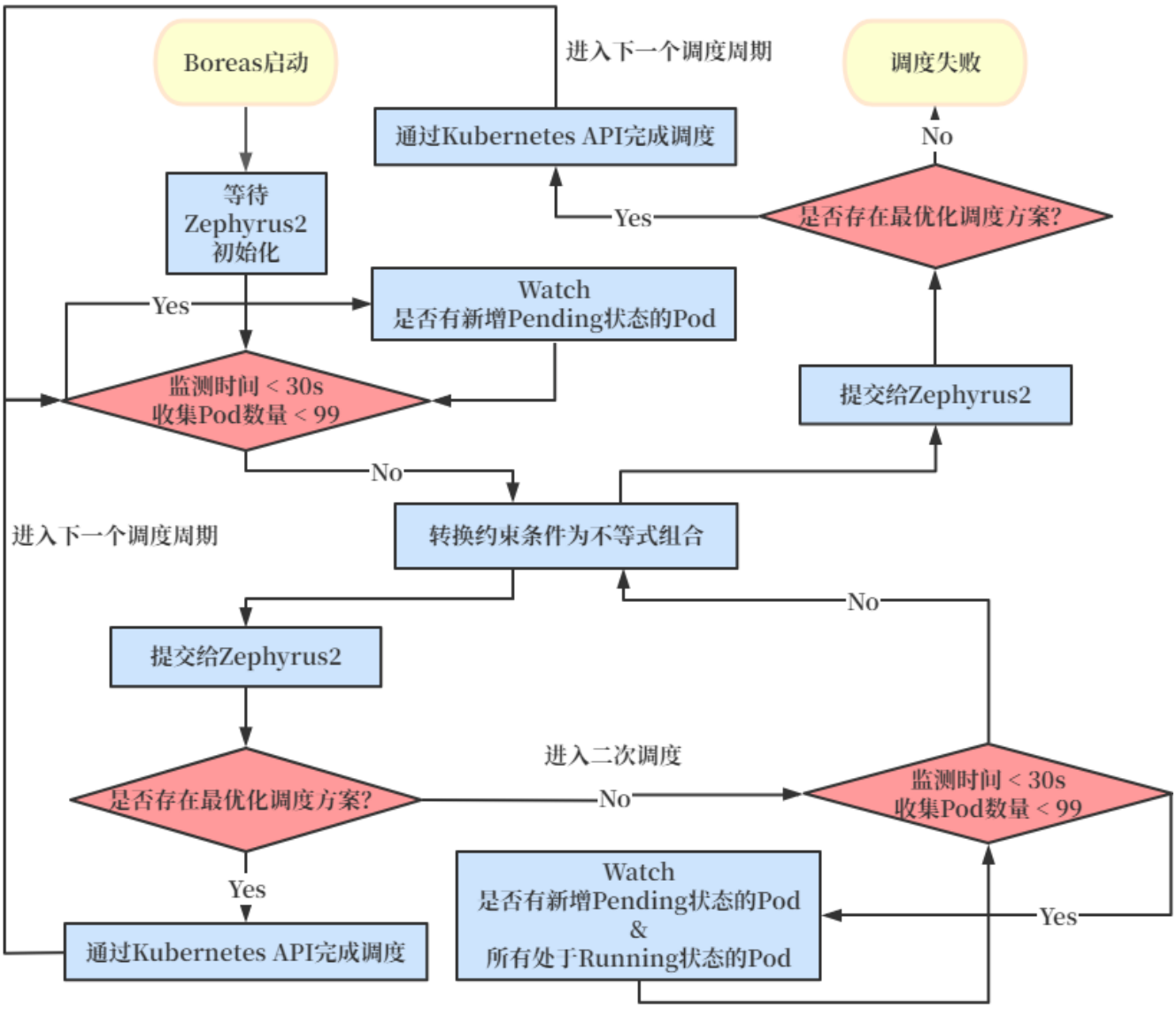

图 4: 改进后的 Boreas 调度算法流程

4 复现细节

4.1 与已有开源代码对比

原有 Boreas 调度器开源在 Github：https://github.com/torgeirl/boreas-scheduler，使用 Python 编写，通过 API 对 Kubernetes 集群进行操作。主要的代码逻辑实现在 health_server.py 和 scheduler.py 中，其中 health_server.py 创建一个简单的 HTTP 服务器，请求 /health 路径时会返回状态码 200，该服务器主要提供给 Kubernetes 检测服务是否正常运行；Boreas 调度器则实现在 scheduler.py 文件中。

scheduler.py 文件中，定义了许多 API 函数供直接使用，例如：nodes_usage() 能够获取节点已使用资源、nodes_available() 能够获取节点剩余资源、schedule() 能够根据参数将 Pod 调度到指定的节点上。其代码逻辑为：

1. 通过 wait_for_optimizer() 函数等待 Zephyrus2 的启动。Boreas Scheduler 是跟 Zephyrus2 同时启动的，但却要依赖 Zephyrus2 的功能，所以在函数中定义了 wait_for_optimizer() 函数探测 Zephyrus2 的运行状况，只有当 Zephyrus2 成功运行后，Boreas Scheduler 才会继续
2. 在每个调度周期通过 get_event_batch() 函数获取下一批需要调度的 Pod。调度周期默认设置为 30 秒，期间如果提前收集到指定数量上限的 Pod 则会提前结束收集。

3. 分析收集到的待调度 Pod，将其所需的资源通过 get_labels_and_sets() 转换为约束条件；并且一同分析集群中的节点状态信息

4. 在 schedule_events() 函数中将收集并转换后的数据通过 optimize() 提交给 Zephyrus2，根据其返回的最优化部署方案，对待调度 Pod 依次进行调度

由于改进 Boreas 的代码逻辑是：在首次调度失败后，再转入二次调度；因此在原有的首次调度函数 schedule_events() 中添加 Flag。新增 nodes_capacity() 函数计算节点资源容量、新增 reschedule_events() 函数将集群状态提交给 Zephyrus2。如果二次调度过程中 Zephyrus2 给出了新的最优化部署方案，则依据此方案进行调度：

- 对于待调度的 Pod，将其添加进 delay_list 队列中，因为这时可能存在需要驱逐的 Pod，为了避免资源占用冲突需要延时调度。
- 对于正在运行的 Pod，如果所在节点与本次调度方案结果相同，则跳过处理；如果不相同，则进行驱逐。

驱逐操作通过新添加的 evict() 函数实现，在该函数中，为了捕获驱逐后上层 ReplicaSet 创建的新 Pod，需要先记录 Pod 对应 ReplicaSet 及其管理的所有 Pod，随后通过 API 对该 Pod 进行驱逐。持续对 ReplicaSet 管理的 Pod 进行监测，当且仅当原有 Pod 被删除且新 Pod 被创建，才将新 Pod 的名字返回。

4.2 实验环境搭建

首先需要搭建 Kubernetes 集群，且除了控制节点外还需要至少两个工作节点，后续的调度实验在工作节点上进行。可以使用 minikube 进行测试环境的搭建，在启动 minikube 时通过 –nodes 参数指定需要创建的工作节点数量。

改进后的代码同样打包成镜像，已上传到 Docker Hub，通过 Docker 进行拉取后即可作为 Kubernetes 集群中的一个 Pod 进行应用。在搭建实验环境时，用户只需通过 kubectl 应用 scheduler-improve.yaml 配置文件，Kubernetes 会自动拉取改进后的 Boreas Scheduler 和 Zephyrus2 镜像。拉取到本地后，两者作为一个 Pod 下的两个容器运行，当 Boreas 对应的 Pod 处于 Running 状态即表示实验环境搭建成功。

4.3 使用说明

- 应用 scheduler.yaml 创建原始 Boreas 调度器，该调度器只实现了调度周期内的局部最优调度；应用 scheduler-improve.yaml 创建改进后的 Boreas 调度器，该调度器实现了全局的最优调度。
- 分别应用 myTest1-default.yaml 和 myTest1-boreas.yaml 说明原始的 Boreas 调度器可以解决 Kubernetes 默认调度器无法完成调度的情况。
- 依次应用 myTest2-1.yaml 和 myTest2-2.yaml 说明改进后的 Boreas 调度器跳出了调度周期的限制，相比原始的 Boreas 调度器可以实现全局最优调度。

5 实验结果分析

存在两个含有 15 核 CPU 资源的工作节点，待调度的 Pod 有：两个需要 7.5 核 CPU 的 Proxy 应用、三个需要 5 核 CPU 的 Backend 应用。在上一个调度周期中，Boreas 收集到需要调度的 Pod 有：一个 Proxy 应用和一个 Backend 应用，Zephyrus2 基于最小化目标函数的优化思想，会将这两个应用调度到同一个工作节点上：

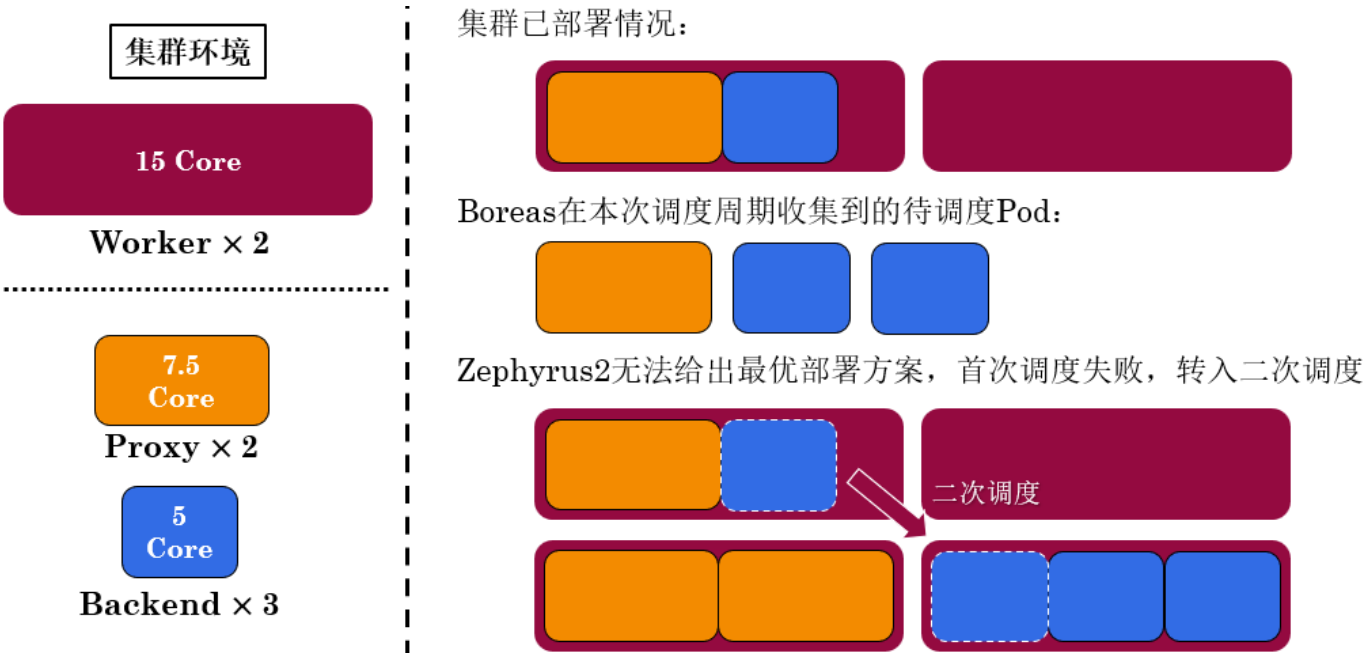

图 5: 实验证明改进后 Boreas 的全局最优调度能力

而在下一个调度周期中，Boreas 收集到待调度的 Pod 有：一个 Proxy 应用和两个 Backend 应用，由于此时节点上的剩余资源无法满足所有 Pod 的需求，因此 Zephyrus2 无法给出最优化部署方案，首次调度失败。随后转入二次调度，通过分析，将原本已调度到工作节点上的 Backend 应用进行驱逐再调度，可以满足所有 Pod 的资源需求，因此二次调度成功。工作节点上的 Backend 应用被驱逐后，经由上层 ReplicaSet 新建，随后加入到待调度的队列中；工作节点上的 Proxy 应用由于与 Zephyrus2 给出的最优化调度方案相同，因此不作任何操作；最后实现将所有 Proxy 应用都调度到一个节点上、所有 Backend 应用都调度到另一个节点上，完成全局最优化调度。

通过实验说明了 Boreas 调度器的局限性以及改进后的效果。

5.1 创新点

本选题完善了 Boreas 实现源码，通过对已运行 Pod 的驱逐再调度，引入二次调度功能，使得原本只是在调度周期内实现局部最优调度的调度器变成全局最优调度，更能适应特定应用场景的调度需求。改进后的代码已打包成镜像上传到 Docker Hub，并且源码上传到 Github，直接拉取镜像即可部署在 Kubernetes 上发挥作用。

6 总结与展望

Boreas 本身是提出了一种实现最优化调度的方案，本选题在其基础上对该方案进行改进。原本的 Boreas 由于存在调度周期，其所实现的最优化调度是局限在每个调度周期内的局部最优调度，虽然能解决 Kubernetes 默认调度器无法处理的调度场景，但仍存在特定场景下无法解决的情况。通过对已调度的 Pod 进行驱逐、捕获由上层 ReplicaSet 新创建的 Pod 并将其加入到等待调度的队列中，实现二次调度功能。添加二次调度功能后的 Boreas 能够实现全局最优化调度，摆脱调度周期的约束，适应更多应用场景，更充分地发挥 Zephyrus2 的最优解搜索能力。

当然，本选题也存在一些不足的地方，例如并未在更多复杂场景下进行实验，实验数据不够充分。并且容器最优化调度本身就是一个热点的研究方向，Boreas 通过 Zephyrus2 计算得到最优化调度方案，在未来的研究中可以测试其它求解器对 Kubernetes 调度问题的兼容性，进行横向对比。此外，Boreas

源码使用 Python 编写，借助的是 Kubernetes 自身提供的 Python 代码 API，但 Kubernetes 的调度器通常是使用 Go 进行编写，借助官方推荐的调度框架以更好地适应整个 Kubernetes 框架，未来可以将其用 Go 语言重写。

参考文献

[1] MENOUER T. KCSS: Kubernetes container scheduling strategy[J]. The Journal of Supercomputing, 2021, 77(5): 4267-4293.

[2] FU Y, ZHANG S, TERRERO J, et al. Progress-based container scheduling for short-lived applications in a kubernetes cluster[C]//2019 IEEE International Conference on Big Data (Big Data). 2019: 278-287.

[3] EL HAJ AHMED G, GIL-CASTIÑEIRA F, COSTA-MONTENEGRO E. KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters[J]. Software: Practice and Experience, 2021, 51(2): 213-234.

[4] WOJCIECHOWSKI Ł, OPASIAK K, LATUSEK J, et al. NetMARKS: Network metrics-AwaRe kubernetes scheduler powered by service mesh[C]//IEEE INFOCOM 2021-IEEE Conference on Computer Communications. 2021: 1-9.

[5] HAN Y, SHEN S, WANG X, et al. Tailored learning-based scheduling for kubernetes-oriented edge-cloud system[C]//IEEE INFOCOM 2021-IEEE Conference on Computer Communications. 2021: 1-10.

[6] 陈培, 王超, 段国栋, 等. Kubernetes 集群上深度学习负载优化[J]. 计算机系统应用, 2022, 31(9): 114-126.

[7] 胡程鹏, 薛涛. 基于遗传算法的 Kubernetes 资源调度算法[J]. 计算机系统应用, 2021, 30(9): 152-160.

[8] LEBESBYE T, MAURO J, TURIN G, et al. Boreas–A Service Scheduler for Optimal Kubernetes Deployment[C]//International Conference on Service-Oriented Computing. 2021: 221-237.

[9] ÁBRAHÁM E, CORZILIUS F, JOHNSEN E B, et al. Zephyrus2: on the fly deployment optimization using SMT and CP technologies[C]//International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. 2016: 229-245.