Kairos: Preemptive Data Center Scheduling Without Runtime Estimates

Kairos:抢占式数据中心调度，无需估计运行时间

优化目标：作业完成时间

基于事件event

ABSTRACT

绝大多数数据中心调度器使用任务运行时间估计来改进其调度决策的质量。运行时间允许调度器实现更好的负载平衡和避免排头阻塞(HOL)。

不容易获取准确的运行时间估计，错误的估计会导致次优调度决策。

减轻不准确估计的影响的技术已经取得了一些成功，但根本问题仍然存在。

Kairos：一个新的数据中心调度器

没有任务运行时间这一先验信息

Least Attained Service (LAS)调度策略

两级调度：

1. per-node：在节点本地实现LAS，抢占
2. centralized：分配任务到节点

实验：在YARN上实现Kairos，并和2个基于YARN的调度器进行比较（FIFO和Big-C）

仿真：在Eagle模拟器中实现，并与Eagle进行性能比较

Eagle：

把节点分为2个子集群，一个用做长作业，一个用做短作业，允许短任务机会性地使用长作业分区中的空闲节点。通过这种工作负载划分技术，Eagle完全避免了排头阻塞。

Eagle优先执行短作业，短作业是根据非抢占的SRPT分布近似值执行的。

Eagle使用任务运行时间估计将作业分为长作业和短作业，并实现SRPT策略。

1 INTRODUCTION

这些作业的数据并行性质加剧了调度的复杂性：作业由多个任务组成，只有当所有任务都完成时，作业才会完成。

运行时间估计的两种一般方法：①先前的运行②初步分析阶段

基于估计的方法的局限性

一种广泛使用的估计技术，使用平均任务执行时间作为作业中所有任务执行的预测（相当于是作业运行时间/任务数量）——可能导致较大的错误(> 100%)。

基于机器学习的方法也会出现明显的估计错误。

导致难以获得可靠的运行时间估计的因素：输入数据的变化、数据倾斜。

处理这些问题的技术，例如队列重新平衡或不确定的调度策略，在减轻错误估计的影响方面已经显示出了一些成功，但是它们并没有从根本上解决问题。

Kairos

不使用运行时间估计

LAS：抢占，已获得最少服务的任务优先。当任务的运行时间分布具有较高差异时，有较好的表现（重尾分布）。

主要的挑战是在数据中心环境中为LAS找到一个好的近似。朴素的实现将导致频繁的任务迁移，并伴随着相应的性能损失。

我们开发了一个两层调度器，per-node和centralized，它完全避免了任务迁移，但仍然提供了良好的性能。

centralized调度器处理了两个挑战：

1. 保证高资源利用率，在没有运行时间估计的情况下。调度器的目标是平衡每个节点的任务数量，并通过限制分配给工作节点的最大任务数量来减少节点之间可能出现的负载不均衡。
2. 保证LAS的分布近似维持LAS原有的性能优势。将任务分配给节点，使得分配给特定工作节点的任务运行时间分布具有很大的差异。

本文贡献

1. 在不使用任务运行时间估计的情况下，我们展示了良好的数据中心调度性能。
2. 我们提出了一个高效的分布式版本的LAS调度规则。
3. 我们在YARN中实现了这种分布近似的LAS，并通过测量和仿真将其性能与最先进的替代品进行了比较。

2 BACKGROUND

估计任务运行时间

作业中任务的运行时间估计值可以从相同任务的过去执行(如果有的话)、类似任务的过去执行中获得，也可以通过在线分析获得。任务持续时间的一种常见估计技术是对它之前的执行时间取平均值，更复杂的技术依赖于机器学习。

挑战（不容易获得准确估计的原因）

1. 之前没提交过
2. 输入数据的变化
3. 数据放置（data placement）的变化
4. 输入数据分布的倾斜
5. 故障和短暂的资源利用率高峰可能会导致掉队，不仅掉队的持续时间不可预测，而且在用于预测任务的未来运行时间的数据集中表示异常值

（使用作业中所有任务的平均执行时间作为该作业中任务的执行时间，误差会非常大）

应对错误估计的措施

纠正机制，例如任务克隆和队列重新平衡，或者使用估计值分布而不是单值估计值。但是都没有从根本解决问题，还增加了系统的复杂度。

（LAS不需要关于任务运行时间的先验信息）

-------------------------------------------------------

短作业优先

Shortest Remaining Processing Time (SRPT)调度策略（需要运行时间估计）：按预期运行时间的递增顺序执行挂起的任务，并在较短的任务到达时抢占任务。相对于平均响应时间，SRPT被证明是最优的。

最近的很多系统都是使用的SRPT的变种，不支持抢占。

LAS（Least Attained Service，不需要先验估计，非常适合具有高差异运行时间的工作负载，例如重尾的数据中心工作负载）：与SRPT类似，LAS使用任务已经接收到的服务时间作为任务剩余运行时间的指示，支持抢占。如果有n个最年轻的任务，则会并发执行（类似于多道程序设计）。

LAS的基本原理

LAS使用获得的服务作为任务剩余服务需求的指示。这个预测在重尾服务需求分布中很有效。如果一个任务已经执行了很长时间，那么它很可能是一个大任务，因此要完成它还有很长的路要走。因此，最好执行更年轻的任务，这些任务更可能是短任务，因此完成得更快。

此外，根据定义，新的传入任务是最年轻的任务，并立即执行。假设是运行时间是重尾分布，那么这个新任务可能很短。如果在执行过程中没有其他任务到达，并且它完成的时间比任何其他等待任务获得的服务时间短，那么它将执行到完成，不需要任何抢占或排队。

7 RELATED WORK

基于运行时间估计的调度策略

Apollo[4]、Yaq[30]和Mercury[25]在工作节点传播预期的backlog。调度任务是为了最小化预期的排队延迟并均衡负载。Yaq还使用每个任务的运行时间估计来实现队列重新排序策略，旨在做到短任务优先。

Hawk[11]、Eagle[10]和Big-C[5]使用运行时间估计将作业分为长作业和短作业。在Eagle和Hawk中，工作节点集被划分为两个子集，它们的大小与每个类中的预期负载成比例。然后，根据作业的预期运行时间，将作业的任务发送到两个子集群中的其中一个。Big-C通过在YARN capacity scheduler中分配更高的优先级来优先处理短作业。工作负载划分和短任务优先的目的是减少[5,11]或消除[10]的排头阻塞。

Tetrisched [35], Rayon [9], Firmament [18], Quincy [24], Tetris [19], 3Sigma [29] and Medea [15] 将调度决策形式化为组合优化问题。所得到的混合整数线性规划要么得到精确解，要么利用启发式计算得到近似解。

Jockey[14]使用一个模拟器来推测系统的演变，并据此决定任务到节点的放置。Graphene[21]使用估计值来决定最复杂需求作业的放置，然后根据剩余的可用资源打包其他作业。Carabyne[20]暂时放松了公平保障，允许作业使用分配给其他作业的资源。

Big-C

Big-C使用可用的运行时间估计来执行任务放置，并在高利用率的情况下优先处理短任务。Big-C继承扩展了YARN的capacity scheduler。作业被分类为长短作业，并为每个类分配一个优先级和一定数量的节点。任务有机会在分配给不同类的节点上运行，但是当具有特定优先级的任务已经准备好运行，而其类中没有空闲节点时，优先级较低的任务将被抢占。

Big-C定义了两个作业类，分别对应长作业和短作业。作业根据可用的运行时间估计进行分类。短作业具有更高的优先级，并分配了大量的节点。在分配给短作业类的节点上投机地运行的长作业可能被新到达的短任务抢占。我们将Big-C配置为用于短作业的资源共享的默认值(95%)。

纠正机制

依赖于任务运行时间估计的系统还包含一些技术来处理不可避免的错误估计。

Tetrisched [35], 3Sigma [29], Rayon [9] and Jockey [14] 定期重新评估调度计划，以防任务的完成时间超出预期。

相比之下，Kairos使用抢占，通过接纳控制来限制队列不平衡的数量（参数Q）。Kairos可以集成推测执行或队列重新平衡技术，而代价是引入启发式方法来检测掉队者(例如，根据他们的进度)，并支持任务迁移(例如，基于检查点)。

（note：可作为对Kairos的改进）

有些系统如Rayon[9]、3Sigma[29]和Big-C[5]使用抢占来纠正调度决策，例如新任务到达时必须使用已经分配的资源。

Kairos的抢占，区别在于两个方面：

1. Kairos使用抢占来避免对运行时间估计的需求，这使得Kairos也适用于运行时间高度可变的环境，这些环境需要多次执行相同的作业，或者无法获得作业之前运行的数据。
2. 在Kairos中，除了允许短任务快速获得服务之外，还允许较长的任务轮流执行，从而确保了进度。

不基于运行时间估计的调度策略

Sparrow[28]通过批量抽样避免使用运行时估计。带有t个任务的作业将2t个探测发送到2t工作节点，在那里这些探测被加入队列。作业的一个任务在其中一个探测到达其队列的头部时得到服务。Sparrow改进了响应时间，因为作业中的t个任务是由连接的2t中负载最少的t个worker节点执行的。性能方面：Eagle > Hawk > Sparrow

（note：不知道任务运行时间的话，就不知道应该先调度谁，也不知道调度之后节点的负载均衡情况，这一段是解决了后者，且具有一定的随机性）

Tyrex[16]的目标是通过根据任务运行时间将工作负载划分到不同的类中，并将不同的类分配给工作节点的独立分区，从而避免排头阻塞。因为运行时间是未知的，所以工作负载分区是通过最初将所有任务分配给分区1，然后在任务执行时间超过阈值ti时将任务从分区i迁移到i + 1来实现的。

（note：有点强化学习的意思，利用任务迁移进行纠正）

Hu等人[22]的目标是根据任务到目前为止接收到的累计时间，在优先队列中组织作业，从而对短作业（不是短任务）进行优先级排序。高优先级队列中的作业分配的资源比低优先级队列中的作业多。任务驻留在centralized调度器上的系统级队列中，并根据相应作业的优先级分配给工作节点。

（note：用任务执行的累计时间来决定对应作业的优先级，然后根据作业的优先级来确定待调度任务的优先级（这个只是解决了优先级排序的问题））

与Kairos不同，在所有这些系统中都不支持抢占，任务一旦启动，就会运行到完成。因此，在高利用率的情况下，对延迟敏感的任务可能会导致排头阻塞和等待时间长。与此相反，Kairos使用抢占来允许传入任务在到达工作节点时立即运行，从而为短任务提供了在有限或没有等待时间的情况下完成的可能性，即使在高利用率的情况下也是如此。

调度器架构

Kairos可以归类为集中式调度器，因为所有任务都是由单个组件分派的，尽管工作节点也执行本地调度决策。

最近出现了向分布式调度器(如Omega[34]、Sparrow[28]、Apollo[4]和Yaq[30])或混合调度器(如Mercury[25]、Hawk[11]和Eagle[10])发展的趋势，以便在高作业到达率下实现低调度延迟。

Kairos可以在集中化的情况下维持高负载并实现低调度延迟，有两个原因

1. 它有效地在中央调度器和工作节点之间分配执行调度决策的负担
2. 任务到节点的分配策略很轻量

由于这些特性，我们认为Kairos也可以实现为分布式调度器。工作节点的状态可以在系统中传播，例如Apollo[4]和Yaq[30]，也可以在分布式调度器之间共享，例如Omega[34]。现有的技术，如randomly perturbing the state communicated to different schedulers [4]和atomic transactions over the shared view of the cluster [34]，可用于限制或避免不同调度器的并发冲突调度决策。

8 CONCLUSIONS AND FUTUREWORK

在本文中，我们提出了一种新的数据中心调度器Kairos，它不使用先验的任务运行时间估计。Kairos通过协同使用两种技术实现低延迟和高资源利用率。

首先，它使用一种轻量级的抢占形式来优先处理短任务而不是长任务，并避免排头阻塞。

其次，它采用了一种新颖的任务到节点的分配，减少了工作节点之间的负载不平衡，并将任务分配给节点，以提高它们快速完成的机会。

我们在YARN中使用一个成熟的原型在小范围内对Kairos进行了实验评估，并在更大的范围内进行了仿真。我们证明Kairos比使用先验任务运行时间估计的最先进的方法实现了更好的作业完成时间。

作为未来工作的一部分，我们计划扩展Kairos，使其更了解其他容器资源。目前，Kairos采用基于槽的分配系统，每个容器占用一个槽，槽是根据固定数量的内核定义的。在未来的工作中，我们首先计划让Kairos具有内存感知能力。目前，Kairos假设CPU是集群中的瓶颈资源，内存不是限制因素。其次，我们计划让Kairos处理异构的CPU分配，不同的容器可以分配不同数量的内核。

（note: 作者考虑了两个future work，但是都是考虑场景的限制提出的，例如再考虑内存，然后考虑异构CPU分配）