|  |  |
| --- | --- |
| **分 数：** |  |
| **评卷人：** |  |

****

**研究生（数据中心技术）课程论文（报告）**

**题 目：尾延迟优化的资源配置方法**

**学 号 M202073170**

**姓 名 王李光**

**专 业 电子信息**

**课程指导教师 施展 童薇**

**院（系、所） 武汉光电国家研究中心**

**2020年 12月15日**

尾延迟优化的资源配置方法

王李光1)

1)(华中科技大学武汉光电国家研究中心, 武汉 430074)

摘 要 现代应用往往部署在分布式系统上，具有高扇出的特点，这也使得尾延迟普遍存在。为了提升用户体验，满足服务级别目标（SLOs）的要求，如何优化尾延迟成为了一个非常重要的问题。优化资源配置是尾延迟优化的重要方面。一方面，我们需要根据给定的工作负载，确定所需的资源。另一方面，对于给定的工作负载，我们需要按照一定的标准对其进行划分，并将其映射给相应的资源。本文主要调研了从资源配置角度来优化尾延迟的三篇论文。Nguyen等人提出了一种黑盒尾延迟预测模型ForkTail，它将Fork节点看作是一个黑盒，并认为任务响应时间服从广义指数分布，从而建立起数学模型来预测Fork-Join结构的尾延迟。Didona等人针对请求的队列头部阻塞（Head-of-Line）和车队效应（Convoy Effect），提出了一种将长请求和短请求区分处理的方法Minos，它提出将长请求和短请求映射到不相交的核上进行处理，并保留一定比例的核专门处理短请求。Faisal等人针对现有方法只是对特定工作负载或者特定性能指标进行优化的现状，提出一种具有高鲁棒性的数据中心网络带宽优化技术2D，它通过多路复用和序列化实现了跨工作负载和性能指标的鲁棒性。

关键词 数据中心；尾延迟；资源配置；负载划分；资源映射

中图法分类号 TP302

A Survey on Resource Allocation Method for Tail Latency Optimization

Wang Liguang1)

1)( Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074)

**Abstract** Modern applications are often deployed on distributed systems and have the characteristics of high fan-out, which makes tail latency common. In order to improve user experience and meet the requirements of service level objectives (SLOs), how to optimize tail delay has become a very important issue. Optimizing resource allocation is an important aspect of tail latency optimization. On the one hand, we need to determine the resources required for a given workload. On the other hand, for a given workload, we need to divide it according to certain criteria and map it to corresponding resources. This paper mainly investigates three papers that optimize tail latency from the perspective of resource allocation. Nguyen et al. proposed ForkTail, a black box tail latency prediction model, which regarded the Fork node as a black box and thinked that the task response time followed the generalized exponential distribution, thus establishing a mathematical model to predict the tail latency of the Fork-Join structure. Didona et al. proposed Minos, a method to distinguish between long and short requests for head-of-line and Convoy Effect requests, which proposed mapping long and short requests to disjoint cores for processing, and reserving a percentage of the cores dedicated to short requests. Faisal et al. proposed a high robustness data center network bandwidth optimization technique 2D, which achieves robustness across workloads and performance metric through multiplexing and serialization, aiming at the current situation that existing methods only optimize specific workload or specific performance metric.

**Key words** Data center; Tail latency; Resource allocation; Workload division; Resource mapping

# 引言

现代应用往往部署在分布式系统上，具有高扇出的特点。所谓高扇出，是指一个模块需要去调用其他很多模块。由于应用部署在分布式系统上，这也就意味着需要被调用的模块可能并不在同一台服务器上，从而导致每个任务的响应时间并不完全相同。用户在使用服务的时候，往往关注的是最慢完成的请求。当用户发出一个请求，只有这个请求对应的所有任务都得到了完成，用户才能感受到完成了服务。因此，我们不能仅仅考虑请求响应的平均时间，还需要考虑尾延迟。所谓尾延迟，是指高百分位延迟。在当下，用户与应用之间的交互性越来越强，过大的尾延迟会给用户带来不良的体验。因此，我们需要关注如何降低尾延迟，来满足服务级别目标（SLOs）。

为了降低尾延迟，我们通常可以考虑为整个数据中心提供更多的资源，例如提供更多的处理器、更多的网络带宽。但是仅仅依靠增加资源并不能解决问题，因为我们的成本是有限的，资源并不能够无限增加。同时，数据中心面临的工作负载也不是一成不变的，在工作负载较低的时候，过多的资源使得数据中心的资源利用率较低，这从成本的角度考虑并不合算。

在进一步进行分析前，我们首先需要明确我们想要解决的问题，即给定服务级别目标（SLOs），在控制成本的前提下优化尾延迟，达到相应的要求。为此，我们可以从两个角度出发考虑解决问题。首先，从确定资源供给的角度，由于我们的成本是有限的，我们当然希望能够分配尽量合适的资源，因为这样使得我们能够在成本最低的情况下达到服务级别目标。我们如果想实现这样的目标，就需要考虑获得请求的尾延迟，根据尾延迟来对资源进行分配和调整。此外，从负载和资源映射的角度，面对大量的工作负载，我们考虑如何对这些负载进行分类，把它们映射到不同的资源上去，从而更好地利用我们的资源。这就使得我们需要考虑以什么样的标准来对工作负载进行分类和划分。我们把这两个解决问题的方向统一归类为尾延迟优化的资源配置方法。

对于资源供给，我们需要根据尾延迟来对所需的资源数量进行估计。资源数量估计的方法可分为两种，分别是基于预测的估计方法和基于反馈的估计方法。基于预测的估计方法需要建立一个尾延迟估计模型，通过模型来提前预测尾延迟，然后根据尾延迟的预测情况来分配资源，达到服务级别目标（SLOs）的要求。基于反馈的估计方法需要监测系统的运行状态，根据监测情况判断尾延迟是否满足响应的要求，并对资源进行调整。

对于负载和资源的映射，首先需要考虑以什么样的标准来对工作负载进行划分。常见的负载划分方法包括根据负载的处理开销进行划分和根据负载的类型进行划分。对于不同处理开销的负载，可以考虑确定一个阈值来对它们进行划分，然后将它们分别交由不同的资源进行处理。同时，也可以根据负载类型的不同来对负载进行划分，例如是写操作还是读操作，并把它们映射到不同的资源上，从而减小尾延迟。

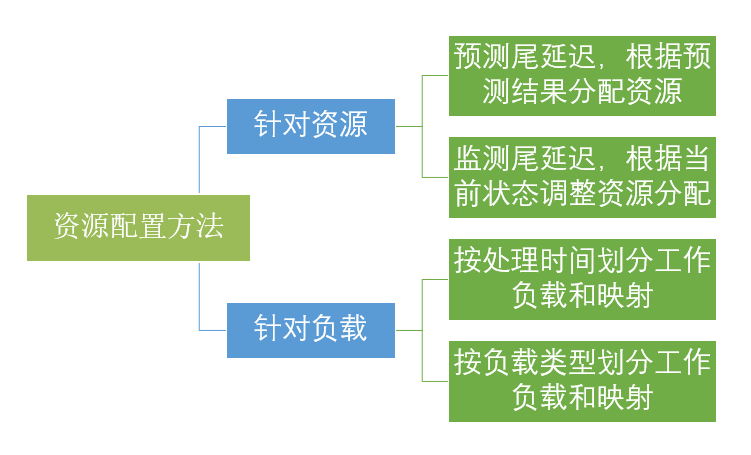


图 1尾延迟优化分资源分配方法

本次主要调研了三篇文献，第一篇文章《ForkTail: A Black-Box Fork-Join Tail Latency Prediction Modelfor User-Facing Datacenter Workloads》是HPDC’18的一篇文章。Nguyen等人在这篇文章中提出了一种黑盒尾延迟预测模型ForkTail，它将Fork节点看作是一个黑盒，并认为任务响应时间服从广义指数分布，从而建立起数学模型来预测Fork-Join结构的尾延迟。第二篇文章《Size-aware Sharding For Improving Tail Latencies in In-memory Key-value Stores》发表于NSDI’19，Didona等人在文章中针对请求的队列头部阻塞（Head-of-Line）和车队效应（Convoy Effect），提出了一种将长请求和短请求区分处理的方法Minos，它提出将长请求和短请求映射到不相交的核上进行处理，并保留一定比例的核专门处理短请求。第三篇文章《Workload Adaptive Flow Scheduling》来源于CoNEXT’18，在这篇文章中，Faisal等人针对现有方法只是对特定工作负载或者特定性能指标进行优化的现状，提出一种具有高鲁棒性的数据中心网络带宽优化技术2D，它通过多路复用和序列化实现了跨工作负载和性能指标的鲁棒性。接下来，本文将就上述三篇文章进行详尽地分析。

# ForkTail: 建模预测尾延迟

## 概述

### 2.1.1 研究背景

Fork-Tail结构是很多面向用户的数据中心服务的基础，例如web搜索和社交网络。所谓的Fork-Tail结构（如图2所示），是指用户的请求会被分成多个任务进行处理，这些任务被分配到不同的Fork节点上进行排队和处理。当任务处理完成后，汇总到Join节点产生结果。正因为Fork-Join结构这样的特性，使得预测请求的尾延迟非常困难，尤其是在负载很大的情况下。

作者认为，为了满足服务级别目标（SLOs），现有的数据中心通常采用过量提供资源的方法，然而这会导致数据中心的资源利用率变低，同时增加了数据中心的成本。接下来作者以Google和Twitter公司的例子来说明这个问题。作者指出，在一个有着12000台服务器的谷歌集群里，总的CPU和内存使用率大多低于50%，也就意味着有大量的资源几乎在任何时候都是空闲的。在Twitter公司的一个大型生产集群中也能发现上述现象。因此，如何在满足严格的SLOs的同时提高资源利用率或负载，是数据中心提供商面临的一个具有挑战性的问题。要想解决此问题，首先需要考虑如何在高负载情况下准确预测与各种Fork-Join结构相关的尾延迟。

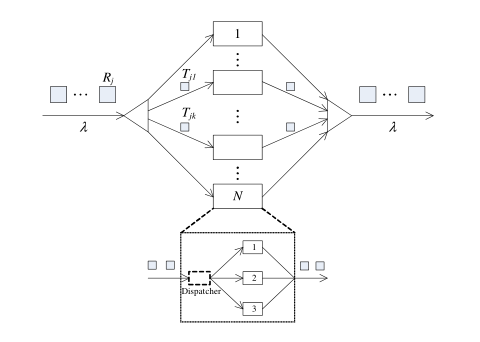


图 2Fork-Join结构

### 2.1.2 现有方法的不足之处

作者指出，现有的针对Fork-Join结构进行建模的方法主要是依靠一类称为Fork-Join排队网络(FJQNs)的排队网络模型。FJQNs是白盒模型，所有的Fork节点都被显式地建模为具有给定排队规则和服务时间分布的服务器。

然而，FJQNs模型具有很大的局限性。一方面，FJQNs非常难以求解，它们只能应用于非常有限的服务时间分布。另一方面，Fork节点的结构是多种多样的，每个Fork节点上可能有多台冗余服务器用于恢复，现有的FJQNs模型很难全面的覆盖多钟多样的结构。

### 2.1.3 作者提出的改进

作者设计了一种黑盒Fork-Join模型，称为ForkTail，期望涵盖尽可能多的Fork-Join结构。在Fork-Join模型中，每个Fork节点都被看作是一个黑盒，如图3所示。

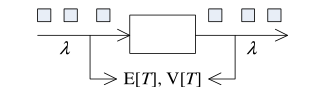


图 3看成是黑盒的Fork节点

此时，不再考虑Fork节点上有多少冗余服务器，也不再考虑任务在Fork节点上是怎么分布、排队和处理的。作者提出在高负荷区域，响应时间分布FT(x)可以近似为广义指数分布函数，最终任务的响应时间分布完全由任务的期望和方差决定。通过建立这样一个模型，可以对请求的尾延迟来进行预测。

## 设计

作者指出，根据前人的研究成果，对于一个G/G/m队列（即一般到达过程、一般服务时间和m台服务器），在重负载的情况下，等待时间分布可以近似为指数分布。显然，这个结论可以推广到请求的响应时间分布上，因为随着负载的增加，响应时间分布收敛于等待时间分布。直观地理解，在面对高负载时，长时间的排队效应有助于消除服务波动。

作者受上述四项的启发，提出对于在高负载的情况下映射到黑盒Fork节点上的任务，其响应时间可以被近似为一个广义指数分布，如公式（1）所示。其中的α为形状参数，β为扩展参数。



任务响应时间的期望和方差由公式（2）和公式（3）给出。



从公式（2）和公式（3）我们可以看到，公式(1)中的分布完全由任务响应时间的均值和方差决定。作者使用这种分布而不是直接使用指数分布的原因是它可以根据参数设置来对应重尾分布或者轻尾分布的工作负载。

作者假设映射到不同Fork节点上的任务的响应时间是独立的随机变量，则多个任务的响应时间分布可以写成是每个任务响应时间分布的连乘积，如公式（4）所示。



作者将尾延迟定义为高百分的响应时间，如公式（5）所示。

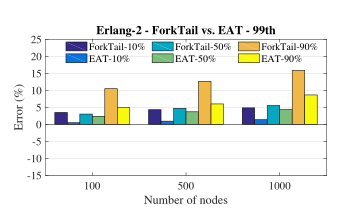
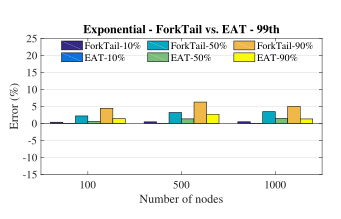


在使用该模型预测尾延迟时，如果已知任务响应时间的分布情况，知道响应时间分布的情况，首先计算给定任务服务时间分布的均值、方差。然后，将上述平均值和方差代入公式，用于近似任务响应时间分布的广义指数分布的α参数和β参数，最后根据公式计算尾延迟。如果提前不知道任务的响应时间分布，则需要根据经验确定期望和方差。

## 实验

为了评估效果，作者对k=N（即任务数等于Fork节点数）和k<N（即任务数小于Fork节点数）的情况分别进行了评估。为简洁起见，本文只选择部分实验结果进行介绍。

首先，作者选用了Erlang-2,、Exponential、Hyperexponential-2等三种服务时间分布，在10%、50%、90%负载的情况下与白盒方法EAT进行了比较，实验结果如图4所示。



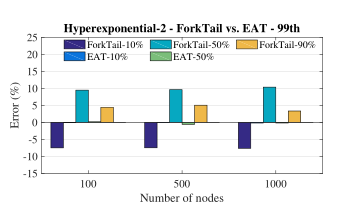
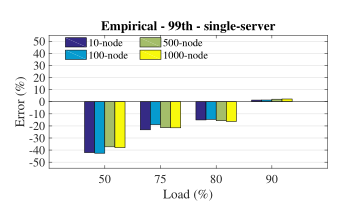
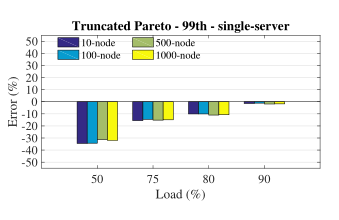


图 4实验结果

通过实验结果可以看出，虽然与EAT相比，ForkTail的误差较大，但仍然维持在10%-20%内。同时，EAT需要根据Fork节点的构成进行调整，还需要考虑任务的排队情况。为了覆盖足够大的工作负载空间，作者进一步考虑具有重尾服务时间分布Empirical、Truncated Pareto和Weibull，使用它们来衡量ForkTail的尾延迟预测误差。这类分布在实际中很常见，EAT方法处理起来比较困难。实验结果如图5所示。



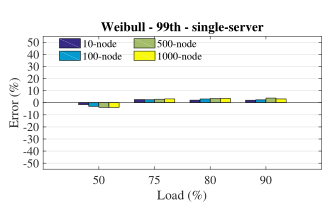


图 5ForkTail实验结果

从实验结果中可以看出，在高工作负载下，Fork-Tail的尾延迟预测误差可以保持在10%-20%内，具有较好的性能。

# Minos: 区分处理长短请求

## 概述

### 3.1.1 研究背景

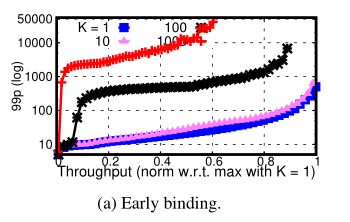
作者指出，当前许多分布式应用使用内存中的键值对存储作为缓存或者数据库，这些应用普遍呈现出高扇出的特点，也就是说他们需要并行地发出大量请求。从应用程序的角度来看，总体响应时间由这些请求中最慢的响应决定，因此优化尾延迟显得非常重要。

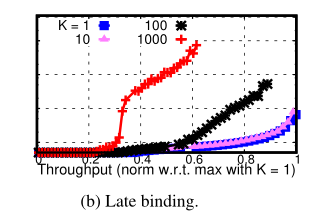
作者通过实验观察到KV存储中的工作负载由大量短请求和少量长请求组成。对于长请求，由于他们的服务时间较长，处理它们将消耗大量可用资源，增加了出现队列头部阻塞（head-of-line blocking）的可能性，也就是短请求排在长请求后面迟迟得不到处理，从而出现了额外的延迟。即使只有少量的长请求，也会显著提高尾延迟。

3.1.2 现有方法存在的问题

作者指出现有的策略通常不考虑请求的大小，主要包括早绑定（Early binding）、晚绑定（Late binding）、伴随窃取的早绑定（Early binding with work stealing）。早绑定是指请求被分派到特定核心的队列，通常基于键的哈希。晚绑定是指请求被保存在一个单一队列中并被调度给空闲的核。伴随窃取的早绑定是指请求按照早期绑定的方式处理，但除此之外，空闲内核还会从其他内核的队列中窃取请求。

作者的实验结果表明，与请求大小相同的工作负载相比，即使只有少量的长请求，上述三种策略的99百分位尾延迟也会大幅增加，实验结果如图6所示。其中，K表示长请求所占的时间单位。





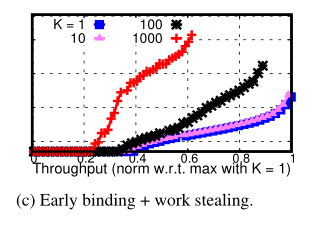


图 6三种不同策略实验结果

### 3.1.3 作者提出的改进

作者提出了大小感知分片的概念来解决上面提到的问题。感知大小的分片意味着对不同大小的请求被发送到不同的核心来进行处理。

作者首先确定一个阈值，根据阈值来划分长请求和短请求，然后将长请求和短请求发送给不相交的内核进行处理。此外，按照一定比例划分出专门的核心来处理短请求，以防止出现车队效应（Convoy Effect）。

## 设计

### 3.2.1 确定长请求和短请求划分的阈值

作者提出，优化N百分位延迟，也就是优化前N%小的请求的延迟。所以作者将前N%小的请求看作短请求。在计算阈值的时候，首先获取到请求大小的累积分布函数（CDF），然后将前N%小的请求划分为短请求，从而得到了长请求和短请求划分的阈值。

### 3.2.2 确定用于处理短请求的核的数量

Minos维护一个成本函数，该函数为给定大小的请求提供特定的处理成本。在此函数中，作者将请求处理的网络数据包数量作为成本，既可以是传入PUT请求中的数据包数量，也可以是传出GET应答中的数据包数量。通过成本函数，作者将处理短请求的核心数量设置为短请求占总处理成本的比例乘上核心数量总数。

### 3.2.3 调度策略

作者只让处理短请求的核来读取接收队列中的传入请求。处理短请求的核首先查看请求大小，如果是短请求，那么就进行处理，如果是长请求，那么就调度给负责处理长请求的核进行处理。作者只让处理短请求的核来读取接收队列中的传入请求的原因在于，如果处理长请求的核接收到一个短请求这个请求可能会在长请求之后经历队列头部阻塞。

## 实验

为了测试方法的性能，作者选择与三种没有采用请求大小感知策略的方法进行对比，它们分别是HKH、SHO、HKH+WS。其中，HKH这个系统实现了请求到核心的早期绑定，也就是本文在前面提到的早绑定（Early binding）。SHO系统实现了请求到核心的后期绑定，即本文在前面提到的晚绑定（Late binding）。HKH+WS系统在HKH之上实现请求窃取，即本文前面提到的伴随窃取的早绑定（Early binding with work stealing）。实验结果如图7所示。

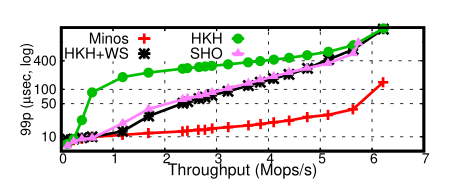


图 7Minos对比实验结果

从实验结果中可以看出，在不同的吞吐量下，Minos实现了最低的尾延迟。比HKH好的原因在于，Minos通过请求大小感知解决了队列头部阻塞的问题。比SHO好的原因在于，Minos通过分配一定比例的核专门处理短请求避免了车队效应。

为了进一步评估性能，作者还使用了具有不同PUT和GET请求比例的工作负载。在PUT和GET请求各占50%的工作负载中，作者同样将Minos与上述三种方法进行了对比，得到的实验结果展示在图8中。从图中可以看出，Minos同样展示了非常良好的性能。

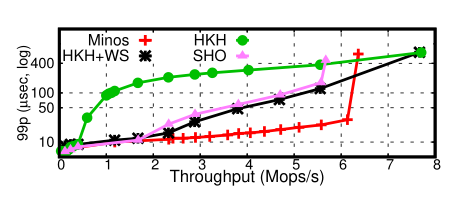
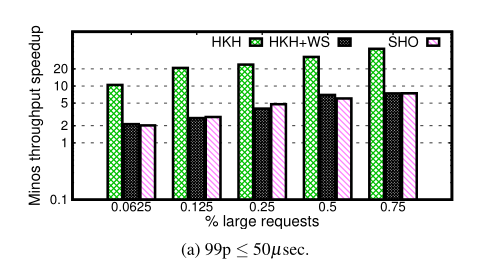


图 8Minos实验结果



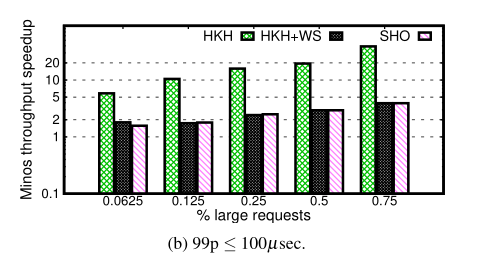
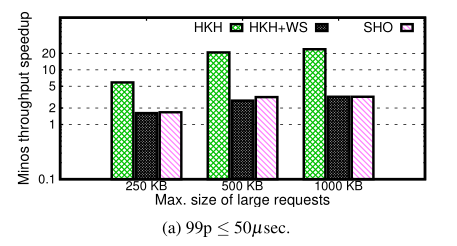


图 9不同长请求比例实验结果



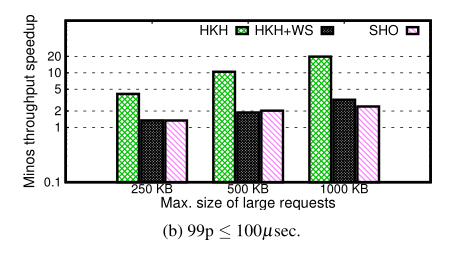


图 10不同请求大小实验结果

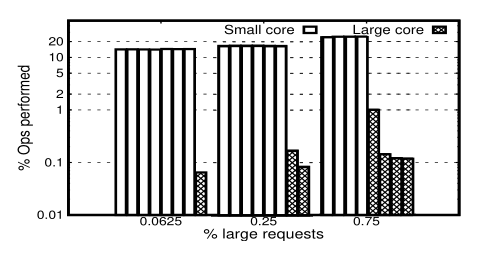


图 11每个核处理的请求数量的百分比

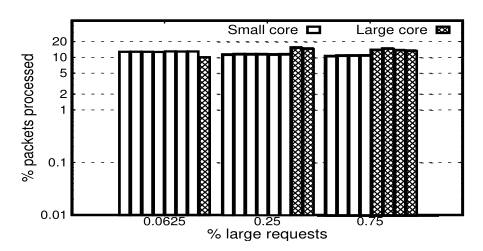


图 12每个核处理的数据包数量的百分比

作者还测试了在工作负载中长请求大小和比例不同时，在给定服务级别目标（SLOs）前提下，不同方法可实现的吞吐量。在进行此项实验时，改变其中一个参数，保证另外一个参数不变，实验结果如图9、10所示。

图9展示了对于给定的第99个百分位延迟SLO，使用占不同百分比的长请求可以实现的最大吞吐量，y轴是取log以后的Minos相对于其他三种方法的提升情况，图10展示了对于给定的第99百分位延迟SLO，使用不同最大大小的长请求可实现的最大吞吐量，y轴也是取log以后的Minos相对于其他三种方法的提升情况。可以看到，Minos相较于其他三种方法均有较大的提升。

作者也测试了Minos的负载均衡情况。图11显示了每个核处理的请求数量的百分比，图12展示了每个核处理的数据包数量的百分比。从图中来看，虽然负责处理短请求的核需要处理的请求更多，但是短请求的数据包较少，这使得所有核心处理的数据包数量大致相同，因此大致执行相同的工作量，充分显示了给予大小感知分片的Minos的良好性能。

为简洁起见，作者进行的其他测试本文不再一一介绍。总的来说，在本文中，Didona等人提出的Minos将长请求和短请求映射到不相交的核上进行处理，并保留一定比例的核专门处理短请求，较好的解决了常见的队列头部阻塞（Head-of-Line）和车队效应（Convoy Effect）现象。

# 2D: 序列化和多路复用分配带宽

## 概述

### 4.1.1 研究背景

作者认为现有的数据中心网络流量调度方案都是针对特定的工作负载和性能指标进行优化。这就意味着针对一种工作负载的最优调度策略对于不同的工作负载可能是一个糟糕的选择。例如，FIFO对于轻尾工作负载是最优的，但是忽略了队列头部阻塞（head-of-line blocking），这使得它对于重尾工作负载的性能很差。

作者认为，工作负载不是一成不变的，反而是具有高可变性，这个高可变性体现在工作负载内部和不同的工作负载之间。对于工作负载内部的高可变性，作者认为大多数流行的网络工作负载（例如网络搜索、数据挖掘等）遵循重尾流大小分布，即这种工作负载是混合了长流和短流的。然而，虽然它们被称为重尾工作负载，但是在较短的时间尺度下，工作负载可能只有短流没有长流，表现成为一个轻尾分布的工作负载。

作者通过实验对这一现象进行了观察，观察结果如表1所示。

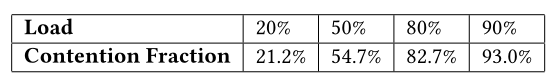


表 1观察结果

从上图中可以看出，即使是重尾分布的工作负载，也会存在大量的短流连续出现的现象，这表明工作负载内部存在可变性。

作者认为不同工作负载之间也存在着高可变性，这是因为每个集群都运行着过个应用程序或作业，从网络流量调度的角度来看，由于轻尾工作负载和重尾工作负载之间存在着复用，这可能导致轻尾工作负载变成重尾工作负载。另外，即使是轻尾工作负载，它们的流大小可能也很不相同，当它们聚合在一起时，可能会变成重尾分布的工作负载。

### 4.1.2 现有方法存在的问题

作者考虑了三种众所周知的调度策略：PS、SRPT和FIFO。上述三种调度策略受到了两个关键限制，首先是它们通常针对固定的工作负载进行优化，如果用于不同的工作负载，它们的性能可能会受到严重影响，这就不适合面对作者指出的工作负载具有可变性的问题。另外，上述方法都针对固定的性能指标进行优化，例如平均流完成时间或者尾流完成时间，而对于一些苛刻的应用程序，减少平均完成时间和尾流完成时间都很重要。作者将上述情况归纳成了一个表格，如表2所示。



表 2情况归纳

作者的实验结果如图13所示。

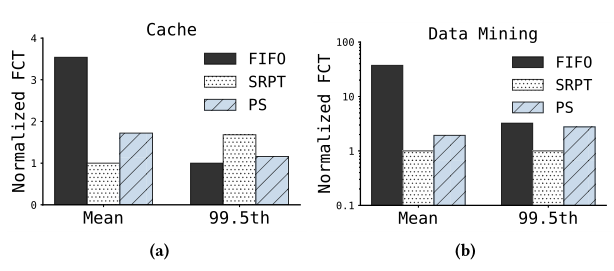


图 13现有方法实验结果

实验结果表明，对于一种工作负载没有明显的最优策略。例如从图13（a）中可以看出对于高速缓存工作负载，FIFO的尾部性能最佳，但平均性能比最佳方案差3.5倍。从图13（b）中可以看出，没有跨工作负载的优化策略。FIFO是缓存工作负载的最佳尾优化策略，它在重尾工作负载(数据挖掘)的尾部性能最差，与性能最佳的策略相比，完成时间最多增加了3倍。

### 4.1.3 作者提出的改进

作者认为在当下，了解工作负载的分布情况是可行的，这也就表明可以考虑设计设计自适应调度策略。

作者提出了一种名为2D的方法，并为自己的工作提出了两个设计目标，主要目标是在所有工作负载中实现尾部性能优化。具体来说，对于给定的工作负载，它应该至少在尾部百分位(例如99百分位)具有与该工作负载的已知最佳方案(FIFO或PS)一样好的性能。第二个目标是优化低百分位的平均完成时间。总的来说，2D的目标就是优化高百分位的性能，同时在较低的百分位数上不影响性能。

作者提出将多路复用和序列化相结合。多路复用可以有效避免队列头部阻塞的问题，而对于大小相似的流进行多路复用则会使作业完成时间增加，此时需要进行序列化。因此，作者提出将多路复用用在不同大小的流之间，将序列化用在大小相似的流之间，通过这样结合的方式优化数据中心网络带宽。

## 设计

作者提出按照流的大小来进行分类，对于大小相似的流，将它们分到同一类中，然后按照先进先出的顺序为类中的每个流提供服务。之所以在同一类的流中采用先进先出的原则，是因为在之前的实验中证明了FIFO的尾部性能最佳，本文在上一小节也提到了这一点。同时，对于同一类的流，采用先进先出的策略最简单，非常容易实现。

对于大小差异大的流，作者将它们分到不同的类中，然后对于不同的类以加权的方式进行多路复用，确保能够实现跨不同的工作负载的良好性能。

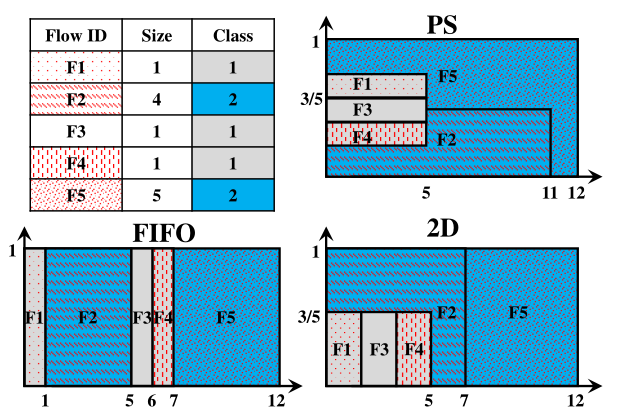


图 14 2D和其他方案的工作情况比较

从图14中我们可以看出，有两种不同大小的流量(小流量和大流量)以在线方式到达，对于FIFO，存在着队列头部阻塞的问题，而对于PS，存在着相似大小的流之间不必要的多路复用。2D将工作流分为两类，通过多路复用避免了队列头部阻塞的问题，通过每个类内的序列化避免了不必要的复用。

作者提出了一种根据变异系数（Coeffi-cient of Variation）的平方来对流进行分类的算法，通过这种算法来确定大流和小流之间的阈值。通过算法计算变异系数的平方，最终每个类内所有流之间的额变异系数的平方均小于1。

对于确定每一类的工作流所占有的带宽，作者提出给每一类确定不同的权重，这个权重可以由公式（1）计算。



其中分子是某一类中包含的流的数目，分母是所有流的数目。通过这样的方式，我们就确定了每一类流所占有的带宽。

2D的完整结构如图16所示

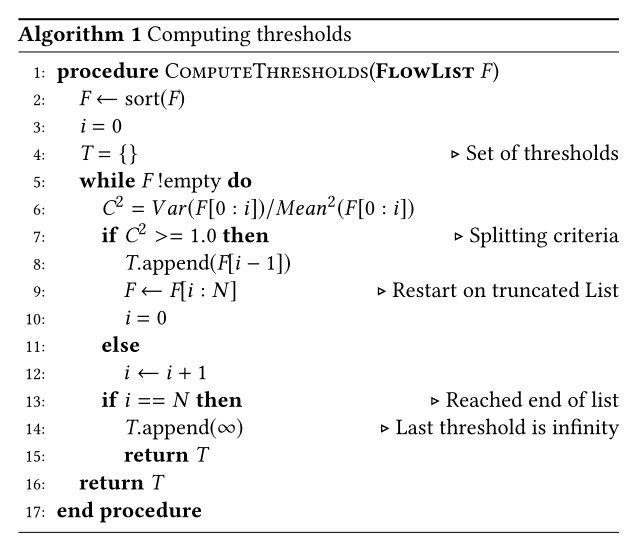


图 15确定大小流阈值的方法

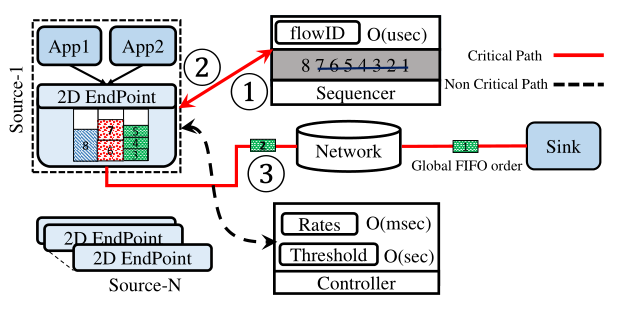
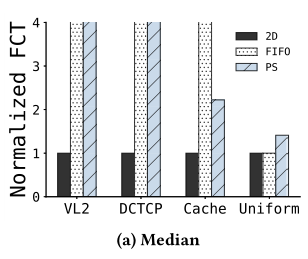
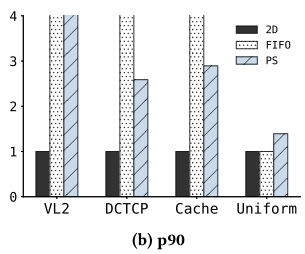


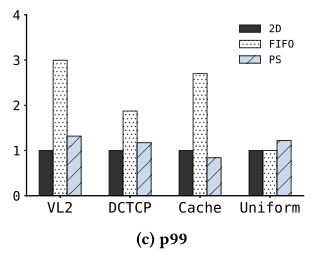
图 16 2D的完整结构

## 实验

为了测试2D的性能，作者使用了三种真实工作负载和一种合成的均匀工作负载。第一个工作负载来自包含长短流混合的网络搜索应用程序(DCTCP)，其方差较大。第二种流量大小分布来自数据挖掘集群（VL2），这种分布非常不均衡。第三个工作负载来自Facebook的缓存集群（Cache），这种工作负载是一种轻尾分布的负载。作者将合成的均匀分布的工作负载记为Uniform。2D与FIFO和PS在四种工作负载上的表现如图17所示。







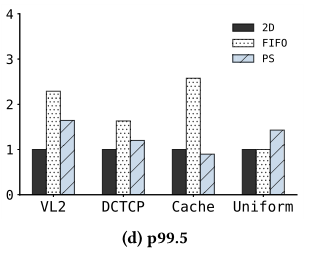


图 17不同方法在工作负载上的表现

为了方便观察，作者以2D为准进行了正则化。从图中可以看出，对于尾流完成时间（图d），即使在最坏的情况下，2D也要比FIFO和PS好9%左右，而流完成时间的中位数比其他方法提高了12倍，平均流完成时间提高了1.7倍。

图18显示了在重尾的VL2工作负载上，流完成时间的累计分布函数。从图中可以看出，2D通过将不同大小的流分类进行多路复用，以及在相似大小的流中进行序列化，使得每个流的延迟显著降低。

为了阐释性能，作者还在更多的工作负载上和多种方法进行了比较，为了简洁起见本文不再一一描述。

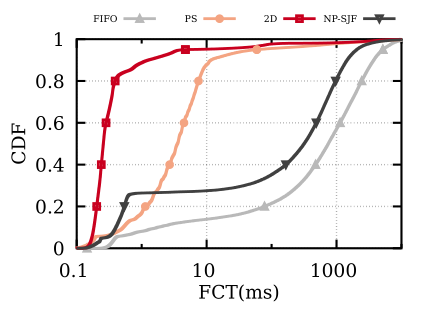


图 18VL2工作负载上不同方法的CDF

综上所述，Faisal等人提出的数据中心网络带宽优化技术2D，通过多路复用和序列化实现了跨工作负载和性能指标的鲁棒性。

# 总结

尾延迟优化的资源配置方法，一类是通过估计尾延迟，来对达到服务级别目标（SLOs）所需要的资源进行估计。另一类则是通过负载和资源的动态映射，使得资源得到充分利用，降低尾延迟。本文通过调研三篇文献，简要叙述了尾延迟优化的资源配置方法。

# 参考文献

[1] Didona Diego and Zwaenepoel Willy. Size-aware Sharding For Improving Tail Latencies in In-memory Key-value Stores. in: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). Boston, MA, USA: USENIX Association, February 26–28, 2019. 79–94.

[2] Faisal Abdullah Bin, Bashir Hafiz Mohsin, Qazi Ihsan Ayyub, et al. Workload Adaptive Flow Scheduling. in: Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies. Heraklion, Greece: ACM, December 04-07, 2018. 241–253.

[3] Nguyen Minh, Alesawi Sami, Li Ning, et al. ForkTail: A Black-box Fork-join Tail Latency Prediction Model for User-facing Datacenter Workloads. in: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing. Tempe, Arizona: ACM, June 11-15, 2018. 206–217.