



学士学位论文

THESIS OF BACHELOR



论文题目：部署轻量级 AI 的负载均衡算法研究

学生姓名：王浩宇

学生学号：515021910430

专业：信息工程

指导教师：蒋铃鸽

学院(系)：电子系

## 上海交通大学 学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：\_\_\_\_\_

日期：\_\_\_\_\_年\_\_\_\_\_月\_\_\_\_\_日

## 上海交通大学 学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保 密口，在 \_\_\_\_\_ 年解密后适用本授权书。  
不保密口。

(请在以上方框内打√)

学位论文作者签名：\_\_\_\_\_ 指导教师签名：\_\_\_\_\_

日 期：\_\_\_\_\_年\_\_\_\_月\_\_\_\_日

日 期：\_\_\_\_\_年\_\_\_\_月\_\_\_\_日

# 部署轻量级 AI 的负载均衡算法研究

## 摘要

第五代移动通信系统的基站中普遍采用了数据中心网络以实现各种应用数据的处理，其中负载均衡算法是发挥其大规模、高健壮性、高带宽、低延迟等特点的基石。本文通过总结三个领域的最新进展：深度增强学习算法、数据中心网络中的数据流负载均衡算法和交换机处的缓冲区流量调度算法，针对 5G Cloud RAN 场景下的负载均衡问题提出了一个新的解决方案。此处的负载均衡是指广义的负载均衡，即包含了从端系统到交换机到端系统整体路径上的负载均衡，涵盖数据流的优先级标记、交换机缓存处的入队、出队操作及选路、拥塞信息采集等一系列过程。该解决方案是一个混合型的结构，对于短数据流采用性能极佳的 DRILL 分布式负载均衡算法，基于本地信息给出快速的均衡决策；利用离散形式的策略梯度算法和决定性（连续空间下）的 actor-critic 算法集中式地解决了长数据流的选路、流速率、优先级和对于网络架构中交换机处采用的 MLFQ 的队列阈值的实时部署，根据动态的网络流量情况，以及以往数据得到的经验，给出一种接近最优的流量优化方法。

**关键词：**负载均衡 增强学习 数据中心网络 5G Cloud RAN

# A STUDY ON LOAD BALANCING ALGORITHM WITH LIGHTWEIGHT AI

## ABSTRACT

The base stations in 5G wireless communication network generally utilize data center network to achieve the processing of various applications, and the load balancing algorithm employed in it is the significant foundation that helps to realize its features of big-scale, robustness, high-bandwidth, and low latency. This thesis deals with the general load balancing problem in data center networks. Under the scenario of the Cloud Radio Access Network of the fifth generation of wireless communication system, we propose a new paradigm for a combination of load balancing algorithm and flow scheduling algorithm. The system is a hybrid model with several hierarchies, namely centralized flow scheduling component and distributed load balancing component. The centralized flow scheduling scheme employs the state-of-the-art deep learning algorithm, Deep Deterministic Policy Gradient and Stochastic Policy Gradient algorithm, to solve the problem of assignment of rate limit, queue priorities and path selection for the long flows in the datacenter network and the thresholds of the Multi Level Feedback Queue commonly used in modern commodity switches. The distributed load balancing algorithm, which addresses load balancing problem of narrow sense, utilizes the recent model DRILL proposed by Soudeh Ghorbani, aiming at achieving lower flow complete time and lower total transmission time, not to mention the raising of rate of link utilization.

**KEY WORDS:** load balancing, reinforcement learning, data center network, 5G Cloud RAN

# 目 录

插图索引	v
<b>第一章 绪论</b>	<b>1</b>
1.1 课题研究背景 . . . . .	1
1.2 国内外研究现状 . . . . .	1
1.3 本文研究思路及组织结构 . . . . .	2
<b>第二章 增强学习算法</b>	<b>3</b>
2.1 基于值函数的增强学习 . . . . .	3
2.1.1 深度模型 DQN . . . . .	3
2.1.2 经验回放 . . . . .	4
2.1.3 Double DQN . . . . .	4
2.2 基于策略的增强学习 . . . . .	5
2.2.1 Stochastic 策略梯度方法 . . . . .	5
2.2.2 Deterministic 策略梯度方法 . . . . .	6
2.3 Actor-Critic 方法 . . . . .	7
2.3.1 DPG 算法 . . . . .	7
2.3.2 DDPG 算法 . . . . .	7
2.4 本章小结 . . . . .	8
<b>第三章 5G Cloud RAN 中的负载均衡</b>	<b>10</b>
3.1 无线移动通信系统基站的演进 . . . . .	10
3.2 5G 时代的数据中心网络 . . . . .	10
3.2.1 网络架构 . . . . .	10
3.2.2 流量特征 . . . . .	12
3.3 负载均衡问题 . . . . .	13
3.4 数据中心网络中的负载均衡算法 . . . . .	13
3.4.1 定义、目标与主要步骤 . . . . .	13
3.4.2 数据中心与 WAN 中负载均衡的比较 . . . . .	15
3.4.3 近年来提出的负载均衡算法 . . . . .	16
3.5 数据中心网络的流量调度 . . . . .	21
3.5.1 背景 . . . . .	22
3.5.2 调度算法 . . . . .	22
3.5.3 PIAS 架构下的流量调度 . . . . .	24

---

<b>第四章 基于深度增强学习的流量优化系统</b>	<b>25</b>
4.1 系统架构 . . . . .	25
4.1.1 集中式流量调度 . . . . .	25
4.1.2 分布式负载均衡 . . . . .	27
4.1.3 网络拓扑 . . . . .	29
4.2 仿真过程 . . . . .	30
4.2.1 仿真数据源 . . . . .	30
4.2.2 仿真环境搭建 . . . . .	30
4.2.3 增强学习深度模型的训练 . . . . .	31
4.2.4 仿真结果 . . . . .	31
4.2.5 性能分析 . . . . .	34
<b>第五章 结论</b>	<b>35</b>
<b>参考文献</b>	<b>36</b>
<b>致 谢</b>	<b>39</b>

## 插图索引

2-1 Deep Q-Network 结构图 . . . . .	4
2-2 Actor-Critic 结构图 . . . . .	7
3-1 RAN (左) 与 Cloud RAN (右) <sup>[15]</sup> . . . . .	10
3-2 Fat-Tree 架构 (图中为 $k = 4$ 的情形) <sup>[16]</sup> . . . . .	11
3-3 CamCube 架构, 蓝点为服务器 <sup>[18]</sup> . . . . .	11
3-4 BCube 架构, 图中方框代表交换机, 圆圈代表服务器 <sup>[20]</sup> . . . . .	12
3-5 Fastpass 结构图 . . . . .	17
3-6 MPTCP 在协议栈中的位置 . . . . .	18
3-7 CONGA 的主要结构 . . . . .	19
3-8 Expeditus 的网络结构 . . . . .	20
3-9 LocalFlow 中的仿真网络拓扑: 采用 4-端口交换机的 fat-tree 架构 . . . . .	21
3-10 长尾分布示意图 . . . . .	23
3-11 PIAS 中的多级反馈队列 . . . . .	24
4-1 系统整体架构 . . . . .	25
4-2 DDPG 网络模型在 tensorboard 中的结构图 . . . . .	27
4-3 Stochastic 策略梯度方法的神经网络在 tensorboard 中的结构图 . . . . .	28
4-4 仿真所用的网络拓扑 . . . . .	29
4-5 构建网络拓扑的部分源代码 . . . . .	29
4-6 仿真数据源 . . . . .	30
4-7 从 dockerfile 构建自己的镜像, 其中 “From jaimeps/rl-gym:latest” 这一行显示了本仿真环境的基础镜像 rl-gym, 后续行均为在其基础上构建 ns-3 环境的过程 . . . . .	31
4-8 Docker 平台与容器 . . . . .	31
4-9 FCT 与训练时长折线图 . . . . .	32
4-10 Stochastic Policy Gradient 训练过程 . . . . .	32
4-11 数据流完成时间 . . . . .	33
4-12 不同发送端的总传输时延 . . . . .	33
4-13 上行平均链路利用率 . . . . .	34
4-14 下行平均链路利用率 . . . . .	34

# 第一章 绪论

## 1.1 课题研究背景

在第一代与第二代无线移动通信中，蜂窝网络（cellular）或无线电接入网（Radio Access Networks，缩写 RAN）由相互独立的基站组成，每个基站配有电力、备用电源、空调系统、环境监测、回程链路等其他支持设施。每个基站内部含有射频信号发射单元，射频信号通过电缆从地面机柜传输至位于高处的基站天线。为了降低射频信号在电缆上的传输损耗，第三代和第四代移动通信采用了分布式基站系统。分布式基站将射频拉远单元（Remote Radio Head，缩写 RRH）与基带信号单元（Baseband Unit，缩写 BBU）分离，使 RRH 可以直接部署在基站塔顶距离天线很近的地方，并通过光纤的连接使两者间的距离变大（可达几百米或几千米）而损耗减小。随着 4G 网络的商用和普遍部署，蜂窝网络的流量负载逐年增长，为满足用户对于虚拟现实、增强现实、4K、8K 视频等应用的流量需求以及越来越高的通信体验要求和万物互联的多层次覆盖，5G 的研发工作已经广泛进行。作为 5G 的关键技术，Cloud RAN 为无线接入网引入了云技术，使很多 BBU 集中到一个 BBU 池。通过应用最近的数据中心网络技术，Cloud RAN 能够在 BBU 池中的 BBU 之间建立低功耗、高可靠、低延迟、高带宽的连接。它应用了开放的平台与实时虚拟化技术以实现动态共享的资源分配并支持多厂商、多种技术并存的环境。在这种技术架构下，由于 BBU 池覆盖的范围比原来的一个基站所能覆盖的范围大得多（即一个用户从一个 RRH 切换到另一个 RRH 时，他前后连接的两个 BBU 在同一个 BBU 池中<sup>[1]</sup>），蜂窝系统中流量分布不均匀的问题可以通过在 BBU 池中部署灵活的、动态自适应的负载均衡算法得以解决。

## 1.2 国内外研究现状

传统蜂窝网络中的负载均衡算法已被大量研究，如小区呼吸技术<sup>[2]</sup> 已被广泛应用于第二代和第三代移动通信网络中。由 X. Lin 和 S. Wang 提出的 Cloud RAN 中的高效 RRH 切换机制在考虑系统能效条件下实现了负载均衡<sup>[3]</sup>。C. Ran 和 S. Wang 等人<sup>[1]</sup> 从另一个角度提出了 Cloud RAN 中的最佳负载均衡算法。他们周期性地监控衡量均衡度的公平指数，当其低于特定阈值时，则重新设计每个小区覆盖的区域以实现系统的负载均衡。C. Tsai 和 M. Moh 在 Cloud RAN 架构下比较了 8 种均以降低物联网（Internet of Things，简称 IoT）设备通信延迟为目标的负载均衡算法。这些算法或者采用预先定义的指标与阈值以实现监测与负载均衡，或者采用了较为固定而非实时改变的策略。与之相反，B. Shahriari 和 M. Moh 在部分可见的动态环境的情景下提出的一般线上学习（Generic Online Learning，简称 GOL）系统可以很好地应用于实时解决 5G Cloud RAN 中的负载均衡问题。但是，两人的工作主要是针对数据中心网络中虚拟机的高速缓存与云存储上的负载均衡内容，与本课题主要关注的数据流的选路及拥塞信息采集不同。现阶段主流的数据中心负载均衡算法分为两类，一类是集中式负载均衡（如 Hedera，Fastpass 等），拥塞信息采集的工作由中心化的操作完成，这一类算法可以关注到全局的数据特点，更好地进行选路，但是一般而言其信息流的延迟较高；另一类算法是分布式负载均衡算法（如 CONGA，CLOVE 等），它们利用本地的信息使交换机作出选路等决策，有

较低延迟，但对于全局信息的掌握不像集中式那样清晰。与负载均衡减少数据流完成时间的目标相同，交换机处缓存的流量调度机制也被广泛研究，Wei Bai 等人提出的 PIAS 架构利用多优先级反馈队列，对数据流的优先级进行细化标记，提升了流量调度的性能。一般而言，目前的负载均衡算法与流量调度算法在研究时均默认两者中的另一方是性能优良的算法，本文将二者联合起来考虑，构建一个综合系统，以实现更好的流量优化。

前面提到的 GOL 系统中所采用的学习算法是人工智能领域结合深度学习与增强学习的一种决策优化算法，是针对实时决策的最好算法之一。以此方法为核心的 AlphaGo<sup>[4]</sup> 接连战胜了人类最顶尖的围棋高手，成为举世瞩目的焦点。增强学习在与环境的交互过程中学习。在交互的每一步，智能系统会基于环境的状态选择一个动作以改变环境的状态，并受到相应的奖励或惩罚。增强学习在解决模型事先未知的情景下表现得非常有用。由于其可与环境交互并从交互过程中得到的奖励或惩罚学习更好的决策，它已被应用于各个领域，如计算机视觉中的姿态检测与跟踪问题、自然语言处理中的命名实体识别、知识图谱中的关系预测等，最近也被应用于通信网络中，如数据中心网络 coflow 调度机制问题。

### 1.3 本文研究思路及组织结构

在 B. Shahriari 和 M. Moh 工作的启发下，本课题旨在采用深度增强学习的方法提升 Cloud RAN 架构中负载均衡算法的性能。通过利用深度学习可以从环境与智能系统交互及历史信息等中提取抽象特征的能力，并借助增强学习系统感知环境、接收奖励与惩罚的能力，构建增强学习算法解决均衡问题。本论文的第二章讨论增强学习算法，从基于值函数的增强学习、基于策略的增强学习、actor-critic 算法三方面介绍深度神经网络与三种增强学习算法的结合；第三章讨论 5G Cloud RAN 中的负载均衡问题，此处的负载均衡是指广义的负载均衡，即包含了从端系统到交换机到端系统整体路径上的负载均衡，涵盖数据流的优先级标记、交换机缓存处的入队、出队操作及选路、拥塞信息采集等一系列过程，在对 5G 时代背景下的数据中心有了基本认识之后讨论目前的负载均衡及流量调度算法，如 MPTCP、CONGA、Expeditus、LocalFlow、DRILL、PIAS 等；第四章介绍本文提出的基于深度增强学习的流量优化系统，从系统架构和仿真过程两方面阐述系统的实现、仿真环境、性能分析等内容；最后总结全文，并对基于本文的进一步的研究提出一些见解。

## 第二章 增强学习算法

增强学习是一类算法，是让计算机实现从一开始什么都不懂，通过不断地尝试，从错误中学习，最后找到规律，学会达到目的的方法，这就是一个完整的增强学习过程。实际中的增强学习例子有很多，比如在 2016 年和 2018 年分别打败了人类最顶尖的棋手李世石和柯洁的 AlphaGo<sup>[4]</sup>，是机器第一次在围棋场上战胜人类高手；Google Deepmind 和 OpenAI 的研究者们让计算机自己学着玩游戏 Atari 以及 Dota2，并获得了非凡的表现，这些都是让计算机在不断的尝试中更新自己的行为准则，从而一步步学会如何下好围棋，如何操控游戏得到高分。

增强学习算法是针对实时决策的最好算法之一，具体而言，它研究一个 agent 如何在交互式环境中连续地做出一系列动作，以获得最大累积奖励的问题。一般来说，增强学习将问题建模成一个马尔可夫决策过程，并有以下组成部分：状态空间  $\mathcal{S}$ ，动作空间  $\mathcal{A}$ ，状态转移概率  $\mathcal{P}$  和一个回报函数  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ 。对于状态-动作空间内的任意轨迹  $s_1, a_1, s_2, a_2, \dots, s_T, a_T$ ，状态转移满足马尔可夫性质： $Pr(s_{t+1}|s_1, a_1, \dots, s_t, a_t) = Pr(s_{t+1}|s_t, a_t)$ 。当 agent 在状态  $s$  下选择了动作  $a$ ，它会收到环境返回的奖励值  $r$ （可能为负值）并进入一个新的状态  $s'$ 。在这个过程中，agent 选择动作的方式被称为策略。

### 2.1 基于值函数的增强学习

#### 2.1.1 深度模型 DQN

基于值函数的增强学习算法主要有以下几种：Q-Learning, Sarsa 和 DQN。这里的值函数由  $Q$  来表示，我们定义  $Q(s, a)$  为：对状态  $s$  下选择动作  $a$  的总回报的估计。在状态空间和动作空间有限时，一定存在某种策略是最优的，即该策略在每个状态下总能选择获得最多奖励（最终累积）的动作，我们将这样的最优策略的  $Q$  函数记为  $Q^*$ 。如果我们得到了  $Q^*$ ，那么策略的选取将会非常简单：在每个状态  $s$  下采用贪婪策略，即选择使  $Q^*$  最大的动作  $a$ 。因此我们的目标就是找到一个对  $Q^*$  的较好的估计，并采取贪婪策略执行动作。考虑到较后面的动作产生的奖励对最开始的动作的  $Q$  值影响较小，我们引入折扣因子  $\gamma$ ，那么就有

$$Q^*(s, a) = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \dots \quad (2-1)$$

其中， $\gamma$  控制了状态  $s$  下  $Q$  取决于未来的程度，同时反映了 agent 预判未来动作带来回报的能力，通常我们设置  $\gamma$  为一个接近但小于 1 的数。我们可以将式2-1写成递归的形式：

$$Q^*(s, a) = r_1 + \gamma(r_2 + \gamma r_3 + \gamma^2 r_4 + \dots) = r_0 + \gamma \max_a Q^*(s, a) \quad (2-2)$$

事实上，上式就是一个贝尔曼方程 (Bellman equation)。我们可以将式2-2改写成值函数的通式： $Q(s, a) = r + \gamma \max_a Q(s, a)$ 。Watkins 和 Dayan 证明了<sup>[5]</sup> 只要状态空间是有限的，且每个状态-动作对都能重复出现，值函数  $Q$  就可以收敛到  $Q^*$ 。Q-Learning 和 Sarsa 算法通过在环境中的大量摸索，可以将  $Q$  表（即值函数，一个以状态  $s$  为列，以动作  $a$  为行的表）中的值不断更新，直到收敛至  $Q^*$ 。然而通常情况下，状态空间可能不是有限的，此时  $Q$  表就是一张无限大的表，我们不可能建立这样

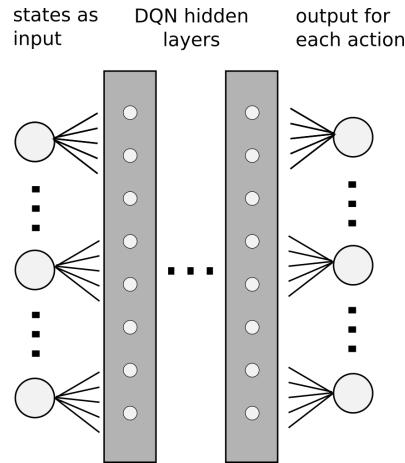


图 2-1 Deep Q-Network 结构图

一张表并更新其中的值。我们知道，理论上神经网络可以对任意函数进行估计，因此我们考虑用神经网络估计值函数  $Q$ 。该网络的输入是状态  $s$ ，输出是对每个动作  $a$  的值函数  $Q$  的估计。如图2-1所示，当神经网络有很多层时，这种算法就是 Deep Q-Network (DQN)。

### 2.1.2 经验回放

尽管我们可以用神经网络估计  $Q$  值，但是此时 Watkins 和 Dayan 的结论就不成立了。Volodymyr Mnih 等研究者指出：使用神经网络表示值函数  $Q$  是不稳定的<sup>[6]</sup>。由此他们引入了一些能够使训练稳定的非常有用的机制，其中就包含经验回放 (experience replay)。

在 agent 的学习过程中，它在某个状态  $s$  下执行了动作  $a$ ，收到即时奖励  $r$  并到达下一个状态  $s'$ 。注意到这种模式是不断重复的： $(s, a, r, s')$ ，若我们的优化过程是即时发生的（产生一个上述四元组就对网络参数更新一次）那么这种学习模式就是 online 的。由于神经网络不能直接对  $Q(s, a)$  赋值，我们采用梯度下降算法使  $Q(s, a)$  向  $r + \gamma \max_a Q(s', a)$  不断靠近，通过多次重复该过程，agent 就获得了越来越多的正确信息，并使它的策略向最优策略收敛。然而这种 online 的学习也存在着一些问题： $(s, a, r, s')$  四元组由于是连续产生的，它们彼此之间的相关性很高，因此可能会导致神经网络的过拟合以致不能有很好的泛化能力；对经验的使用较为低效，这是由于每个四元组只使用了一次。经验回放机制可以较好的解决这个问题：通过对这些状态转移的存储，在学习过程中我们可以从四元组库中随机选取 minibatch 并计算梯度以更新网络参数（加入经验回放机制的 DQN 如算法2-1所示）。

### 2.1.3 Double DQN

我们知道，DQN 由于是 Q-learning 的扩展而包含了  $\max_a Q^*(s, a)$ ，而在训练过程中的  $Q$  是神经网络在当前状态下对于值函数的预测，并不是真实的最优点，那么对  $Q$  取 max 就有可能包含了  $Q$  值的最大误差，直接用它对此时的神经网络更新参数就会带来 overestimate 的问题。Hado van Hasselt 等人在 2016 年通过引入 Double Q-learning<sup>[8]</sup> 提出了 Double DQN<sup>[9]</sup> 的概念，很好地解决了这个问题。他们将原来 DQN 中使用同一个神经网络选择、评价动作的过程分离，在  $t$  时刻下分别用参数为  $\theta_t$  的最新的 online 神经网络估计值函数  $Q$ 、用参数为  $\theta'_t$  的神经网络评价该策略的值（即计算 loss 以实现

---

### 算法 2-1 Deep Q-learning with Experience Replay<sup>[7]</sup>

---

```

Initialize replay memory  $\mathcal{D}$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode = 1,  $M$  do
    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$ 
    for  $t = 1, T$  do
        With probability  $\epsilon$  select a random action  $a_t$ 
        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ 
        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
        Set  $s_{t+1} = s_t, a_t, r_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$ 
        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$ 
        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$ 
    end for
end for

```

---

$\theta_t$  网络的梯度反向传播)，事实上我们每隔一段时间用  $\theta_t$  为  $\theta'_t$  更新参数。具体而言， $\theta_t$  神经网络的更新目标  $Y_t^{\text{DoubleQ}}$  是：

$$Y_t^{\text{DoubleQ}} \equiv r_t + \gamma \max_a Q(s_{t+1}, a; \theta'_t) \quad (2-3)$$

其中  $r_t$  是即时奖励， $\arg \max_a Q$  是由  $\theta_t$  网络对下一步动作的选择，这意味着和 Q-learning 相同，此处依然根据当前的网络参数估计、使用贪婪策略选择动作。

## 2.2 基于策略的增强学习

基于值函数的增强学习算法得到的是给定某状态下每种动作的  $Q$  值，agent 选择  $Q$  值最大的动作（贪婪策略下）。与基于值函数的算法不同，基于策略的增强学习不输出某状态下的动作的值，而是直接输出具体的某个动作，即直接对策略进行建模。基于策略的增强学习算法按输出分为两类：Stochastic 方法输出动作空间  $\mathcal{A}$  上的概率分布，Deterministic 方法直接输出动作  $a$  本身。

### 2.2.1 Stochastic 策略梯度方法

在此种方法中，agent 的动作由随机 (stochastic) 策略  $\pi(s)$  决定。随机策略的意思是它不输出一个单独的动作，而是输出各种动作的概率分布，所有动作的概率之和为 1。我们采用  $\pi(a|s)$  表示状态  $s$  下选择动作  $a$  的概率。简单而言，策略  $\pi$  并不最大化任何值，只是对于某个状态  $s$  给出所有可能动作的概率。策略  $\pi$  的价值函数定义为折扣回报的期望：

$$V(s) = E_{\pi(s)}[r + \gamma V(s)] \quad (2-4)$$

这里我们对状态  $s$  下的所有可能动作的  $r + \gamma V(s)$  取平均（而没有取最大值）。另外我们定义一个优势函数  $A(s, a)$ :  $A(s, a) = Q(s, a) - V(s)$ 。显然，这个函数的意义是：在状态  $s$  下采取动作  $a$  带来的回报比动作空间的回报均值多出了多少。最后我们用  $\rho$  来表示状态的分布，即在某种状态的概率，那么  $\rho^{s_0}$  就是环境初始状态的概率分布， $\rho^\pi$  则是在采取某策略  $\pi$  下的状态分布。与 Deep Q-Network 的想法类似，我们同样可以用神经网络（参数为  $\theta$ ）估计策略（即一个以状态  $s$  为输入、动作概率分布  $\pi_\theta$  为输出的函数）。agent 可以根据输出的动作概率选择下一步的动作。为了优化策略，我们需要定义衡量策略好坏的函数  $J(\pi)$ :

$$J(\pi) = E_{\rho^{s_0}}[V(s_0)] \quad (2-5)$$

$J(\pi)$  可以告诉我们对于所有可能的初始状态一个策略能够获得的折扣回报的平均值。根据策略梯度定理<sup>[10]</sup>，我们知道  $J(\pi)$  的梯度可以轻易由下面的式子计算得到：

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_S \rho^{\pi_\theta}(s) \int_A \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^{\pi_\theta}, a \in \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) A(s, a)] \end{aligned} \quad (2-6)$$

式2-6提供了神经网络参数更新的方向： $\nabla_\theta \log \pi_\theta(a|s)$  是状态  $s$  下选择动作  $a$  的概率增加的方向， $A(s, a)$  是选择该动作的优势，两项结合起来就是得到比平均回报的多的动作的概率增加，得到比平均回报少的动作的概率减小。尽管我们不能对每一个状态和动作计算策略梯度，我们提前产生大量的样本，这些样本服从与期望值相接近的概率分布（可证明 agent 在环境中运行时所遇到的状态和执行的动作是服从  $\rho^\pi$  和  $\pi(s)$  的无偏差样本）。当我们收集了 agent 的足够多的四元组  $(s, a, r, s)$  时，通过计算式2-6，我们就可以使用梯度下降算法更新神经网络参数，提升策略的表现。

## 2.2.2 Deterministic 策略梯度方法

Deterministic 策略梯度方法<sup>[11]</sup> 是 2014 年最先由 David Silver 等人提出的，他们考虑了一种决定性策略  $a = \mu_\theta(s)$ ，即以  $\theta$  为参数的神经网络以状态  $s$  为输入，直接输出动作  $a$ 。我们首先定义从  $t$  时刻开始的总折扣回报（total discounted reward） $r_t^\gamma$ :

$$r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k), 0 < \gamma < 1 \quad (2-7)$$

与 Stochastic 策略梯度方法相似，我们定义一个衡量策略表现的函数  $J(\mu_\theta) = \mathbb{E}[r_1^\gamma | \mu_\theta]$ （其意义就是在策略  $\mu_\theta$  下，从初始时刻  $t = 1$  开始的总折扣回报的期望）和折扣状态分布  $\rho^{\mu_\theta}(s)$ 。这样我们就有：

$$J(\mu_\theta) = \int_S \rho^{\mu_\theta}(s) r(s, \mu_\theta(s)) ds = \mathbb{E}_{s \sim \rho^{\mu_\theta}} [r(s, \mu_\theta(s))] \quad (2-8)$$

Silver 等人证明了决定性的策略梯度定理，打破了使决定性策略梯度算法变为可能：

$$\begin{aligned} \nabla_\theta J(\mu_\theta) &= \int_S \rho^{\mu_\theta}(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s, a) |_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s \sim \rho^{\mu_\theta}} [\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a) |_{a=\mu_\theta(s)}] \end{aligned} \quad (2-9)$$

通过使用式2-9，我们就能够像随机策略梯度算法一样更新神经网络参数，利用 agent 在环境中得到的大量  $(s, a, r, s)$  样本优化策略。

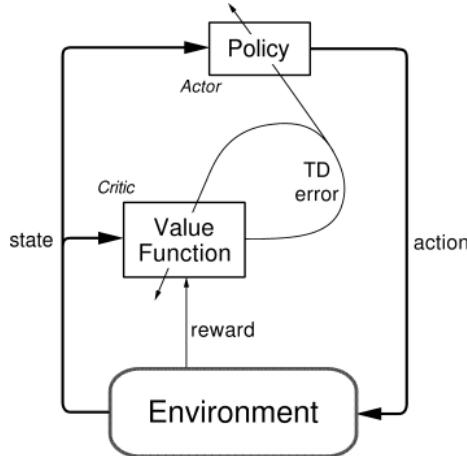


图 2-2 Actor-Critic 结构图

## 2.3 Actor-Critic 方法

回顾基于值函数和基于策略的增强学习算法，我们发现它们都是利用神经网络来估计某个函数。那么我们不妨将二者结合起来，使用相同的神经网络，既估计策略本身又估计策略的价值函数。这样做有很多好处：我们同时优化两个目标函数可以使学习过程更快更有效；一同优化两个目标可以相互起到规范化的作用，带来训练过程中更大的稳定性。

### 2.3.1 DPG 算法

David Silver 等人提出的 DPG (Deterministic Policy Gradient)<sup>[11]</sup> 算法都属于决定性的 actor-critic 算法。DPG 能够很好地解决连续动作空间上的问题，主要分为 on-policy 和 off-policy 两种。On-policy 算法中的 critic 负责估计动作的价值函数（这里使用一个可微的动作价值函数  $Q^\omega(s, a)$  替换真实动作价值函数  $Q^\mu(s, a)$ ，并使用 Sarsa 算法对其参数进行更新），actor 负责通过策略梯度算法调整决定性策略  $\mu_\theta(s)$ 。Off-policy 算法与 On-policy 算法的不同之处在于 critic 的参数更新算法使用了 Q-learning。Silver 等人通过计算得到了衡量策略表现的函数  $J$  的梯度<sup>[11]</sup>（如式2-10所示），由此 DPG 算法的 actor 就可以对折扣回报的期望应用链式法则以更新 actor 参数。

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{s \sim \rho^{\mu_\theta}} [\nabla_{\theta^\mu} Q(s, a | \theta^Q)|_{s=s_t, a=\mu(s_t | \theta^\mu)}] \\ &= \mathbb{E}_{s \sim \rho^{\mu_\theta}} [\nabla_a Q(s, a | \theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s | \theta^\mu)|_{s=s_t}] \end{aligned} \quad (2-10)$$

### 2.3.2 DDPG 算法

DDPG (Deep Deterministic Policy Gradient)<sup>[12]</sup> 算法是使用深度神经网络对 DPG (Deterministic Policy Gradient)<sup>[11]</sup> 中 Off-policy 算法的扩展，它与 DPG 一样都属于 actor-critic 算法。DDPG 一共包含四个相互关联的深度神经网络，其中两个是对 DPG 中的 actor  $\theta^\mu$  和 critic  $\theta^Q$  分别进行估计的神经网络，另外两个来自如式2-3中选择与评价分离机制（Double DQN 的思想）的两个目标网络  $\theta^{\mu'}$  和  $\theta^{Q'}$ 。训练过程中， $\theta^\mu$  和  $\theta^Q$  都采用如算法2-1中的经验回放机制，对记忆库中的  $(s_i, a_i, r_i, s_{i+1})$  四元组采样出大小为  $N$  的 minibatch，以避免时间上邻近的样本之间的相关性带来的网络过拟合情况。critic

网络的损失函数  $L$  的计算过程如算法2-2所示，首先用  $\theta^{\mu'}$  和  $\theta^{Q'}$  计算目标  $y$ ，再用当前的  $\theta^Q$  估计值函数  $Q$ ，再对二者作差。actor 网络参数的更新与 DPG 算法的式2-10相同。

目标网络  $\theta^{\mu'}$  和  $\theta^{Q'}$  的训练采用一种平滑的更新机制，通过设置  $\tau$ ，它们的参数缓慢地跟着  $\theta^{\mu}$  和  $\theta^Q$  更新： $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ ,  $\tau \ll 1$ 。这意味着目标网络参数的改动是被限制的，这能够提升学习过程的稳定性<sup>[12]</sup>。对于 agent 观察到的环境是用低维特征向量表示的场景，不同的特征向量可能具有不同的物理单位，因而使用一个网络对不同尺度的输入进行建模是困难的。Timothy P. Lillicrap 等人利用批标准化技术 (batch normalization<sup>[13]</sup>) 解决了这一问题。在低维的情形下，他们对状态输入、 $\theta_\mu$  网络的所有层、 $\theta_Q$  网络的所有层应用了批标准化，抑制了差异值偏差 (covariate shift) 在训练过程中的扩大，进而可以缓解训练中的梯度消失 (gradient vanishing) 问题。另外，连续的动作空间中很容易出现 agent 对环境探索不足的问题，但由于 DDPG 采用 Off-policy 机制，agent 对于环境的探索与学习算法是分离的，因而可以对策略  $\mu$  引入噪声，每个时刻的噪声从一个噪声随机过程  $\mathcal{N}$  中采样得到：

$$\mu'(s_t) = \mu(s_t | \theta_t^\mu) + \mathcal{N} \quad (2-11)$$

其中  $\mathcal{N}$  是一种适应于 agent 所处环境的 Ornstein-Uhlenbeck 过程<sup>[14]</sup>。该过程具有时间互相关性，适合于增强学习的场景。

## 2.4 本章小结

本章论述了目前在各场景下表现较好的增强学习算法，这些算法有些是针对离散的动作空间的，有些是针对连续的状态空间的，另外，本章提到的对于深度神经网络的使用、经验回放机制、actor-critic 算法等为后续章节建立本文提出的流量优化系统中用到的算法做以铺垫，在第四章我们会看到，基于数据中心网络负载均衡的需求特性和增强学习算法的优势，将二者结合是自然的，并会为均衡算法带来很大的性能提升。

## 算法 2-2 Deep Deterministic Policy Gradient<sup>[12]</sup>

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q$ ,  $\theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1, M **do**

    Initialize a random process  $\mathcal{N}$  for action exploration

    Receive initial observation state  $s_1$

**for** t = 1, T **do**

        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

        Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

**end for**

**end for**

## 第三章 5G Cloud RAN 中的负载均衡

### 3.1 无线移动通信系统基站的演进

在第一代与第二代无线移动通信中,蜂窝网络(cellular),或无线电接入网(Radio Access Networks,缩写 RAN),是由相互独立的基站组成的,每个基站配有电力、备用电源、空调系统、环境监测、回程链路等其他支持设施。每个基站内部含有射频信号发射单元,射频信号通过电缆从地面机柜传输至位于高处的基站天线。为了降低射频信号在电缆上的传输损耗,第三代移动通信采用了分布式基站系统。分布式基站将射频拉远单元(Remote Radio Head,缩写 RRH)与基带信号单元(Baseband Unit,缩写 BBU)分离,使RRH可以直接部署在基站塔顶距离天线很近的地方,并通过光纤的连接使两者间的距离变大(可达几百米或几千米)而损耗减小。建立在分布式基站架构的基础上,Cloud RAN为无线接入网引入了云技术,使很多BBU集中到一个BBU池。通过应用数据中心网络技术,Cloud RAN能够在BBU池中的BBU之间建立低功耗、高可靠、低延迟、高带宽的连接。它应用了开放的平台与实时虚拟化技术以实现动态共享的资源分配并支持多厂商、多种技术并存的环境。在这种技术架构下,由于BBU池覆盖的范围比原来的一个基站所能覆盖的范围大得多(即一个用户从一个RRH切换到另一个RRH时,他前后连接的两个BBU在同一个BBU池中)<sup>[1]</sup>,蜂窝系统中流量分布不均匀的问题可以通过在BBU池中部署灵活的、动态自适应的负载均衡算法得以解决。

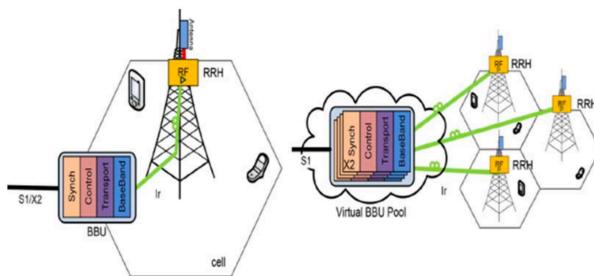


图 3-1 RAN (左) 与 Cloud RAN (右)<sup>[15]</sup>

### 3.2 5G 时代的数据中心网络

近年来,随着越来越多的企业和组织将其服务转向云端、云服务的需求日益增长,人们建立了许多规模庞大的数据中心。数据中心的大量服务器通过如Fat-Tree<sup>[16]</sup>、VL2<sup>[17]</sup>、CamCube<sup>[18]</sup>、DCell<sup>[19]</sup>、BCube<sup>[20]</sup>等可横向拓展的模型连接在一起,为云服务提供大量的计算资源。

#### 3.2.1 网络架构

按网络架构的不同,数据中心可分为以交换机为中心、以服务器为中心和混合型三种架构。在以交换机为中心的架构中,交换机被用于提供服务器之间的多条通路连接以及数据包的转发;在以

服务器为中心的架构中，服务器既提供计算功能还扮演着交换机的角色与其他服务器相连；混合型架构使用服务器与交换机共同实现数据的转发，并通常在服务器间提供多条不等长的路径。下面我们将对每种架构举例说明：

- **Fat-Tree**: Fat-Tree 是一种以交换机为中心的多层结构，其拓扑从下到上分为三层：edge，aggregation 和 core。如图3-2所示，对于一个  $k$  叉树 ( $k$ -ary)，Fat-Tree 共包含  $k$  个 pod，每个 pod 有两层交换机，每层有  $\frac{k}{2}$  个。每个交换机共有  $k$  个端口。在 edge 层，交换机的  $\frac{k}{2}$  个端口连接着  $\frac{k}{2}$  台服务器，剩余的  $\frac{k}{2}$  个端口连接 aggregation 层的交换机。在 core 层共有  $\frac{k^2}{4}$  个核心交换机，每个核心交换机有一个端口连接着  $k$  个 pod 中的一个。这种架构能够极大地减少部署交换机的开销，但也会导致较多的互连开销。

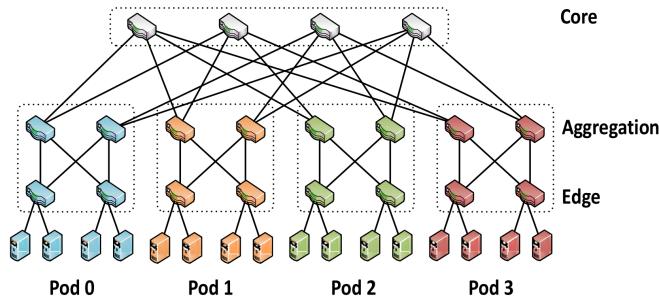


图 3-2 Fat-Tree 架构 (图中为  $k = 4$  的情形)<sup>[16]</sup>

- **CamCube**: CamCube 是一种以服务器为中心的空间网状结构。图3-3展示的是含有 27 个服务器的 3D-Torus 结构。P. Costa 在论文中提出了一种基于键 (key) 的网络，通过使用 CamCube 的 API，能够充分利用空间网格结构的优越性，节省交换机和路由器的开销。同时由于服务器的散热开销小于交换机，这种结构比较节能。CamCube 的主要缺点是对于非键值的应用，数据转发路径可能会很长，路由算法的复杂度可能会很高<sup>[21]</sup>。

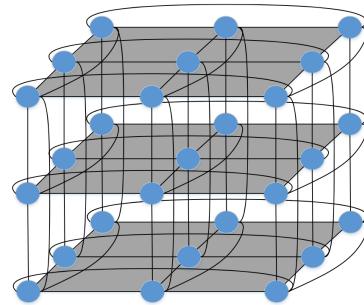


图 3-3 CamCube 架构，蓝点为服务器<sup>[18]</sup>

- **BCube**: BCube 是一种使用服务器与交换机共同实现数据转发的混合型架构。其结构可以递归地定义： $BCube_0$  由连接在一个  $n$  端口交换机的  $n$  台服务器组成； $BCube_1$  由  $n$  个  $BCube_0$  组成，它们与  $n$  个  $n$  端口交换机相连； $BCube_k$  ( $k \geq 1$ ) 由  $n$  个  $BCube_{k-1}$  组成，它们与  $n^k$  个  $n$

端口交换机相连。 $n=4$  的情形如图3-4所示。BCube 架构可以以较低开销提供很高的对分带宽 (bisection bandwidth)，但其缺点在于它的递归深度受限于服务器的网络接口控制器 (NIC)。

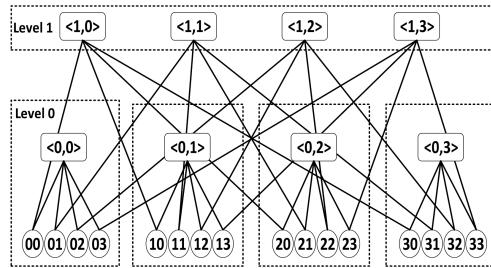


图 3-4 BCube 架构，图中方框代表交换机，圆圈代表服务器<sup>[20]</sup>

### 3.2.2 流量特征

数据中心网络中运行着很多不同的应用，从对低延迟有较高要求的网页搜索任务，到类似数据库更新的这种有着高吞吐量要求的任务。通过对于 Netflow<sup>[22]</sup> 和 SNMP<sup>[17, 23, 24]</sup> 数据的分析可以看到数据中心的流量特征。首先，数据中心的流量目的地与拥有者有关：在云数据中心，近 80% 的流量属于其机架的内部转发；学校和私营企业的数据中心，40%-90% 的流量穿越在不同的机架间。这是由于云数据中心的应用被很好地部署在各机架上以避免机架间的数据传输。第二，数据中心内的大部分数据流都很小（几 KB），并且这些数据大部分是分布式系统中的元数据（meta-data）请求。只有少量数据流的大小是 100MB 到 1GB，几个 GB 的数据流极少。第三，尽管数据中心的大部分数据流都较小，总体数据中大部分还是由大数据流构成，即大数据流的数目少，却占了数据中主要的部分。最后，源头为数据中心机架上的数据本质上是类似于受开关函数控制的流量，开关函数是符合重尾（heavy-tailed）分布的。然而总体的流量分布是随机的、不可预测的。

对于数据中心的拥塞问题，从直觉来说，丢包率与链路的利用率成正相关，但最近的研究表明，二者之间没有关联，是某些流量模式导致了严重的丢包率<sup>[25]</sup>。另外，链路的利用率在更高层中增长迅速<sup>[23, 24]</sup>，例如：核心层中的链路利用率经常达到 70%，然而核心链路的丢包率却是惊人的少<sup>[23]</sup>。研究表明，数据包的丢失主要由每一层的微小流量突发造成，并且边缘层中的突发数据比核心层中更多。另外一个特征是，由于拥塞问题，一小部分链路总是比数据中心的其他链路经历更多的丢包，这种情况可以通过部署适当的负载均衡算法得以解决。通常，数据中心会部署备用设备和多余的链路以保证高可靠性，但设备或者链路的损坏在数据中心依旧时有发生，其可靠性与网络的拓扑结构相关。我们希望负载均衡算法能够对网络组成部分的损坏情况做出快速反应。我们注意到有关网络组成部分的损坏的一些特征：1) 即便是价格低廉的路由器也非常可靠；2) 软件的错误既可以导致设备损坏也可能导致链路的断开；3) 链路断开会导致大量的包丢失，并极大影响 TCP 的性能；4) 网络中的冗余链路可以减缓链路断开的影响。然而，当前的流量调度、负载均衡算法难以动态利用数据中心的冗余链路。这是因为当前的算法主要借鉴了静态的较为稳定的网络架构中成熟的均衡算法。

(如等价多路径算法<sup>1</sup>和多协议标签交换<sup>2</sup>)。一个好的均衡算法应当考虑到上述数据中心网络的特征。具体来说，该机制应当以微秒级的速度传输大的数据流，并以更快的速度传输小数据流；应当能动态的对链路的断开进行实时调整，将数据流引向拥塞情况不严重的链路上。

### 3.3 负载均衡问题

负载均衡是人们长期以来关注的课题。它存在于各个场景下：基于因特网的服务<sup>[26]</sup>（域名系统委派<sup>3</sup>中的负载均衡；客户端或服务器端的负载均衡等）、数据中心网络<sup>[27]</sup>（服务器之间存在多条路径导致的流量分布不均等）、无线通信系统（处理冗余的通信链路等）。在不同的场景下，鉴于所期望实现的不同目标，研究者们设计了各种负载均衡算法。从研究这些算法思想的过程中，可以汲取一些思路，并学习如何在特定情况下进行仿真实验以验证算法的有效性。

传统蜂窝网络中的负载均衡算法被广泛研究，如小区呼吸技术<sup>[2]</sup>已被应用于第二代和第三代移动通信网络中。由 X. Lin 和 S. Wang 提出的 Cloud RAN 中的高效 RRH 切换机制在考虑系统能效条件下<sup>[3]</sup>实现了负载均衡。C. Ran 和 S. Wang 等人从另一个角度提出了 Cloud RAN 中的最佳负载均衡算法：他们周期性地监控衡量均衡度的公平指数，当其低于特定阈值时，则重新设计每个小区覆盖的区域以实现系统的负载均衡<sup>[1]</sup>。C. Tsai 和 M. Moh 在 Cloud RAN 架构下比较了 8 种均以降低物联网<sup>4</sup>设备通信延迟为目标<sup>[15]</sup>的负载均衡算法。由于本文主要考量 5G 通信系统内数据中心网络中的负载均衡问题，这里对传统无线通信系统中的相关问题不做过多的讨论。

### 3.4 数据中心网络中的负载均衡算法

我们知道，在 5G 无线通信系统中，数据中心网络是基站技术不可或缺的关键组成部分，这一小节主要研究数据中心网络中的负载均衡算法。数据中心网络通常为各种应用提供对分带宽 (bisection bandwidth) 而采用可横向拓展的模型<sup>[28]</sup>，大量数据在网络中的服务器间通过多条路径传输。基于前面提到的数据中心网络的数据流量特性，本节首先讨论数据中心网络负载均衡的定义、特征、目标，之后比较数据中心网络与广域网<sup>5</sup>中负载均衡的不同，最后引出本文主要场景下的负载均衡算法。

#### 3.4.1 定义、目标与主要步骤

容量受限的网络中的流量调度可以建模成一个被深入研究过的问题：多商品流问题<sup>6</sup>。数据中心网络的流量调度问题就是找到如下 MCF 问题的最优解：

- $\text{maximize} : \sum_i U_i(x_i)$
- $\sum_{u:(s,u) \in E} f_{u,v}^{s,d} = \sum_{w:(v,w)}, \forall v, s, d$
- $\sum_{u:(s,u) \in E} f_{s,u}^{s,d} = \sum_{i:s \rightarrow d} x_i, \forall s, d$

<sup>1</sup>Equal-Cost Multipath Routing，缩写为 ECMP

<sup>2</sup>Multi-Protocol Label Switching，缩写为 MPLS

<sup>3</sup>Domain Name System (DNS) delegation

<sup>4</sup>Internet of Things，缩写为 IoT

<sup>5</sup>Wide Area Network，缩写为 WAN

<sup>6</sup>Multi-Commodity Flow，缩写为 MCF

- $\sum_{s,d} f_{u,v}^{s,d} \leq c_{u,v}, \forall (u,v) \in E$

其中，目标函数  $\sum_i U_i(x_i)$  代表“商品的效用”，它取决于流的发送速率。 $x_i$  是流  $i$  的发送速率。 $f_{s,u}^{s,d}$  是链路  $(u,v)$  上的流速率， $(s,d)$  代表了流的源地址和目的地址。负载均衡问题本质上是对流进行链路分配，只依靠负载均衡不能保证 MCF 问题的最优解，这是因为最优的策略与流的发送速率有关。通常情况下，发送速率由端系统的传输控制协议（如 TCP）决定。在一个具有固定发送速率的对称网络<sup>1</sup>中，负载均衡算法等价于对最大链路利用率的最小化问题的求解： $\min \max_{(u,v) \in E} \frac{\sum_{s,d} f_{u,v}^{s,d}}{c_{u,v}}$ 。不过，由于 MCF 问题是 NP-hard 问题，难以在微秒级的时间内得到最优解，负载均衡问题需要依靠先验性的算法解决。

Jiao Zhang 等人<sup>[28]</sup>提出，数据中心网络负载均衡的主要目标可以分为四类：高性能、可扩展性、鲁棒性、能效。

- 高性能：吞吐量和延迟是高性能目标的两个要求。如前文所述，数据中心内的大部分数据来自较大的流，因此算法的表现主要取决于均衡机制对大流的处理上。不同于网络运营商的考虑（他们更关注于吞吐量），数据中心的用户主要关注低延迟的问题。Amazon 的调研指出，每 100ms 的延迟就会导致销量下降 1%。延迟的主要因素是不平衡的流量分配，即在链路中较小的流可能会被较大的流堵住，而致使流完成时间<sup>2</sup>的急剧增加。
- 可扩展性：随着数据中心运行起越来越多的应用，其规模近年来不断扩大。由于大多数负载均衡机制要求存储链路利用信息，如 CONGA 使用叶节点交换机维持一个每条路径的拥塞情况表，当数据中心规模增加时，可能存在用尽边缘交换机内存的风险。另外如果均衡机制是针对某种架构设计的，如 CONGA 是为 2 层拓扑结构设计的，那么将其应用到三层结构中，拥塞信息可能就无法准确地收集。
- 鲁棒性：即便数据中心使用的设备都较为可靠，链路的断开与交换机的损坏依旧时有发生；经常性的软件或硬件更新会导致网络拓扑结构的改变，这会导致端系统间的链路发生改变。上述问题均是负载均衡算法需要面临并解决的问题。目前主要有两种解决方案：一种是使用一个中心控制器周期性地监控网络状态，当错误发生或网络结构改变时，中心控制器改变负载均衡算法以做出合适的调整；另一种在交换机处分布式地监视端口信息，当错误发生时或结构变化时将端口设置成不可用状态。
- 能效：数据中心的能效问题日益受到关注，由于数据中心提供大量的对分带宽和多条备用链路，这些冗余会造成大量的能量损耗。尽管相关研究指出<sup>[29]</sup> 最近几年数据中心的能耗增长逐渐放缓，维持一个数据中心的运行的成本也是极其惊人的。主要有两种途径减少能量损耗：根据工作负载的特性，关闭没在使用的设备（如动态能量管理机制 DPM），以及降低设备的性能（如动态电压频率调整机制 DVFS）。

一般而言，负载均衡机制可以分为两个主要步骤：收集拥塞信息和选路。

### (1) 收集拥塞信息

负载均衡机制需要根据拥塞信息（如链路利用率、显示拥塞指示<sup>3</sup>等）来确定数据流的路径，也

<sup>1</sup> 对称网络是指网络中对于所有的源端-目的端对  $(s,d)$ ， $s$  和  $d$  之间最短路径上的所有到源端  $s$  距离相同的交换机到目的端  $d$  都有相同的容量

<sup>2</sup> Flow Completion Time，简称为 FCT

<sup>3</sup> Explicit Congestion Notification，缩写为 ECN

就是说负载均衡的一个关键步骤是要选择一种或多种代表拥塞信息的衡量标准，并将它传送至为数据流做选路决定的实体。收集拥塞信息的方法有以下三类：

- 基于 TCP 的采集：TCP 将数据包的丢失视为网络拥塞的信号，并且当检测到拥塞情况时会减小拥塞窗口。基于 TCP 的采集方法（如 MPTCP<sup>[30]</sup>, CLOVE<sup>[31]</sup> 等）利用与 TCP 或 TCP 变种相似的信号检测拥塞情况，ECN 就是负载均衡中常用的一种信号。
- 发送速率：数据流的发送速率是可以由交换机和端系统直接检测到的数据。基于各个数据流已发送的数据量，可以在时间上控制不同数据流的发送，以此实现所有数据流的平均发送速率。由此，基于发送速率的方法（如 Hedera<sup>[32]</sup>, CONGA<sup>[27]</sup>, LocalFlow<sup>[33]</sup>, Fastpass<sup>[34]</sup>）就根据已发送的数据量判断拥塞情况，通常在软件定义网络<sup>1</sup>中使用较多。这是因为其中的 OpenFlow 交换机自动地记录已发送的数据量，且控制单元可以随时访问这个量。
- 交换机的队列长度：交换机缓存的利用率是流过该交换机的数据流的直接反映。基于交换机队列长度的方法（如 Expeditus<sup>[35]</sup>, DRILL<sup>[36]</sup>）利用路径上交换机的队列长度代表链路的利用率、检测拥塞情况。这种方式对于内网的情形较为方便。

## （2）选路

在获取数据中心网络的拥塞信息后，负载均衡机制需要对到达的数据流分配不同的路径以实现各路负载的平衡。在不同的场景下，负载均衡机制的目标与算法略有不同，以下列举了四种选路的方式：

- 拥塞最少：最理想的选路方式是把每个数据流分配到最不拥塞的链路上。因此这种方式可以通过收集每条路径的拥塞信息并将数据流传递至拥塞最少的路径上。这种方式（如 CONGA<sup>[27]</sup>, Expeditus<sup>[35]</sup>）的性能与链路利用率的响应时间与准确性密切相关。
- 较少的拥塞：由于很难实时采集链路利用率的准确值，另一种方式是将数据流从拥塞情况严重的路径转移到拥塞情况较轻的路径上去，同时减少了记录所有路径信息的开销（如 MPTCP<sup>[30]</sup>, DRILL<sup>[36]</sup>, Hedera<sup>[32]</sup>）。
- 轮询调度：原始的轮询调度是将数据流一个一个地分配至所有可行的路径。由于它对数据流和链路信息不可知，在重负载情况下很容易产生流的碰撞。因此可以根据数据流的长度和链路利用率采用加权轮询的方式均衡分配，此种方法（如 CLOVE<sup>[31]</sup>, Presto<sup>[37]</sup>）的性能受到链路权重的影响。
- 集中式分配：以上提到的三种方法，对于每个数据流的选路过程彼此间相互独立，这可能会导致局部最优的情况。因此集中式分配的方式对数据流进行整合式的分配以达到全局最优（如 Fastpass<sup>[34]</sup>, LocalFlow<sup>[33]</sup>）。

### 3.4.2 数据中心与 WAN 中负载均衡的比较

在很多方面，数据中心与 WAN 中的负载均衡问题是相近的，比如问题的定义，比如二者均是对多条链路分配网络流量等，然而二者之间有很多的不同点：

- WAN 中的网络拓扑结构通常是不规则的，而数据中心的网络结构通常是对称的，即在每对端系统间的链路是 equal-cost 的，这提供了高对分带宽并减少了计算链路的复杂性。

<sup>1</sup>Software Defined Network，缩写为 SDN

- WAN 可能包含不同的链路容量和硬件种类，因此流量调度机制就要求将不同结点的限制考虑进去。对比之下，在数据中心的一层中，交换机常常是一种类型的，链路容量也完全相同，这极大地简化了问题。
- WAN 中服务与调度的变化通常要花数小时或者数天，而在具有高速链路的数据中心，较小的流只需要数十或者数百毫秒进行传输，数据突发所导致的拥塞可能在毫秒级的时间内就消失了，这就要求流量调度算法足够及时地处理拥塞情况。
- WAN 通常没有中心控制器，全局的网络信息难以被收集、链路断开难以被解决，而数据中心采用的一些可编程架构，如 SDN，都提供了中心控制器，以对网络的全局信息进行收集与处理，因此可以提供中心化的实时流量调度。

通过比较 WAN 和数据中心网络，可以更清晰地看到数据中心网络负载均衡的特殊环境。具体来说，数据中心的网络结构更有规则，数据流量变化更快，中心化的控制比较常见，这为负载均衡算法留出了更多的设计空间。

### 3.4.3 近年来提出的负载均衡算法

本小节从分布式和集中式的分类角度介绍近年来提出的负载均衡算法。这两类机制的主要区别在于是否有一个中心控制器采集拥塞信息并实时为数据流选路。对于集中式机制，中心控制器可以容易地获得全局的链路利用信息，但通常都需要额外的开销向控制器传送信息，此外，集中式算法的时间粒度较分布式而言比较粗糙，大部分集中式的控制器都得不到最及时的信息以处理数据中心网络中的数据突发。由于分布式系统中的选路决策发生在端系统或交换机处，这样就可以足够及时地应对数据突发。但通常分布式的算法是为对称网络设计的，当拓扑结构有所改变时（如链路的断开、交换机的崩溃）就很难应对。

#### (1) 集中式负载均衡

- Hedera<sup>[32]</sup>: Hedera 是一个主要针对大数据流处理的集中式算法。由于 ECMP 不能很好地处理大数据流，甚至会造成流的碰撞，Hedera 提出了一种估计大数据流并中心化地对其分配路径，以最小的开销使网络利用率最大化。具体而言，Hedera 的第一步是在边缘交换机处检测大数据流，第二步是根据分配算法为这些大数据流选路，最后，控制器在相关的交换机上部署路径信息。在检测数据流的阶段，中心控制器在边缘交换机处周期性地监测流速率，当一个数据流的速率大于链路容量的 10%，则认为它是一个大数据流。大数据流的选路有两种方法：全局优先匹配和模拟退火算法。全局优先匹配为大数据流选择一条能够容纳该数据流的链路，这种方式适合工作在负载较轻的网络中，当网络负载增加、链路饱和时则不能保证所有的数据流都满足预设条件。模拟退火算法将目的地址相同的大数据流集合起来，并为它们分配一个核心交换机，这可以极大地减小控制器的搜索空间。这种算法每次处理一批数据流，采用迭代机制寻找最优的分配方式。在选路之后，控制器将相应的转发规则通过 OpenFlow 协议部署在相关的交换机上。当链路发生断开情况，Hedera 通过使用可扩展、可容错的 2 层路由转发协议 PortLand<sup>[38]</sup> 解决这一问题。仿真结果显示在平均对分带宽上 Hedera 比 ECMP 表现更好，且其头部开销是可接受的，流速率估计过程的复杂度与端系统和大数据流的数量呈正相关。全局优先匹配与模拟退火算法的运行时间为数十至数百毫秒，这符合数据中心对于处理速度的要求。然而，并不是所有的大数据流都会达到速率限制，Hedera 中的大数据流检测方

法可能是不准确的。另外，除非当一个大数据流达到速率限制，它才会被分配一个合适的路径，这会导致一些数据包的安排不合理。至于控制过程，尽管数据采集与大数据流调度可以很快地完成，仿真过程中整个操作流程花了 5 秒，对于数据中心网络，将其减小到 200 毫秒是比较合理的，但这会增加额外的开销，实际中部署起来也较难实现。

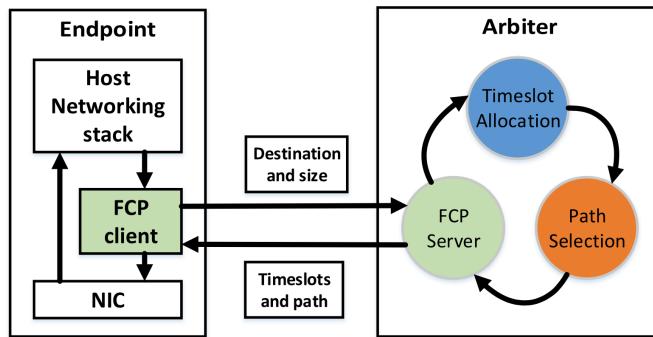


图 3–5 Fastpass 结构图

- Fastpass<sup>[34]</sup>: Fastpass 是一个混合型的负载均衡机制，它将传输控制和流量的调度相结合以达到数据中心网络的“零队列”(zero-queuing)。Fastpass 控制每个数据包的发送时间和传输路径，其主要的组成部分是时间片分配和选路机制<sup>3–5</sup>。其中，Fastpass 控制协议（英文简称 FCP）负责传输数据交互的需求和时间片的分配决定。时间片分配这一部分控制着每个数据包发送的时间，以实现最小化 FCT 的目标，它是流水线的方式实现的：在一个时间片内以尽力服务的模式实现一部分数据交互的需求，另一部分的需求要等到下一个时间片处理。之后，选路算法为已分配时间片的数据包指定零队列的发送路径，该算法采用图论中的边染色算法（以一个二分图作为输入，为图的每条边染色，使得同一结点上的两条边颜色不相同）。针对网络结构的变化，Fastpass 采用 replication 策略以应对链路断开或者控制器的出错。具体而言，端系统向一个错误隔离机制报告数据包的丢失，该机制继续向控制器上报，此时控制器就会避免向这个出错的部分分配流量。当控制器出现问题时，备用的控制器就会取代之前的控制器继续流量调度。与<sup>[34]</sup> 中提到的一个采用 TCP 协议的基准算法相比较，Fastpass 可以明显减少队列长度和 FCT，并达到较高的吞吐量。Fastpass 的主要弱点是部署较难，可扩展性不好。由于中心控制器需要为每个数据包决定时间片与发送路径，Fastpass 的性能主要由控制器的处理速度决定。单个控制器可以处理数百或数千个端系统的均衡问题，但对于拥有更多端系统的大型数据中心网络，单个控制器的计算资源可能会成为瓶颈，因此可能需要多个控制单元的合作。多控制器的架构以及如何保持数据的一致性亦是目前研究的热点<sup>[39, 40]</sup>。

**(2) 分布式负载均衡** 我们共列举七种主流的分布式负载均衡算法，其中前四种是已知拥塞信息的均衡机制，它们均通过采集拥塞信息做出基于全局的流量调度；后三种是拥塞信息未知条件下的均衡机制。

- MPTCP<sup>[30]</sup>: 多路径 TCP（英文简称 MPTCP）是一种根据拥塞信息做出相应均衡决策的机制。它利用端系统之间的多条路径并设置多个子链接来充分利用带宽、提高吞吐量。MPTCP 使用 SYN 来确定使用该机制，在成功建立 TCP 连接后，客户端从 SYN 消息得知服务器具有的额

外 IP 地址或 TCP 端口，这样就可以使用不同 IP 地址或端口建立额外的连接。在传输过程中，每个 MPTCP 子流有其独特的序列号并维持一个拥塞窗口。此外，MPTCP 修改了 TCP 中的一些特征，以使多个子流同时工作。<sup>[30]</sup> 通过仿真实验得到，与 TCP 相比，MPTCP 极大提升了数据中心的吞吐量。但 MPTCP 有两个主要问题：1) 数据中心网络的大部分数据流都比较小，将它们分解成子流以使用额外的 TCP 连接实际上增加了 FCT。另外在某些情境下，使用多条链路（与使用单条路径相比）只能带来较少或有限的增益。2) MPTCP 在为子流选路上依赖于 ECMP 和其他路由机制，如果某些子流被分配至拥塞的路径上，其效果就会下降。FMTCP<sup>[41]</sup> 和 A-MPTCP<sup>[42]</sup> 是两个针对 MPTCP 性能提升的机制。

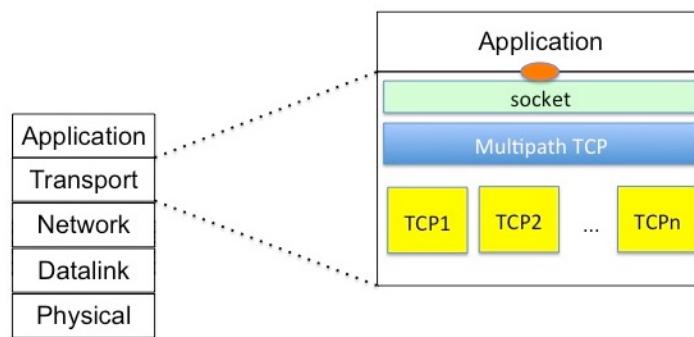


图 3-6 MPTCP 在协议栈中的位置

- CONGA<sup>[27]</sup>: CONGA 是为两层叶脊 (leaf-spine) 拓扑网络设计的最佳负载均衡机制。其设计者指出：集中式机制对于拥塞情况的反应太慢、单纯的局部机制又很难收集到全局拥塞信息，因此他们设计了一种能够在叶结点交换机上收集全局拥塞信息的分布式算法，他们利用 VxLAN<sup>[43]</sup> 虚拟化技术承载链路利用率信息，使用 VxLAN 头域中额外的四个标记，将拥塞信息通过数据包从目的叶结点交换机传送至源叶结点交换机。为了记录链路的拥塞信息，每个叶结点交换机维护目的地址拥塞表。每当反馈信息到达交换机就更新一次该表。此外，CONGA 将数据流分成 flowlet 以实现较少数据包重排的细粒度负载均衡目标。一个 flowlet 是由叶结点交换机监测判断的，即当两个连续数据包之间的时间空隙超过了路径导致的最大延迟时，则判断前后数据包为两个 flowlet，否则为同一个 flowlet。flowlet 由叶结点交换机维护的拥塞信息表指定一条拥塞最少的路径进行传输。CONGA 与 ECMP 和 MPTCP 进行了性能比较，结果显示 CONGA 实现了比 ECMP 和 MPTCP 更少的 FCT（在企业负载和数据挖掘负载任务上有或无链路断开故障），且在 incast 场景下 CONGA 的吞吐量是 MPTCP 的 2-8 倍。然而 CONGA 也有不足之处：1) 首先它要求对叶结点交换机和 VxLAN 的头部进行修改，这对于目前的数据中心实现起来有一定困难；2) CONGA 还需要花几个往返时延<sup>1</sup>（数十至数百微秒）等待拥塞信息的反馈，对于数据中心而言，可能这段时间内拥塞已经消失了；3) 当多个流同时到达交换机处，根据拥塞信息表交换机可能为几个流选择同一条路径，从而造成拥塞。
- CLOVE<sup>[31]</sup>: 一般而言，集中式的均衡机制对于延迟敏感的短流反应较慢，而分布式的负载均

<sup>1</sup>Round-Trip Time，简称为 RTT

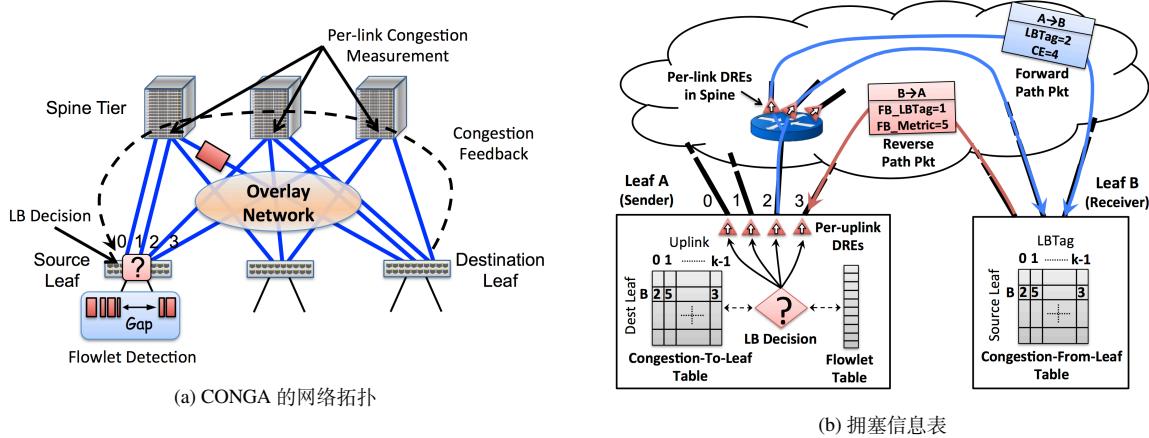


图 3-7 CONGA 的主要结构

衡常常要修改交换机或端系统的栈。因此，CLOVE 选择在交换机的软件上进行修改，以避免对硬件和端系统做改动。CLOVE 在物理层上使用标准的 ECMP，通过在交换机软件中修改数据包的头部以指导交换机如何传送数据。CLOVE 包含三个主要部分，1) 路径发现：CLOVE 中的交换机软件通过改变源端口地址的方式同时发射许多探头，之后源监视器就可以选择源地址；2) flowlet：CLOVE 采用 flowlet 以避免数据包的重排并在交换机处检测 flowlet；3) 已知拥塞的路由：CLOVE 使用加权轮询机制对 flowlet 进行均衡。有两种方式得到路径权重：第一种是采用 ECN 检测拥塞、计算权重的 CLOVE-ECN；第二种是使用带内网络遥测技术<sup>1</sup>精确收集链路利用率的 CLOVE-INT。仿真结果显示 CLOVE-ECN 能够在中等或较重负载下显著提升 FCT 的平均表现（对比 ECMP）；CLOVE-INT 基本与 CONGA 的表现相似（不论在对称或在非对称的网络中）。但由于并非所有数据中心网络都使用虚拟化技术，在未采用虚拟化技术的场景下采用 CLOVE 需要进一步的探究。

- Expeditus<sup>[35]</sup>：在例如 fat-tree<sup>[16]</sup> 和 VL2<sup>[17]</sup> 等基于 Clos 的架构中，在叶结点交换机处维持拥塞信息通常需要大量的内存，因此，实际上很难实时采集全局拥塞信息。为了解决这个问题，Expeditus 针对 fat-tree 架构提出了两种设计：一跳 (one-hop) 拥塞信息采集和两阶段选路 (two-stage path selection) 过程。1) 拥塞信息采集：每个交换机既要监视上行（从自身向上游交换机）的链路利用率也要监视下行（从上游交换机向自身）的链路利用率。下行信息通过数据包的头部由出站端口 (egress ports) 轮询缓存占用率得到。具体而言，当一个数据包从一个核心交换机发出，其包的头部记录了出站端口的速率与核心交换机的 IP 地址。之后，下游的聚合交换机接收该数据包，获取核心交换机的 IP 地址，对相应链路利用率进行更新。之后它将核心交换机的 IP 地址替换为自己的 IP 地址，轮询出站端口的占用率，更新收包的信息并将它继续向下转发，最后，边缘交换机采用与聚合交换机类似的处理方式将数据包转发给端系统。由下行拥塞信息表记录的占用率若在一段时间内没有更新就会被置为 0。2) 选路：两阶段选路过程为数据流选择源端到目的端的转发路径和目的端到源端的反向路径。初始状态下，由于交换机处没有拥塞信息，SYN 和 ACK-SYN 包使用 ECMP 进行路由转发。在第一阶段，SYN

<sup>1</sup>In-band Network Telemetry，简称为 INT

包会携带反向路径的拥塞信息至目的端的边缘交换机，因此在接收到 SYN 包后，边缘交换机就可以选择到达聚合交换机的最不拥塞的反向路径。在第二阶段，SYN-ACK 包会携带前向转发路径的拥塞信息到源端的边缘交换机。之后，当 SYN-ACK 包到达源端聚合交换机时，交换机会决定到达核心交换机的最不拥塞的转发路径。由于目的端的聚合交换机已在第一阶段确定下来，由于 fat-tree 的结构源端的聚合交换机也随之确定，源端的边缘交换机不需要选择转发路径的源端聚合交换机。Expeditus 在叶脊和 fat-tree 两种架构下与 ECMP 和 CONGA 做了性能比较。仿真结果显示在不同场景下，Eepeditus 的 FCT 和末端延迟比 ECMP 和 CONGA 降低了 25% 到 30%，但是 Expeditus 对数据中心的网络拓扑有着较严格的要求。

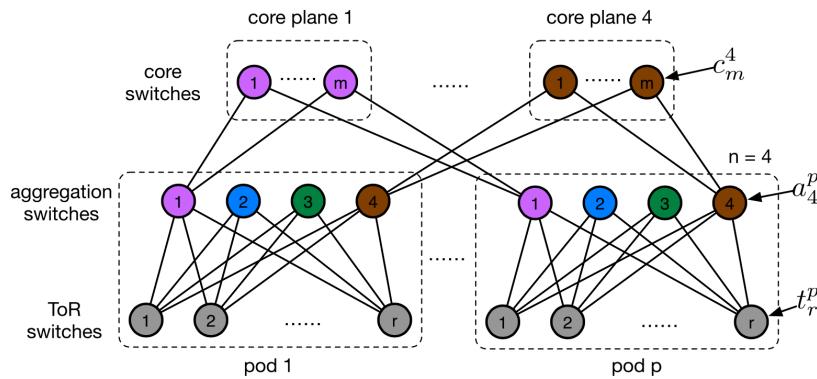


图 3-8 Expeditus 的网络结构

- **Presto<sup>[37]</sup>**: Presto 同样采用 ECMP 机制实现非对称拓扑结构中端系统之间数据流的最优均衡。Presto 的设计者发现，在对称网络中，如果所有的数据流都是很小的流，ECMP 的表现是接近最优的，当数据流的大小分布的标准差变小时，ECMP 造成的不均衡情况就会有所减轻。因此 Presto 将数据流分成等大的小数据流 flowcell，并将它们均等地在网络中用 ECMP 分发。Presto 采用 TCP 数据包分片的最大值<sup>1</sup>64KB 作为 flowcell 的默认大小，实现了网络中细粒度的负载均衡。Presto 的接收端采用通用接收报文<sup>2</sup>将乱序的包合并为段并将它们组装起来。与 ECMP 和 MPTCP 相比，Presto 达到了更大的吞吐量和更少的 FCT。它可以显著地减少链路断开造成的影响，并在采用加权调度算法时可以达到合理的吞吐量。但是 Presto 在处理非对称网络时不是最优的解决方案，即便加权算法可以将一些负载从拥塞的路径上调度至其他路径。
- **LocalFlow<sup>[33]</sup>**: LocalFlow 是一个实现在 OpenFlow 交换机上的本地算法，它提出对每个数据流调度的算法是次最优的，这是因为大数据流不能很好地利用数据中心网络多路径这一特征。通过对每个数据流空间上进行分离并指定合适的路径，LocalFlow 在对称网络中表现得很好，已经在大型数据中心得以应用。LocalFlow 调度过程是有四个主要步骤的循环体：1) 对每个数据流测量速率（使用字节计数器）；2) 同一目的地址的数据流被聚合成一个大数据流，之后大数据流被分成 L 个 bin，其中 L 是输出端口的个数，参数 δ 被用来调整每个 bin 的大小以减少分流的数量；3) 通过对 bin 大小的排序指定每个 bin 的输出端口（根据端口带宽），即较大的 bin 被分至带宽剩余较多的端口；4) 控制器更新 OpenFlow 交换机转发路由表中的规则

<sup>1</sup>TCP Segment Offload，简称为 TSO

<sup>2</sup>Generic Receive Offload，简称为 GRO

并重置字节计数器（为下一次循环做准备）。LocalFlow 增加了 TCP 中端系统 dup-ACK 的阈值来抵消数据包重排的负面影响。LocalFlow 被证明是一个在对称网络下可以使 MCF 优化达到最优情况的均衡机制（如在 fat-tree 和 VL2 拓扑中的 1024 个端系统网络下接近最优）。在由链路断开导致的网络失败情况，LocalFlow 也表现较好，与 MPTCP8 个子流的算法表现相近。实际运用中，LocalFlow 的转发路由表只需要很少的空间，这是由于网络中只有较少的数据流需要分解。LocalFlow 的主要问题在于它只是比较适合于对称网络中，例如，当高负载或链路断开导致的拥塞已发生时，LocalFlow 不能很快地处理，且其调度的间隔较长，应该被缩短至毫秒级的时间以适应现有的数据网络。

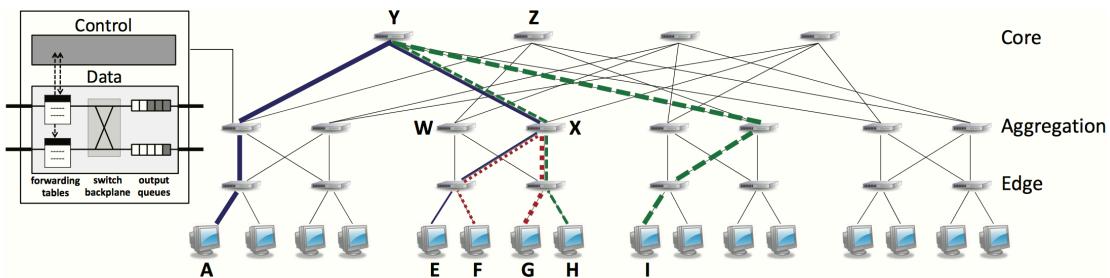


图 3-9 LocalFlow 中的仿真网络拓扑：采用 4 端口交换机的 fat-tree 架构

- DRILL<sup>[36]</sup>: DRILL 是一种只运用交换机本地信息的分布式均衡机制，它指出采集全局信息通常会导致较长的控制循环或额外的开销，这难以满足数据中心网络的实时性要求。另外，即便有了全局信息，由于交换机是分布式的、互相不协同工作的，它们可能同时选择了相同的拥塞情况最轻的路径，从而导致这些路径的拥塞。受到超市排队模型中“两种选择的力量”的启发<sup>[44]</sup>，DRILL 利用交换机本地信息实现了一种数据包随机分配的机制：当数据包到达 DRILL 中的每个交换机处，交换机随机选取两个可用端口，将它们的队列长度与一个有记录的端口比较，之后数据包从三个端口中队列最少的端口发送（有记录的端口是指传输上一个数据包的端口）。DRILL 中没有额外的开销因而不需要更改硬件与协议，这对于部署在大型数据中心网络是一个优点。仿真结果显示，DRILL 在重负载的情况下达到了比 CONGA 和 Presto 更少的 FCT，且 DRILL 还在 incast 场景下比 Presto 的数据包重排更少。DRILL 的主要不确定性在于实际场景下它是否能表现得像仿真时合成的数据那样好。

### 3.5 数据中心网络的流量调度

如前所述，数据中心网络的负载均衡算法解决了收集拥塞信息和交换机处为数据流选路的问题，显然，其主要目的是增大网络吞吐量、减少数据包的 FCT。前文提到的大部分算法在收集拥塞信息时利用了交换机处的缓存（buffer）队列以判断网络拥塞状况，然而它们对于交换机采用了什么样的队列、或者交换机对于先后到达的数据包的处理这个问题没有过多关注，即端系统与交换机之间数据报文的交互大部分是默认采用了 TCP 协议。早在 2010 年，Mohammad Alizadeh 等人就针对数据中心网络对 TCP 协议进行了改进，利用 ECN 作为反馈机制，提出了 DCTCP<sup>[45]</sup>，极大减小了交换机对于缓存空间的需求，且对数据突发有较高的容错率、可以低延迟处理小数据流。同样地，MPTCP<sup>[30]</sup>

亦是对 TCP 协议的一种改进，但是由于 MPTCP 是一种基于端系统（Host-Based）的负载均衡算法，它在实际运用中会增加网络的 incast，较其他算法而言更难部署。Wei Bai 等人在 2015 年提出了 PIAS 架构<sup>[46]</sup>，在端系统和交换机处使用多级反馈队列<sup>1</sup>，在假定数据流信息不可知的情况下使用 DCTCP 协议（主要是为了利用 ECN）和遗传算法解决了 FCT 的最小化问题。由于本节主要关注交换机对于数据流量的初始处理，为与前述内容区分，我们将数据流到达交换机处进入缓存队列的过程称为流量调度，将后续的选路及拥塞信息的采集称为负载均衡。

### 3.5.1 背景

对于在不同时间到达交换机的数据流处理这个问题，我们可以从计算机操作系统中 CPU 对于进程的处理这个类似的问题中受到启发。对于一个具有一颗单核 CPU 的系统而言，在一个时间点上 CPU 只能处理一个进程。为了更好的用户体验，多程序设计的目标就是从宏观上同时运行多个进程，最大化 CPU 的利用率。由于每个进程需要的 CPU 时间不同，且进程有可能需要和 I/O 接口交互，如何使 CPU 高效地在不同进程间切换是 CPU 进程调度的主要研究问题。CPU 进程调度算法的好坏主要有以下标准：

- CPU 利用率：应使 CPU 尽可能一直在处理进程
- 吞吐量：单位时间内 CPU 完成的进程数量
- 周转时间：从进程提交到进程完成的总时间，包括等待进入内存、在进程队列中等待、CPU 中的执行时间、执行 I/O 的时间
- 等待时间：在进程队列中等待时间的总和，CPU 调度算法对进程在 CPU 中运行的时长没有影响，它只对等待时间有影响
- 响应时间：在一个交互系统中，进程提交的时间和 CPU 最早产生输出的时间差是响应时间，即该进程可能在 CPU 中的运行分为几段

### 3.5.2 调度算法

#### 3.5.2.1 先到先服务调度

先到先服务调度算法<sup>2</sup>是一种最简单的 CPU 调度算法，即先提交请求的进程被优先分配到 CPU，它可以用先进先出<sup>3</sup>队列实现。当一个进程进入就绪队列，它的 PCB 被链接到 FIFO 的尾部；每当 CPU 执行完一个任务后，CPU 分配给队头的进程，该进程从队列中删除。采用 FCFS 算法的平均等待时间较长，且由于它是一种非抢占式算法，一旦 CPU 被分配给某一个进程，则不论这个进程在 CPU 上运行多长时间，CPU 会被一直占用直到程序终止或请求 I/O。回到数据中心的场景，显然，当一个大数据流（可能是数百 MB 或者几个 GB）到达交换机时，若采用 FCFS 调度机制会造成这一条路径的长时间拥塞，因此 FCFS 调度算法不适用于数据中心网络。

<sup>1</sup>Multiple Level Feedback Queue，简称为 MLFQ

<sup>2</sup>First-Come, First Served (FCFS) Scheduling Algorithm

<sup>3</sup>First Input First Output，简称为 FIFO

### 3.5.2.2 轮转法调度

轮转法<sup>1</sup>调度算法是为分时系统设计的，它增加了抢占机制实现进程间的切换。通常定义一小段时间（十到一百毫秒）为一个时间片。就绪队列被看作一个循环队列，CPU 调度机制在循环队列中循环分配，按时间片分配给队列中的进程。为了实现 RR 调度，首先将就绪队列以 FIFO 的形式存储，之后调度算法从队列中的第一个进程开始运行一个时间片的时间，依次类推。RR 算法的性能主要取决于时间片的大小，如果时间片取得很大其性能就会近似 FCFS 算法，但是时间片也不是越小越好，因为若时间片的长度小于上下文切换的时间，CPU 的时间主要会浪费在进程间的切换。经验指出，RR 调度算法下的平均等待时间比较长。

### 3.5.2.3 最短作业优先调度

最短作业优先调度算法<sup>2</sup>的主要思想是每当 CPU 执行完一个任务后把 CPU 分配给具有最短 CPU 区间的进程（此处是指下一个 CPU 区间的长度，并不是指该进程需要的总长度）。若两个进程是同等长度的，就使用 FCFS 调度方法。可证明 SJF 是最佳的，即对于一组给定的进程，SJF 算法的等待时间最短。这是因为将短进程移至长进程前，短进程等待时间的减少大于长进程等待时间的增加<sup>[47]</sup>。然而实际情况是，我们很难知道下一个 CPU 区间的长度。一种方法是近似 SJF 调度，通过预测下一个 CPU 区间的长度（通常预测为之前 CPU 区间长度的指数平均）。但是对于数据中心网络的大小符合长尾分布（如图3-10）的数据流而言，若交换机处理的前面大量的数据流都是短数据流，如果用指数平均预测，很可能造成下一个预测偏差极大的情况发生（即遇到一个大数据流），可能造成拥堵或对后续的选路产生影响，因此 SJF 在数据中心场景下无法实际应用。

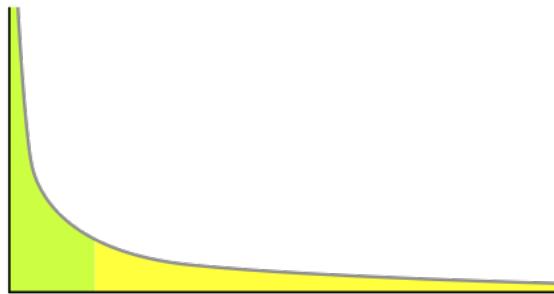


图 3-10 长尾分布示意图

### 3.5.2.4 多级反馈队列调度

多级反馈队列（MLFQ）调度算法将就绪队列分成多个独立队列。根据进程的优先级分配到相应队列，每个队列可以有各自不同的调度算法，如前台进程和后台进程可以分成两个队列；前台进程使用 RR 算法，后台进程使用 FCFS 算法。此外，队列之间也存在调度，通常采用固定优先级抢占调度。MLFQ 调度算法可以由如下五个参数定义：

- 队列数量

<sup>1</sup>Round-Robin，简称为 RR

<sup>2</sup>Shortest Job First (SJF) Scheduling Algorithm

- 各队列采用的调度算法
- 升级到较高优先级的条件
- 降级到较低优先级的条件
- 进程提交时进入某个队列的条件

MLFQ 的定义使它成为 CPU 调度中最常采用的机制，因为它可以被配置以适应各种系统或场景。在数据中心网络中交换机对于数据流的处理可以采用 MLFQ 调度算法，且其较多参数的特点赋予人工智能算法以施展其强大函数拟合能力的空间。

### 3.5.3 PIAS 架构下的流量调度

PIAS<sup>[46]</sup> 默认已有较好的负载均衡算法，主要考虑数据中心网络中的流量调度问题。PIAS 主要由以下三部分构成：1) 中心控制器周期性地从端系统采集流量信息，计算数据流进入 MLFQ 各队列的阈值，并将计算结果分发至各端系统；2) 端系统使用该阈值为即将发送的数据包标记优先级；3) 交换机采用 MLFQ 调度算法将数据流继续向目的地址转发。PIAS 架构的主要贡献在于解决了如下的三个问题：

- 如何确定数据流进入 MLFQ 队列的阈值：PIAS 通过遗传算法解决了 FCT 的最小化问题，确定了性能良好的 MLFQ 的阈值
- 数据中心的流量时刻在变化，如何保证动态环境中 PIAS 的表现：受到基于 ECN 的速率控制<sup>[45]</sup> 的启发，PIAS 也采用相似的机制缓解了已确定的阈值与变化的数据流量特征间的不匹配问题。
- 如何保证 PIAS 与 TCP/IP 协议栈的兼容性：PIAS 在端系统中采用 DCTCP 协议或其他包含 ECN 通知的其他 legacy TCP 协议。

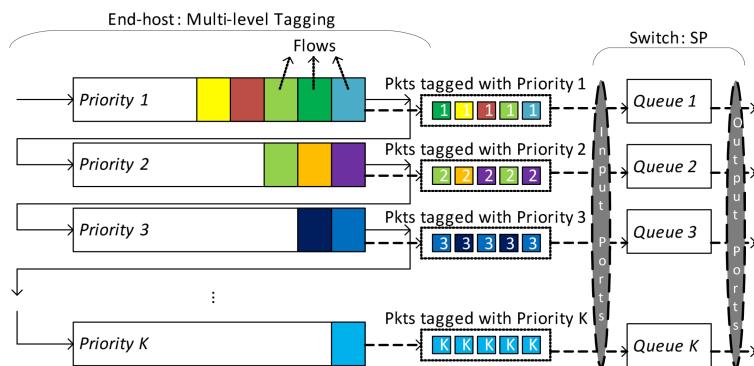


图 3-11 PIAS 中的多级反馈队列

## 第四章 基于深度增强学习的流量优化系统

基于前文对于增强学习算法、数据中心的流量特点、负载均衡算法及流量调度问题的讨论，我们提出一种可以根据 5G Cloud RAN 架构下的数据中心网络流量实时动态地调整参数以达到最优负载均衡的系统——基于深度增强学习的流量优化系统。该系统解决了数据流在数据链路上从端系统、边缘交换机<sup>1</sup>、聚合交换机、核心交换机直到目的地址的传输过程中历经的发送、入队、拥塞信息采集、选路等问题，将使用增强学习的流量调度算法和负载均衡机制相结合，给出数据中心网络中流量优化的一种解决方案。与 Li Chen 等人的工作<sup>[48]</sup>不同，本文更关注于负载均衡机制效果的优化。

### 4.1 系统架构

本文提出的系统采用混合型结构，由集中式的流量调度算法和分布式的负载均衡机制两部分组成。集中式流量调度是一种基于 PIAS 架构的多层次调度机制，MLFQ 所采用的阈值参数和大数据流均由核心层设置和处理，核心层负责向外围层分发 MLFQ 队列的阈值、周期性采集数据流量信息等，端系统与交换机之间的通信采用包含 ECN 通知的 legacy TCP 协议。负载均衡机制采用 DRILL 分布式均衡算法，利用网络虚拟化技术中的封装格式实现拥塞信息的采集工作，以维护选路用的拥塞信息表。

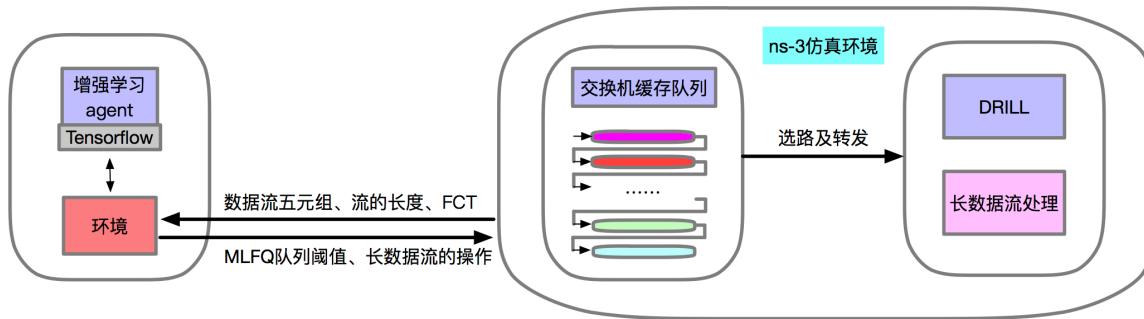


图 4-1 系统整体架构

#### 4.1.1 集中式流量调度

目前将深度增强学习算法应用于数据中心网络的主要问题是数据流信息采集与决策生成之间的较长延迟，对于当前数据中心大于 10Gbps 的链路带宽，要想达到数据流级别的优化，操作的往返延迟至少要在毫秒级，如果不引入特殊设计的硬件这对于深度增强学习是不可实现的。鉴于数据中心网络数据流大小的长尾分布，应该把短数据流的处理在端系统处解决，将较长的数据流交给一个中心组件处理。PIAS 架构利用 MLFQ 解决流量调度这一问题，刚好适合于深度增强学习算法，这是因

<sup>1</sup>在数据中心网络结构中叶结点交换机、边缘交换机、架上（Top of Rack，简称为 ToR）交换机均指与端系统直接相连的交换机，聚合交换机（aggregation switches 或 aggregate switches）和核心交换机（core switches）相当于叶脊结构中的脊结点交换机（spine switches）

为 MLFQ 的降级阈值需要靠计算确定，以适应特定环境。另外，当较长的数据流被降级到某优先级后，就可以交由中心组建处理，对长数据流的路由、速率限制、优先级等做出单独的决策，而这一过程同样可由深度增强学习算法来实现。

#### 4.1.1.1 MLFQ 阈值的确定

PIAS 采用分布式的数据包标记，在每个端系统处有  $k$  个优先级  $P_i$ ,  $1 \leq i \leq K$  和  $(K - 1)$  个降级阈值  $\alpha_j$ ,  $1 \leq j \leq K - 1$ ，其中从  $P_1$  到  $P_K$  的优先级依次递减， $\alpha_1 < \alpha_2 < \dots < \alpha_{K-1}$ 。每当端系统生成一个新的数据流，它的数据报文会被标记为最高优先级，随着其发送的字节数越来越多，该数据流的数据包优先级逐渐下降到  $P_j$  ( $2 \leq j \leq K$ )。从  $P_{j-1}$  降级到  $P_j$  的阈值是  $\alpha_{j-1}$ 。我们假定  $\alpha_K$  为正无穷，这样最大的一些数据流的后发送的数据包都在第  $K$  个优先级的队列中。标记利用的是 IP 报文的 DSCP 域。通过数据流的已发送字节数和降级阈值，端系统可以做本地化的决定。当网络流量情况变化时，由中心组件对队列的降级阈值做调整。我们定义数据流大小的累积分布函数  $F(x)^1$ ，并用  $L_i$  表示给定数据流带给第  $i$  级队列  $Q_i$  ( $i = 1, \dots, K$ ) 的数据包的个数。由优先级队列的定义可知， $L_i$  的期望： $E[L_i] \leq (\alpha_i - \alpha_{i-1})(1 - F(\alpha_{i-1}))$ 。定义数据流的到达速率为  $\lambda$ ，那么数据包到达第  $i$  级队列  $Q_i$  的速率  $\lambda_i = \lambda E[L_i]$ 。队列的服务速率取决于更高优先级的队列是否为空，因此当链路的服务速率为  $\mu$  时，最高优先级  $P_1$  的容量总为  $\mu_1 = \mu$ 。定义  $\rho_i = \lambda_i / \mu_i$  为  $Q_i$  的利用率，则  $Q_1$  的空闲率为  $(1 - \rho_1)$ ， $Q_2$  的服务速率为  $\mu_2 = (1 - \rho_1)\mu$ 。依此类推，我们有  $\mu_i = \prod_{j=0}^{i-1} (1 - \rho_j) \mu$ ,  $\rho_0 = 0$ 。队列  $i$  的平均延迟时间  $T_i = 1/(\mu_i - \lambda_i)$ 。对于一个大小在  $[\alpha_{i-1}, \alpha_i]$  区间内的数据流，它将经历不同优先级队列造成的延迟直到它到达第  $i$  级队列。令  $i_{max}(x)$  为比  $x$  大的最小阈值队列级数，那么对于大小为  $x$  的数据流，其平均 FCT： $T(x) \leq \sum_{i=1}^{i_{max}(x)} T_i$ 。我们用  $g_i = F(\alpha_i) - F(\alpha_{i-1})$  表示大小落在  $[\alpha_{i-1}, \alpha_i]$  区间内的数据流的百分比，因此， $g_i$  就是两个连续队列阈值之间的间隔。如果用  $g_i$  等价地表示  $\alpha_i$  ( $g_i \geq 0, i = 1, \dots, K - 1$ )，FCT 最小化问题就可以表达为如下形式：

$$\min_g \tau(g) = \sum_{l=1}^K (g_l \sum_{m=1}^l T_m) = \sum_{l=1}^K (T_l \sum_{m=l}^K g_m) \quad (4-1)$$

我们将上式转换成增强学习的语言：

- 状态空间：状态由网络中当前时刻  $t$  已完成数据流的集合  $F_t^d$  表示。每个数据流由一个 5 元组确定：源 / 目的 IP 地址，源 / 目的端口号，传输协议。由于这里只考虑已完成的数据流，数据流的 FCT 和大小被看作是流的属性，即一个数据流共有 7 个特征。
- 动作空间：动作空间由中心组件计算得到，在时间  $t$ ，由 agent 确定的动作（或决策）是一组 MLFQ 的队列阈值  $\alpha_t^t$ 。
- 奖励：环境给 agent 反馈的奖励是对前一时刻的动作有多好的评价，这里将连续两个时刻的目标函数的比值作为奖励： $r_t = \tau_{t-1}/\tau_t$ 。它标志着前一时刻的动作是否导致了 FCT 的降低，或者对于整体性能有所提升。

由于此增强学习问题的动作空间是连续的，应采用第2.3.1小节中的 DPG 算法。如算法2-2所示，对于某时刻从端系统新到达的数据流采用深度神经网络进行学习，在经验回放机制中存储一个四元组  $s_t, a_t, r_t, a_{t+1}$ 。奖励（或称回报） $r_t$  和下一时刻的状态  $s_{t+1}$  是在端系统的下一次更新时获得的，因此

<sup>1</sup> $F(x)$ ：任一给定数据流的大小比  $x$  字节小的概率

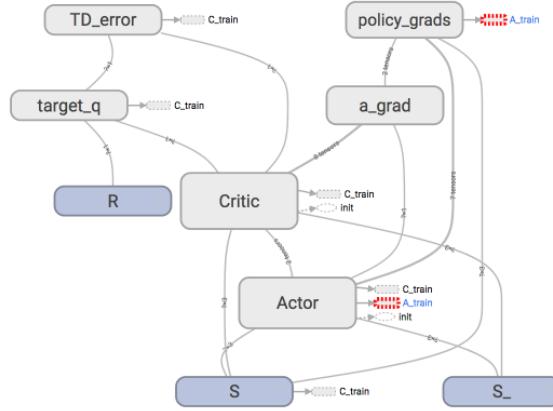


图 4-2 DDPG 网络模型在 tensorboard 中的结构图

agent 需要在缓存中保留  $s_t$  和  $a_t$  直到收集完四元组的所有元素。神经网络的参数更新是从经验回放机制的记忆库中随机选取一批特定数量的四元组，以实现更稳定的学习过程、避免参数不收敛。通过计算前后时刻的目标函数的商，可以得到后一时刻的动作的奖励，运用 actor-critic 算法最终实现式4-1的解决（网络结构如图4-2）。

#### 4.1.1.2 长数据流的处理

我们将 MLFQ 的最后一个阈值  $\alpha_{K-1}$  作为判断长数据流和短数据流的标准，大于该值的数据流均由本地处理。作为上述 DDPG 算法的输出值， $\alpha_{K-1}$  是根据网络情况实时变化的。对于长数据流，我们同样采用 PG 算法：

- 状态空间：状态由当前时刻  $t$  网络中所有活跃的数据流  $F_t^a$  和已完成数据流的集合  $F_t^d$  表示。每个数据流由 5 元组确定，除此之外活跃数据流还有一个优先级的属性；已完成的数据流有流的大小和 FCT 属性。
- 动作空间：对于  $t$  时刻的每一个数据流  $f$ ，agent 为它做出的动作是  $Q_t(f), R_t(f), P_t(f)$ ，其中  $Q_t(f)$  是数据流的优先级， $R_t(f)$  是速率限制， $P_t(f)$  是数据流的传输路径。 $Q_t(f)$  是从数据流到优先级的映射，即  $Q_t(f) \in [1, K]$ 。
- 奖励：只计算已完成数据流的回报：可以选择发送速率、链路利用率、连续时刻的吞吐量的比值或差等。由于较难实时获取数据流级别的信息，这里选择对于已完成数据流的前后时刻平均吞吐量的比值作为回报。

考虑到 DDPG 的收敛相对较慢，以及状态空间可以表达为一个离散空间，此处对于长数据流的处理采用 Stochastic 策略梯度方法（详见第2.2.1小节）。网络结构如图4-3所示。

#### 4.1.2 分布式负载均衡

本小节的负载均衡是针对短数据流而言的，这是因为当长数据流在端系统处的已发送字节大于一定阈值后，它们的数据报文会被降级至较低优先级队列交由中心组建处理。如前面讨论过的，由于深度增强学习的运行时间不允许对于每一个数据流实时做出决策，因此我们在短数据流的负载均

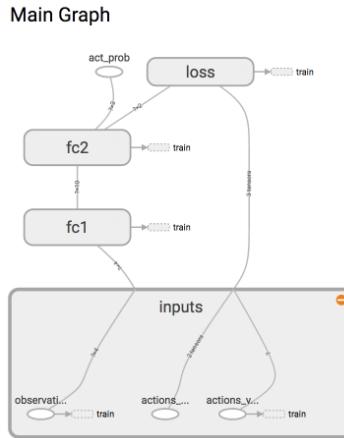


图 4-3 Stochastic 策略梯度方法的神经网络在 tensorboard 中的结构图

衡机制中采用分布式的方法，即不由中心组件控制选路等过程。我们在第3.4.3小节中讨论的负载均衡算法选出一种适合于本文环境的算法。首先，MPTCP 算法主要基于 ECMP 路由机制，当其中的子流被分配至一条拥塞情况较严重的路径时，对于其性能的影响较大，且使用多条链路有时只能带来有限的增益；作为负载均衡问题的经典算法 CONGA，尽管其仿真结果上表现很好，但由于它需要对交换机的软件进行修改，并采用 VxLAN 的头部传输链路拥塞信息，这对于大型数据中心网络的部署和实际效果有一定影响（等待拥塞信息的反馈常常需要花费几个往返时延）；与 CONGA 算法相似，CLOVE 也采用了虚拟化技术，需要对交换机软件进行修改；Expeditus 的拥塞信息采集的延迟对于实际场景中也较长，且其对于网络拓扑有较严格要求，不是一种较为普适的模型；和 MPTCP 一致，Presto 也是基于 ECMP 均衡机制，由于 ECMP 在实际部署时可能存在重大问题，且 Presto 的数据流分割对于 PIAS 架构是不适用的；LocalFlow 在面对拥塞情况时可能表现很差，其调度的间隔较长，不能保证对于拥塞情况的实时处理；而 DRILL 负载均衡机制不仅满足了我们对于短数据流实时性处理的要求，且其不需要更改硬件或协议等，在不同场景下表现均很好，我们在分布式负载均衡中选择 DRILL 算法。

DRILL 算法的出发点是对于流的平均分配<sup>1</sup>，即均等地将达到的数据流分配至所有最短路径上。在理想情况下， $n$  台聚合交换机会收到 pod 间数据交互中来自叶结点交换机的  $1/n$  的数据，这样源地址、目的地址相同的路径会经历同等的利用率。本质上，不论是 CONGA，Hedera，ECMP，还是 Presto 等负载均衡算法的目标都是对于 ESF 的近似。由于 DRILL 采用的是数据包的粒度对数据流进行选路决策，比一般负载均衡机制要更细致，这就需要对交换机队列提出更高的要求，而这种特点刚好符合我们采用的 PIAS 架构下的 MLFQ 多优先级队列。当然，这种细粒度的代价是数据包的重排问题，由于 DRILL 在选路过程中将数据流分成数据包的粒度，其每个数据包所经历的链路数量比一般算法（如 Presto）多近 30%。数据包在不同路径上经历的队列延迟方差越大，重排问题越严重，DRILL 对此问题的解决是利用优先级队列严格控制数据包的队列延迟。因此数据包的重排实际上很少出现（乱序数据包数量的均值少于 0.5），这也是 DRILL 在不同场景下都有很好性能的原因。

<sup>1</sup>Equal Split Fluid，简称为 ESF

### 4.1.3 网络拓扑

仿真所用的网络拓扑采用如图4-4所示的叶脊结构：一台核心交换机（CoreSwitch）下连有六台聚合交换机（AggSwitch），每两个聚合交换机组成一个 pod，一个 pod 中有三台叶结点交换机和六台端系统（Host），其中叶结点交换机与端系统直接相连。端系统到叶结点交换机间的链路速度为 100Gbps，延迟 500ps；叶结点交换机到聚合交换机、聚合交换机到核心交换机间的链路速度为 40Gbps，延迟 500ps；仿真环境的网络地址为 10.1.1.0，子网掩码 255.255.255.0（如图4-5）。仿真过程中的 pod 内、pod 间数据流如图4-4中的虚线所示。

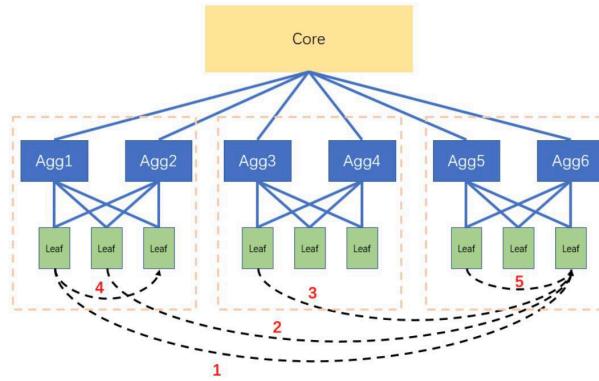


图 4-4 仿真所用的网络拓扑

```

CoreSwitch->GetPortDevice(8), AggSwitch1->GetPortDevice(6));
link2DL[1]=ptrLink2DL->AddLink(1, "40Gbps", "500ps", "ns3::NutsSuiteQue",
CoreSwitch->GetPortDevice(9), AggSwitch2->GetPortDevice(6));
link2DL[2]=ptrLink2DL->AddLink(2, "40Gbps", "500ps", "ns3::NutsSuiteQue",
CoreSwitch->GetPortDevice(10), AggSwitch3->GetPortDevice(6));
link2DL[3]=ptrLink2DL->AddLink(3, "40Gbps", "500ps", "ns3::NutsSuiteQue",
CoreSwitch->GetPortDevice(11), AggSwitch4->GetPortDevice(6));
link2DL[4]=ptrLink2DL->AddLink(4, "40Gbps", "500ps", "ns3::NutsSuiteQue",
CoreSwitch->GetPortDevice(12), AggSwitch5->GetPortDevice(6));
link2DL[5]=ptrLink2DL->AddLink(5, "40Gbps", "500ps", "ns3::NutsSuiteQue",
CoreSwitch->GetPortDevice(13), AggSwitch6->GetPortDevice(6));

for(uint32_t j=0;j<6;j++)
{
    leafLinkUL[j]=ptrLeafLinkUL->AddLink(j+3, "100Gbps", "500ps", "ns3::NutsSuiteQue",
    Serverpod1[j]->GetHostDevice(0), LeafSwitch1[j]->GetPortDevice(10));
    leafLinkDL[j]=ptrLeafLinkDL->AddLink(j+3, "100Gbps", "500ps", "ns3::NutsSuiteQue",
    LeafSwitch1[j]->GetPortDevice(2), Serverpod1[j]->GetHostDevice(1));
    leafLinkUL[j].Add(ptrLeafLinkUL->AddLink(j+3+1, "100Gbps", "500ps", "ns3::NutsSuiteQue",
    Serverpod2[j]->GetHostDevice(0), LeafSwitch2[j]->GetPortDevice(10)));
    leafLinkDL[j].Add(ptrLeafLinkDL->AddLink(j+3+1, "100Gbps", "500ps", "ns3::NutsSuiteQue",
    LeafSwitch2[j]->GetPortDevice(2), Serverpod2[j]->GetHostDevice(1)));
    leafLinkUL[j].Add(ptrLeafLinkUL->AddLink(j+3+2, "100Gbps", "500ps", "ns3::NutsSuiteQue",
    Serverpod3[j]->GetHostDevice(0), LeafSwitch3[j]->GetPortDevice(10)));
    leafLinkDL[j].Add(ptrLeafLinkDL->AddLink(j+3+2, "100Gbps", "500ps", "ns3::NutsSuiteQue",
    LeafSwitch3[j]->GetPortDevice(2), Serverpod3[j]->GetHostDevice(1)));
}

NetDeviceContainer host=NetDeviceContainer(leafLinkUL[0].Get(0), leafLinkUL[1].Get(0));
host.Add(NetDeviceContainer(leafLinkUL[0].Get(1), leafLinkUL[0].Get(5)));
host.Add(NetDeviceContainer(leafLinkUL[0].Get(2), leafLinkUL[0].Get(5)));
Ipv4AddressHelper ipv4;
Ipv4SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfacesUL;
interfacesUL = ipv4.Assign(host);

```

图 4-5 构建网络拓扑的部分源代码

## 4.2 仿真过程

为了检测本文提出的流量优化系统的性能效果，我们对比了 ECMP、CONGA、DRILL 和本文的模型在 ns-3 仿真环境下的 FCT、总传输时延和平均链路利用率。系统中的中心组建深度增强学习模型是用 TensorFlow 深度学习框架<sup>[49]</sup> 实现的，通过在具有两块 NVIDIA GeForce GTX 1080 Ti 显卡的服务器<sup>1</sup>上预训练，使得神经网络的参数不断收敛，学习到不同模式、不同数据流下的最优 MLFQ 参数与长数据流的调度方法。

### 4.2.1 仿真数据源

针对数据中心网络的流量特点，仿真数据源应满足如下特点：

- 网络中应经常有碰撞、拥塞等情况的发生，不同的端系统可能在短时间内同时发送多个数据流，且其大小、持续时间各不相同
- 数据中心网络承担着不同的应用，各应用产生的数据流的优先级不同
- 数据源发出的数据流在统计上应大致符合长尾分布

基于以上考虑，仿真所用的数据源如图4-6所示，BurstTime 为数据流的突发时间，PktSize 是其大小，Priority 为数据流本身的优先级，E2E Delay 为端到端延迟。

	BurstTime	PktSize(Byte)	Priority	E2E Delay(μs)
<b>BBL-BBH</b>				
R1	35.71	11264.00	5	25.00
R2	35.71	106496.00	4	100.00
R3	35.71	21504.00	3	100.00
R4	35.71	3072.00		100.00
R5	500.00	65536.00		100.00
R6	500.00	65536.00	2	500.00
R7	5000.00	1703936.00		500.00
R8	10000.00	1703936.00		10000.00
R9	500.00	65536.00	1	100.00
R10	100.00	6144.00		100.00

图 4-6 仿真数据源

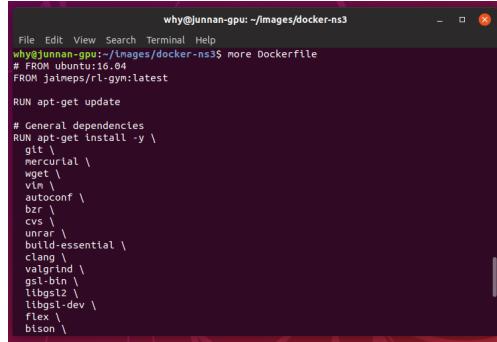
### 4.2.2 仿真环境搭建

笔者在服务器上安装了 Docker 平台<sup>[50]</sup>，在容器（container）中运行仿真环境 ns-3。由于增强学习算法需要利用 GPU，且 Docker 中已有增强学习模块的镜像（image），因此我们采用编译 dockerfile 的方式构建自己的镜像（如图4-7所示），再从该镜像实例化为容器。由于我们希望能够看到增强学习的训练过程，我们在容器中额外开放一个端口使它能运行 jupyter notebook 服务<sup>2</sup>。我们利用 ns-3 中提供的 Ethernet Switch 和 Standard Host 模块实现常规以太网交换机和端系统，在 github 中 snowzjx/ns3-load-balance repository<sup>3</sup>的基础上实现 ECMP、CONGA、DRILL 以及本文提出的架构的仿真。

<sup>1</sup>Intel Xeon CPU E5-4640 v3 @ 1.90GHz, 48 CPUs, Ubuntu 17.10, GNU/Linux 4.13.0-21-generic x86\_64

<sup>2</sup>从镜像实例化容器的命令行为 “sudo nvidia-docker run -d -p 7771:8888 why2011btv/ns-3.rl-gym”，进入该容器的命令行为 “docker exec -it 510fb70d5fc4 bash”。

<sup>3</sup><https://github.com/snowzjx/ns3-load-balance>



```
wby@junnan-gpu: ~/images/docker-ns3
File Edit View Search Terminal Help
wby@junnan-gpu:~/images/docker-ns3$ more Dockerfile
# FROM ubuntu:16.04
FROM jaimeps/rl-gym:latest

RUN apt-get update
# General dependencies
RUN apt-get install -y \
    git \
    mercurial \
    wget \
    vim \
    autoconf \
    bzip2 \
    curl \
    curl \
    build-essential \
    clang \
    valgrind \
    gsl-bin \
    libgsl2 \
    libcurl-dev \
    flex \
    bison \

```

图 4-7 从 dockerfile 构建自己的镜像，其中“From jaimeps/rl-gym:latest”这一行显示了本仿真环境的基础镜像 rl-gym，后续行均为在其基础上构建 ns-3 环境的过程

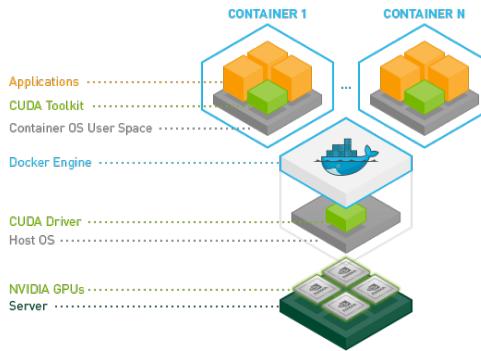


图 4-8 Docker 平台与容器

### 4.2.3 增强学习深度模型的训练

深度增强学习算法主要涉及两个模型的训练，一个是为确定 MLFQ 队列阈值的 DDPG，一个是长数据流处理的 Stochastic Policy Gradient。其中，DDPG 用到了两个全连接的隐藏层，分别有 600 个神经元，输出层共  $K - 1$  个神经元，对应  $K - 1$  个队列阈值。输入层接收状态输入，由 700 个神经元表示。Stochastic PG 神经网络由 10 层全连接的隐藏层表示，每层有 300 个神经元，agent 接收的状态信息由 700 个神经元作为输入层。二者均训练约 8 小时，参数的收敛与否可以从系统得到的 reward 上判断（如图4-10）。训练过程中，数据流的 FCT 与训练时长如图4-9所示，可以看到训练过程反映在 FCT 上是较平稳的。

### 4.2.4 仿真结果

#### 4.2.4.1 流完成时间

从图4-11可以看出，MLFQ 将数据流分为两批进行处理，短数据流使用 DRILL 算法，长数据流由中心组件单独处理，均可以达到比之前的算法（ECMP、CONGA、DRILL）更好的性能，当然由于系统主要利用 DRILL 算法进行负载均衡，因此在部分数据流的表现上与 DRILL 相近。考虑到数据中心网络的数据流符合长尾分布的特征，对于长数据流的处理是至关重要的，本文的系统在长数据流上的表现是所有参考算法中最好的（pod 内相对 DRILL 带来 9.3% 的 FCT 提升、相对 CONGA

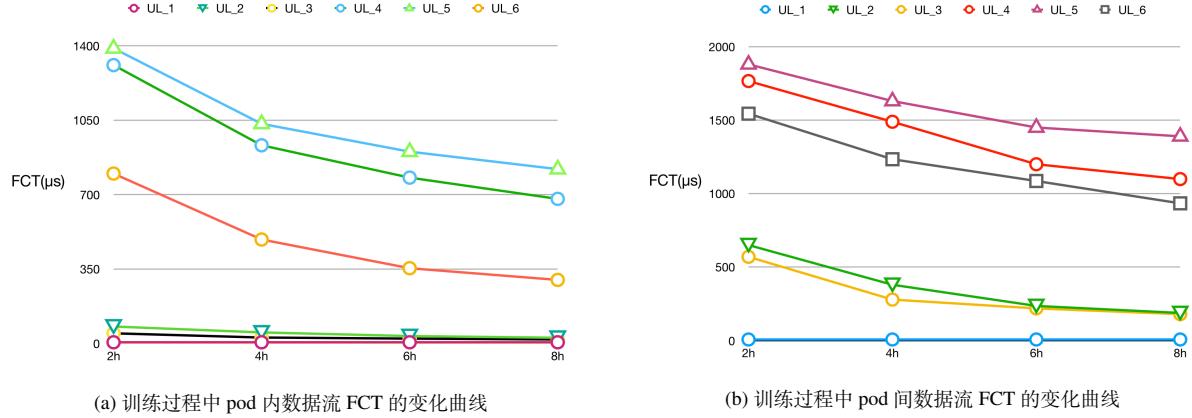


图 4-9 FCT 与训练时长折线图

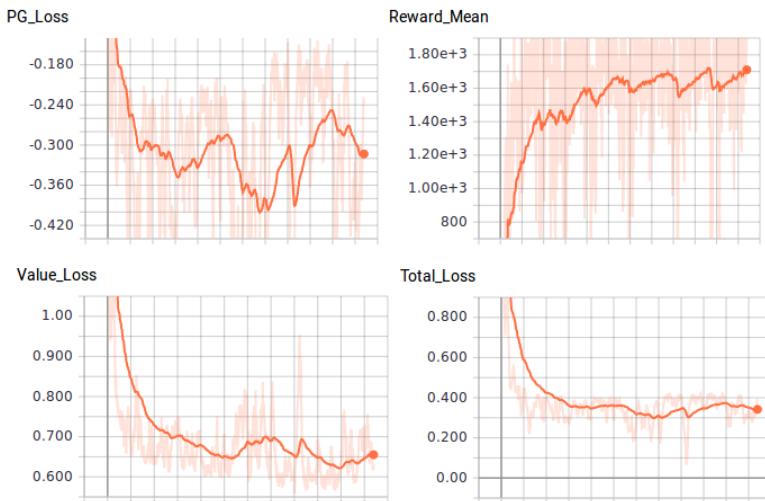


图 4-10 Stochastic Policy Gradient 训练过程

带来 28% 的提升、pod 间相对 DRILL 带来 7.4% 的 FCT 提升、相对 CONGA 带来 23% 的提升）。另外，对于流量负载较轻的网络，几种算法的表现的差别不大；当流量负载增加时，可以看到本文提出的系统对 FCT 有较明显的提升。

#### 4.2.4.2 总传输时延

通过测量数据发往不同 pod 的端口（本组实验共设置 5 个端口）可以比较选路过程中各个算法的优劣。总体而言，CONGA 与 ECMP 的完成时间相近，DRILL 和本文的系统相近，在涉及不同 pod 间的数据传输（路径需要经过聚合交换机）和平均完成时间较长的数据，本文系统均表现优异，结果如图4-12所示。

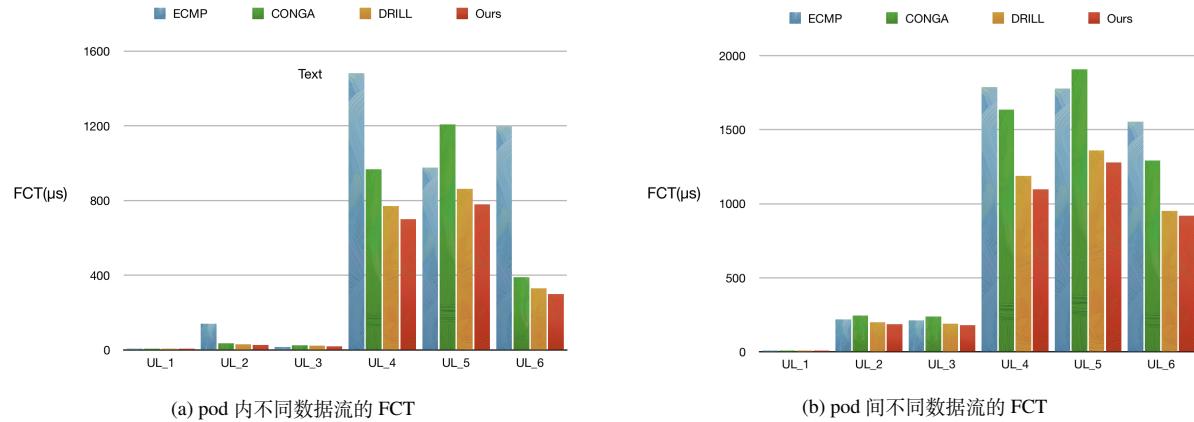


图 4-11 数据流完成时间

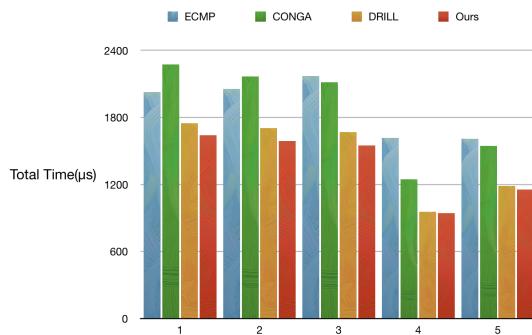


图 4-12 不同发送端的总传输时延

#### 4.2.4.3 平均链路利用率

如图4-13 (a)，可以看出在上行链路叶结点交换机的端口利用率上，CONGA 在一开始的利用率要高于另外两种算法，这是因为初始时端口的拥塞并不严重，可以根据不同端口的拥塞情况来选择端口，因此利用率更高。后一时刻的利用率下降是因为这个周期的数据包已经传输完毕。本文提出的流量优化系统的端口利用率在特定情况下较高，这是因为对于长数据流而言，其流速率、转发路径、优先级由中心组件的离散增强学习算法单独确定，会导致其利用与普通数据流不同的端口。聚合交换机的端口利用率情况与叶结点交换机类似，这是因为叶结点交换机选择的两个端口对应两个聚合交换机，而每个聚合交换机仅有一个端口通向核心交换机，因此数据量与利用率情况类似（由图4-4的拓扑结构决定）。

对于下行链路（如图4-14），四种算法在端口上的平均利用率基本相同，这是因为在核心交换机上均采用了 ECMP，所以端口利用情况基本相同（DRILL 主要针对 Clos 网络的二层结构，对于具有核心交换机的场景没有特殊处理）。

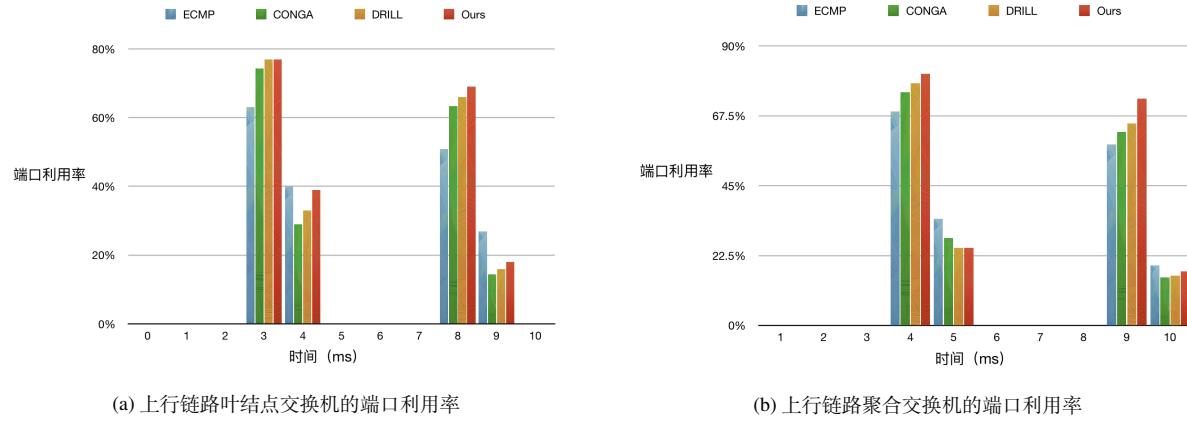


图 4-13 上行平均链路利用率

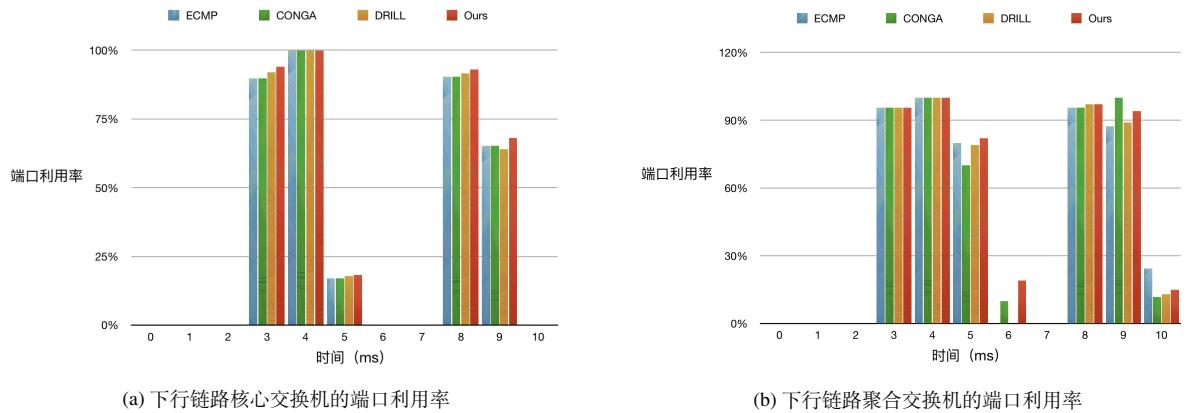


图 4-14 下行平均链路利用率

#### 4.2.5 性能分析

可以看出，通过将 DRILL 负载均衡机制与流量调度算法相结合，对长数据流和短数据流分开处理，可以实时动态地控制数据传输选择最合适的路径，缩短数据流的完成时间，减少总传输时间，在特定情况下提升链路利用率以实现数据中心网络中的流量优化。当然，本文提出的架构也有需要改进的地方：DRILL 负载均衡机制在三层网络中的性能有待加强，在核心交换机出只靠 ECMP 算法可能导致较大的问题；当整个链路的数据量较大时，基于本地的拥塞信息很容易判断为达到最大值，此时只能根据算法随机选择，因此拥塞情况可能不会以最快速度消失。综上所述，本文提出对流量优化系统能够实现良好的负载均衡效果，能有效降低数据流的完成时间，增加链路利用率，对拥塞问题可以得到很好的解决，从性能上看，由于加入了 PIAS 架构中的 MLFQ 优先级队列，并利用近年来提出的深度增强学习算法，使得系统性能对于现有的负载均衡算法有较明显的提升。

## 第五章 结论

本文通过总结三个领域的最新进展：深度增强学习算法、数据中心网络中的数据流负载均衡算法和交换机处的缓冲区流量调度算法，针对 5G Cloud RAN 场景下的负载均衡问题提出了一个新的解决方案，此处的负载均衡是指广义的负载均衡，即包含了从端系统到交换机到端系统整体路径上的负载均衡，涵盖数据流的优先级标记、交换机缓存处的入队、出队操作及选路、拥塞信息采集等一系列过程。该解决方案是一个混合型的结构，对于短数据流采用性能极佳的 DRILL 分布式负载均衡算法，基于本地信息给出快速的均衡决策；利用离散形式的策略梯度算法和决定性（连续空间下）的 actor-critic 算法集中式地解决了长数据流的选路、流速率、优先级和对于网络架构中交换机处采用的 MLFQ 的队列阈值的实时部署，根据动态的网络流量情况，以及以往数据得到的经验，给出一种接近最优的流量优化方法。

由于实验条件的限制，本文没有在实际场景下做相关实验，只是在 ns-3 仿真软件中做了性能分析。我们知道，数据中心网络中数据流量变化迅速，应用种类繁多，采用的网络拓扑结构各不相同（层状、网状或混合型结构、对称或非对称网络、不同类型的商品交换机和端系统等），这些都是对流量的负载均衡算法有着根本影响的因素。本文提出的系统比较适用于 Clos 架构的数据中心网络，如果网络是 3 层的，那么在实际部署过程中就需要对分布式负载均衡算法进行改动。另外，我们还需要考虑到深度增强学习模型的启动问题，即当中心组件发生宕机或其他故障时，此时其网络参数需要提前在硬盘中保存，因为系统重启后，重新训练深度模型所花费的延迟是不可接受的，且在重启后如何利用之前保存的网络参数以迅速恢复对于 MLFQ 队列阈值和长数据流的实时决策亦是我们需要进一步讨论的问题。

另外，在深度增强学习模型的训练过程中，可以发现其预训练所需的时间很长，因此进一步的研究方向可以对算法的效率进行提升。当然，鉴于人工智能算法的复杂性，目前实际应用中效果较好的模型，如计算机视觉领域的 Mask-RCNN，自然语言处理领域的 ELMo、BERT，知识图谱领域的 KBGAN 等算法都需要长时间的预训练，对于深度学习算法的改进是推动整个人工智能时代的动力。我们不能忘记传统算法的优美，亦要利用最前沿模型的强大功能，将二者融合起来才会使问题得到最终解决。

## 参考文献

- [1] RAN C, WANG S, WANG C. Optimal load balancing in cloud radio access networks[C]//. [S.l.]: [s.n.], 2015.
- [2] NIU Z, WU Y, GONG J. Cell zooming for cost-efficient green cellular networks[C]//. [S.l.]: [s.n.], 2010.
- [3] LIN X, WANG S. Efficient Remote Radio Head Switching Scheme in Cloud Radio Access Network: A Load Balancing Perspective[C]//. [S.l.]: [s.n.], 2017.
- [4] SILVER D. Mastering the game of Go with deep neural network and tree search[C]//. [S.l.]: [s.n.], 2016.
- [5] CH. W, P. D. Q-Learning[C]//. [S.l.]: Kluwer Academic Publishers, 1992.
- [6] MNIH V, KAVUKCUOGLU K, SILVER D. Human-level control through deep reinforcement learning[C]//. Vol. 518. [S.l.]: Macmillan Publishers, 2015.
- [7] MNIH V, KAVUKCUOGLU K, SILVER D. Playing Atari with Deep Reinforcement Learning[C]//. [S.l.]: [s.n.], 2013.
- [8] Van HASSELT H. Double Q-learning[C]//. [S.l.]: [s.n.], 2010.
- [9] Van HASSELT H, GUEZ A, SILVER D. Deep Reinforcement Learning with Double Q-learning[C]//. [S.l.]: [s.n.], 2016.
- [10] SUTTON R S, MCALLESTER D, SINGH S, et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation[C]//. [S.l.]: [s.n.], 1999.
- [11] SILVER D, LEVER G, HEESS N. Deterministic Policy Gradient Algorithms[C]//. [S.l.]: [s.n.], 2014.
- [12] LILLICRAP T P, HUNT J J, PRITZEL A. Continuous control with deep reinforcement learning[C]//. [S.l.]: [s.n.], 2016.
- [13] SERGEY I, CHRISTIAN S. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//. [S.l.]: [s.n.], 2015.
- [14] E. U G, S. O L. On the theory of the brownian motion[C]//. [S.l.]: [s.n.], 1930.
- [15] TSAI C, MOH M. Load Balancing in 5G Cloud Radio Access Networks Supporting IoT Communications for Smart Communities[C]//. [S.l.]: [s.n.], 2017.
- [16] AL-FARES M, LOUKISSAS A, VAHDAT A. A scalable, commodity data center network architecture[C]//. [S.l.]: [s.n.], 2008.
- [17] GREENBERG A. VL2: A scalable and flexible data center network[C]//. [S.l.]: [s.n.], 2009.
- [18] COSTA P. CamCube: Rethinking the data center cluster[C]//. [S.l.]: [s.n.], 2012.

- [19] GUO C. Dcell: A scalable and fault-tolerant network structure for data centers[C]//. [S.l.]: [s.n.], 2008.
- [20] GUO C. BCube: A high performance, server-centric network architecture for modular data centers[C]//. [S.l.]: [s.n.], 2009.
- [21] WANG T, SU Z, XIA Y, et al. Rethinking the data center networking: Architecture, network protocols, and resource sharing[C]//. [S.l.]: [s.n.], 2014.
- [22] CISCO IOS NetFlow. [C]//. [S.l.]: [s.n.], 2018.
- [23] BENSON T, ANAND A, AKELLA A, et al. Understanding data center traffic characteristics[C]//. [S.l.]: [s.n.], 2010.
- [24] BENSON T, AKELLA A, MALTZ D A. Network traffic characteristics of data centers in the wild[C]//. [S.l.]: [s.n.], 2010.
- [25] ZHANG J, REN F, LIN C. Modeling and understanding TCP incast in data center networks[C]//. [S.l.]: [s.n.], 2011.
- [26] HONG Y S, NO J, KIM S Y. DNS-based load balancing in distributed Web-server systems[C]//. [S.l.]: [s.n.], 2006.
- [27] ALIZADEH M. CONGA: Distributed congestion-aware load balancing for datacenters[C]//. [S.l.]: [s.n.], 2014.
- [28] ZHANG J, YU F R, WANG S. Load Balancing in Data Center Networks: A Survey[C]//. [S.l.]: [s.n.], 2018.
- [29] KOOMEY J. Growth in Data Center Electricity Use 2005 to 2010[C]//. [S.l.]: [s.n.], 2011.
- [30] RAICIU C. Improving datacenter performance and robustness with multipath TCP[C]//. [S.l.]: [s.n.], 2011.
- [31] KATTA N. CLOVE: How I learned to stop worrying about the core and love the edge[C]//. [S.l.]: [s.n.], 2016.
- [32] AL-FARES M. Hedera: Dynamic flow scheduling for data center networks[C]//. [S.l.]: [s.n.], 2010.
- [33] SEN S. Scalable, optimal flow routing in datacenters via local link balancing[C]//. [S.l.]: [s.n.], 2013.
- [34] PERRY J. Fastpass: A centralized ‘zero-queue’ datacenter network[C]//. [S.l.]: [s.n.], 2014.
- [35] WANG P, XU H. Expeditus: Distributed load balancing with global congestion information in data center networks[C]//. [S.l.]: [s.n.], 2014.
- [36] GHBORANI S. Micro load balancing in data centers with DRILL[C]//. [S.l.]: [s.n.], 2015.
- [37] HE K. Presto: Edge-based load balancing for fast datacenter networks[C]//. [S.l.]: [s.n.], 2015.
- [38] MYSORE R N. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric[C]//. [S.l.]: [s.n.], 2009.

- [39] DIXIT A, HAO F, MUKHERJEE S, et al. Towards an elastic distributed SDN controller[C]//. [S.l.]: [s.n.], 2013.
- [40] BERDE P. ONOS: Towards an open, distributed SDN OS[C]//. [S.l.]: [s.n.], 2014.
- [41] CUI Y, WANG L, WANG X, et al. FMTCP: A fountain code-based multipath transmission control protocol[C]//. [S.l.]: [s.n.], 2015.
- [42] M.Coudron, S.Secci, G.Pujolle, et al. Cross-layer cooperation to boost multipath TCP performance in cloud networks[C]//. [S.l.]: [s.n.], 2013.
- [43] MAHALINGAM M. Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks[C]//. [S.l.]: [s.n.], 2014.
- [44] MITZENMACHER M. The power of two choices in randomized load balancing[C]//. [S.l.]: [s.n.], 2001.
- [45] ALIZADEH M. Data Center TCP (DCTCP)[C]//. [S.l.]: [s.n.], 2010.
- [46] BAI W. PIAS: Practical Information-Agnostic Flow Scheduling for Commodity Data Centers[C]//. [S.l.]: [s.n.], 2015.
- [47] SILBERSCHATZ A, GALVIN P B, GAGNE G. Operating System Concepts[M]. United States of America: [s.n.], 2004.
- [48] CHEN L, LINGYS J, CHEN K, et al. AuTO: Scaling Deep Reinforcement Learning for Datacenter-Scale Automatic Traffic Optimization[C]//. [S.l.]: [s.n.], 2018.
- [49] API Documentation: TensorFlow. [C]//. [S.l.]: [s.n.], 2017.
- [50] Build and run Docker containers leveraging NVIDIA GPUs. [C]//. [S.l.]: [s.n.], 2018.

## 致 谢

四年的大学学习生活在即将划上一个句号，而于我的人生来说却仅仅只是一个逗号，我将面对新的征程的开始。本研究及论文是在我的导师蒋铃鸽教授和实验室的孙煜学长、梁轶学姐和张曼学姐的亲切关怀和耐心的指导下完成的。伟人、名人固然为我所崇拜，可是我更迫切地想要把我的敬意献给我的导师蒋铃鸽教授。也许我不是您最出色的学生，但您却是我所最尊敬的老师。您是如此的治学严谨，学识渊博，视野广阔，思想深刻，您用心为我营造一种良好的学术氛围，让我的论文更加的严谨。另外实验室的学长学姐们给了我很多的帮助，不论是对于论文内容、方向的把控，还是平日组会中的耐心解答，这都是我的论文得以完成的关键要素。同时，我还要感谢一下我的同窗们，如果没有你们的支持和倾心的协助，我是无法解决这些困难和疑惑，最终能够让本文顺利完成。

在本次论文设计过程中，蒋铃鸽老师对该论文从选题，构思到最后定稿的各个环节给予细心指引与教导，使我得以最终完成毕业论文设计。在学习中，老师严谨的治学态度、丰富渊博的知识、敏锐的学术思维、精益求精的工作态度以及诲人不倦的师者风范是我终生学习的楷模，导师们的高深精湛的造诣与严谨求实的治学精神，将永远激励着我。

上海交通大学，这里严谨的学风、优美的校园环境使我大学四年过的很充实和愉快。我的四年大学时光是在电子系度过的。在这四年时间里，我有幸和许多优秀同学一齐学习，听睿智的电子系老师讲授课程。我学到了很多有用的知识，尤其是对我思想和方法上的指导。这些东西一向对我大学的学习和生活有很重要的指导作用，我相信，这些东西将伴随我走完整个人生的道路。此刻回想起在电子系的日子，还是那么的温馨和惬意，我不能不感谢当时电子系的每一位同学和老师，跟你们在一齐学习、生活，那真是其乐融融，妙不可言！

感谢给我带给参考文献的学者们，多谢他们给我带给了超多的文献，使我在写论文的过程中有了参考的依据。感谢我的爸爸妈妈，感谢写作论文他们为我所付出的一切。养育之恩，无以回报，你们永远健康快乐是我最大的心愿。此时，我的情绪无法平静，从开始进入课题到论文的顺利完成，有多少可敬的师长、同学、朋友给了我无言的帮忙，是你们为我撑起一片天空，在那里请理解我诚挚的谢意。

# A STUDY ON LOAD BALANCING ALGORITHM WITH LIGHTWEIGHT AI

This thesis deals with the general load balancing problem in data center networks. Under the scenario of the Cloud Radio Access Network of the fifth generation of wireless communication system, we propose a new paradigm for a combination of load balancing algorithm and flow scheduling algorithm. The system is a mixed model with several hierarchies, namely centralized flow scheduling component and distributed load balancing component. The centralized flow scheduling scheme employs the state-of-the-art deep learning algorithm, Deep Deterministic Policy Gradient and Stochastic Policy Gradient algorithm, to solve the problem of assignment of rate limit, queue priorities and path selection for the long flows in the datacenter network and the thresholds of the Multi Level Feedback Queue commonly used in modern commodity switches. The distributed load balancing algorithm, which addresses the narrowly-sensed load balancing problem, utilizes the recent model DRILL proposed by Soudeh Ghorbani et al., aiming at achieving lower flow complete time and lower total transmission time, not to mention raising the rate of link utilization.

In the first and second generation of wireless communication systems, cellars or radio access networks (usually shortened as RAN), were separate base stations, each equipped with power, air-conditioning systems, environment monitoring systems, backhaul and other essential components. Each base station has its own radio frequency emission devices, and the radio frequency signals were transferred from the base station on the ground to the antennas, which always resulted in a high loss on the cables. In the third generation of communication systems, however, people employed distributed base station systems, separating remote radio head and baseband unit from each other to the remote radio head to be assigned to the place close to the antennas, given that the optic fibers could extremely reduce the loss of long distance signal transmission. Based on the architecture of distributed systems, Cloud Radio Access Network utilizes the recent large-scale data center network technology to establish the stable, energy-efficient, low-latency, and high-bandwidth connections between baseband units. Besides, the usage of an open platform and real-time virtualization technique enables the dynamic resource allocation between different scenes and in different electronic devices. In such architecture, since the area coverage is much bigger than previous single, separate base stations (when one user switches from one baseband unit to another, it is highly possible that the user is still in the same pool of BBUs), the data flow in the cellular networks can be solved using dynamic and real-time load balancing algorithm in the baseband unit pool.

The traditional problem of load balancing has been widely studied by researchers. Examples like cell zooming has been applied to the second and third generation of wireless communication systems. The highly efficient switching scheme in Cloud Radio Access Network proposed by C. Ran and S. Wang et al. considers the energy condition to achieve the optimal load balancing. They periodically measure the index for balance.

When the index is lower than a certain threshold, they re-design the area covered by the base stations to achieve a new status of balance. C. Tsai and M. Moh compared eight methods of load balancing that mainly aim at reducing the communication latency in the scene of Internet of Things. These algorithms either use pre-defined thresholds or metrics to monitor the load or ports of the network switches, or carry out fixed or non real-time strategies. On the contrary, B. Shahriari and M. Moh proposed generic online learning structure using the method of deep reinforcement learning in an partially visible environment, which can be adapted to solve the load balancing problem in the scenario of 5G Cloud Radio Access Network.

Inspired by the work of B. Shahriari and M. Moh, we would like to utilize the great capability of fitting any complex functions of the deep neural networks. Since the aims of load balancing (in a narrow sense) and flow scheduling are the same: reducing the flow completion time, we would like to combine this two scheme into one system, and apply deep reinforcement learning to face with the data center network whose flow features are extremely dynamic since the agent in a reinforcement learning model can handle such problems and make kind of real-time decisions towards the changes.

Hence in this thesis, we first study the deep reinforcement learning algorithms. The very first reinforcement learning algorithms are value based: they use the tables to learn the agent's proper actions towards different states and rewards given by the environment. By utilizing the great capability of fitting complex functions of the deep neural networks, we can develop deep Q-networks which is of great performance. By introducing experience replay, we can solve the problem discovered by Volodymyr Mnih et al. that the expression for Q values using neural networks is unstable. Besides, Hado van Hasselt introduced Double Deep Q-Networks employing double Q-learning scheme in 2016 to solve the problem of overestimation for the parameters. Different from value-based methods, there are another group of reinforcement learning algorithms that directly determine the policy, namely a sequence of actions taken by the agent. When the action space is discrete, the method is called stochastic policy gradient since the model outputs the distribution of actions, but not a single action. The REINFORCE algorithm proposed by Richard S. Sutton, is a typical stochastic one. By using gradient descent, the algorithm consistently updates its parameters towards the tuples the agent got from multiple attempts in the environment, until the parameters come to convergence. In 2016, David Silver et al. proposed a series of algorithms which belong to deterministic policy gradient. They consider a deterministic policy, in which the neural network outputs the action itself directly. The proposal itself becomes a surprise to most of the researchers since it was previously believed that deterministic policy gradient did not exist or could only be obtained when using a specific model. Deep Deterministic Policy Gradient is a typical algorithm of this type. By using the core idea of actor-critic, there are four neural networks in the model, two of them are actors, the other ones serve as critics. And both actor and critic employ the scheme of replay buffers.

In the third chapter of the thesis, we discuss load balancing in 5G Cloud Radio Access Network. We first consider the architecture and traffic features in data center network in the era of 5G. Multiple architectures have been introduced during these years, among which Fat-tree, Camcube and BCube are three typical structures which aim at different scenarios or applications. Fat-tree is a multi-layer structure whose center is switches, while CamCube is net structure whose center is hosts. BCube is a mixed structure which uses hosts

and switches to transfer data at the same time. After the discussion of datacenter network architecture, the definition, targets, and main steps of load balancing in data center networks are introduced. The main steps are the collection of congestion information and path selection for the flows. We then compare the load balancing in data center networks and wide area network to find out what is the main concern in the algorithm design. We finally conclude representative load balancing designed for data center networks in recent years. Among the two categories, namely centralized and distributed schemes, there are superb proposals concerning different specific aims and scenarios. Hedera is a centralized algorithm which aim at large traffic flows while Fastpass is hybrid structure of centralized and distributed schemes. MPTCP is a modification on TCP which assists the process of collection of congestion information. Served for leaf-spine topology, CONGA is a classic algorithm whose performance is nearly optimal by utilizing the virtualization techniques and flow let design. CLOVE is another algorithm that uses the ECN marks in the network virtualization to collect congestion information, while Expeditus is aimed at solving the problem of Clos architecture. Based on ECMP, Presto achieves the optimal balancing in an asymmetric topology and LocalFlow, as its name shows, is a scheme employs OpenFlow switches to achieve local path selection. DRILL, on the other hand, do not need to change any switch hardware or transfer protocols. It investigates a different direction to improve load balancing using a smarter fabric. We also study flow scheduling algorithm in CPU scheduling and switches in data center networks. There are multiple comon-used queue-based methods: First-Come First-Serve, Round-Robin, Shortest Job First and Multi Level Feedback Queue. Bai Wei et al. employ MLFQ into switches in data center networks and proposed an architecture called PIAS to greatly solve the flow scheduling problem. Later on they proposed AuTO, another dynamic online learning algorithm to further address the problem better.

We demonstrate our traffic optimization system in the last chapter. We first introduce the architecture, a hybrid structure which combines distributed load balancing algorithm and centralized flow scheduling algorithm. The centralized component would use stochastic policy gradient algorithm to deal with large flows after given the calculated threshold for judgment, whereas the short flows are directly processed by the MLFQ and transfer to certain paths selected by the DRILL algorithm. The centralized component utilize the Deep Deterministic Policy Gradient algorithm to determine the proper threshold for the MLFQ after training sufficiently (about 8 hours) on GPUs. The metrics regarding the FCT and link utilization comparing with other baseline algorithms are also shown in the chapter to demonstrate a better performance before we conclude the whole thesis in the end.