Layering as Optimization Decomposition: A Mathematical Theory of Network Architectures

报告人：郭晓琳

日 期：2019年4月15日

在线版：

问题的背景

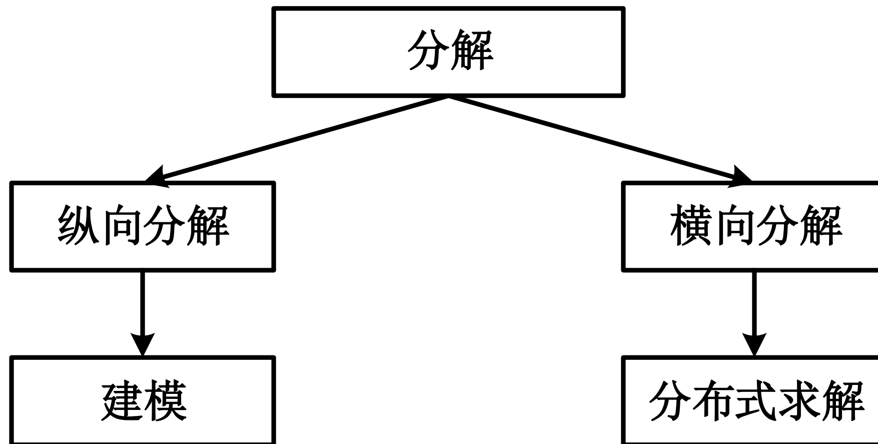
如下图是一个网络的层级结构，在网络中每一层都要实现很多的功能，把网络作为一个整体进行优化是一个非常复杂的事情。

核心思想

通信系统的优化

一个最直接的方式, 通信系统的分析或设计问题可以被建模为最小化成本,或者最大化效用函数的问题，即网络效用最大化问题(NUM)。

求解方法:



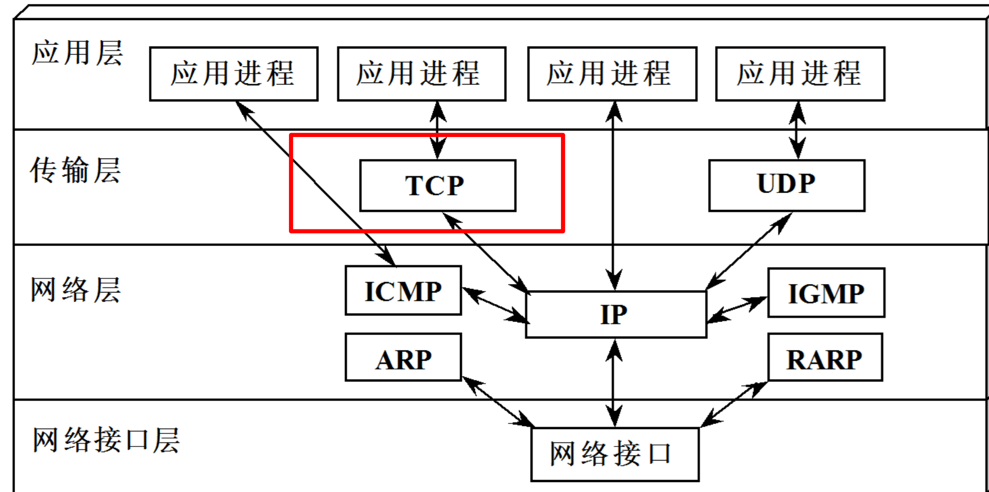


图 2-18 TCP/IP 协议模块关系

建模工具

NUM：

通常，使用NUM作为一个建模的工具，需要考虑下面三个问题：

- 优化目标
- 约束条件
- 优化变量

举例：一个基本的NUM问题，例如TCP：

$$\text{maximize} \sum_{s} U_s(x_s)$$
$$subject\ Rx \leq c$$

网络协议的通用模型

通用的网络效用最大化模型(NUM):

一个一般化的NUM，较为完备的体现网络情况，可以用来解决网络中的绝大部分问题。

效用函数

$$\text{maximize} \sum_s U_s(x_s, P_{e,s}) + \sum_j V_j(w_j)$$

目标函数

$$\begin{aligned} &subject\ to\ Rx \le c(w, P_e), \\ &x \in C_1(P_e), x \in C_2(F)\ or \in \Pi(w), \\ &R \in \mathcal{R},\ F \in \mathcal{F}, w \in \mathcal{W} \end{aligned}$$

约束条件

将其中一些变量变成常量，并指定一些函数依懒项和约束集，可以将通用的NUM转化为一个更具体的NUM，即进行了纵向分解。

效用函数的选择

如何选择NUM中的效用函数U_S？：

- 正向工程：先设计好效用函数，⇨ 再根据函数进行分布式求解。
- 反向工程：由于效用函数不好设计，而分布式算法反而好设计，根据算法，⇨ 得出函数公式。

以下四种情况的任意组合可用于正向工程效用函数的设计：

- 网络中的某些测量指标，如吞吐量、延时、应用流量可以通过效用函数表示。
- 效用可以通过人类心理或行为的模型被定义。
- 效用函数提供了优化资源分配有效性的度量。
- 不同的效用函数产生的最优资源分配满足定义的公平。例如：

$$U(x) = (1-\alpha)^{-1} x^{1-\alpha}$$

模型的设计

目标函数的设计：

- 终端用户效用函数的总和。
- 运营商在网络范围内的成本函数。
- 多目标的优化：在用户目标和运营商目标之间进行权衡。

约束条件的设计：

- 一组在通信基础设施中的物理，技术和经济限制。
- 一组在平衡状态下不能违反的QoS约束。

求解方法

存在的分解技术：

- 原始分解
- 对偶分解

并且，现状是大多数相关文章都是基于对偶分解的分布式算法

。

采用分解技术，是由于凸优化的发展：

近年来，许多高效的算法，例如内点法，推动了非线性凸优化的发展，因此，有许多研究利用非线性优化的最新发展来解决通信系统分析中更为广泛的问题。

例如，TCP的拥塞控制就是被反向工程，隐式解决基本的NUM（凸优化）问题。

TCP 拥塞控制的建模实例

网络效用最大化(NUM)

$$\text{maximize} \sum_{s} U_s(x_s)$$
$$subject\ Rx \leq c$$

x_s：源s的速率

U_s：源s的效用函数

c：链路的容量

假设效用函数 U_s：

- 平滑
- 增函数
- 凹的
- 仅依赖本地速率

$$\boldsymbol{R} = \begin{matrix} & s_1 & s_2 & s_3 \\ l_1 & 1 & 0 & 1 \\ l_2 & 1 & 1 & 1 \\ l_3 & 0 & 1 & 1 \end{matrix}$$

根据最优化理论，在原始问题不好找极值时，可以找其对偶问题的极值。

模型转化

拉格朗日对偶问题：

$$\min \lambda \geq 0 \quad D(\lambda) := \sum_s \max_{x_s \geq 0} \left(U_s(x_s) - x_s \sum_l R_{ls} \lambda_l\right) + \sum_l c_l \lambda_l$$

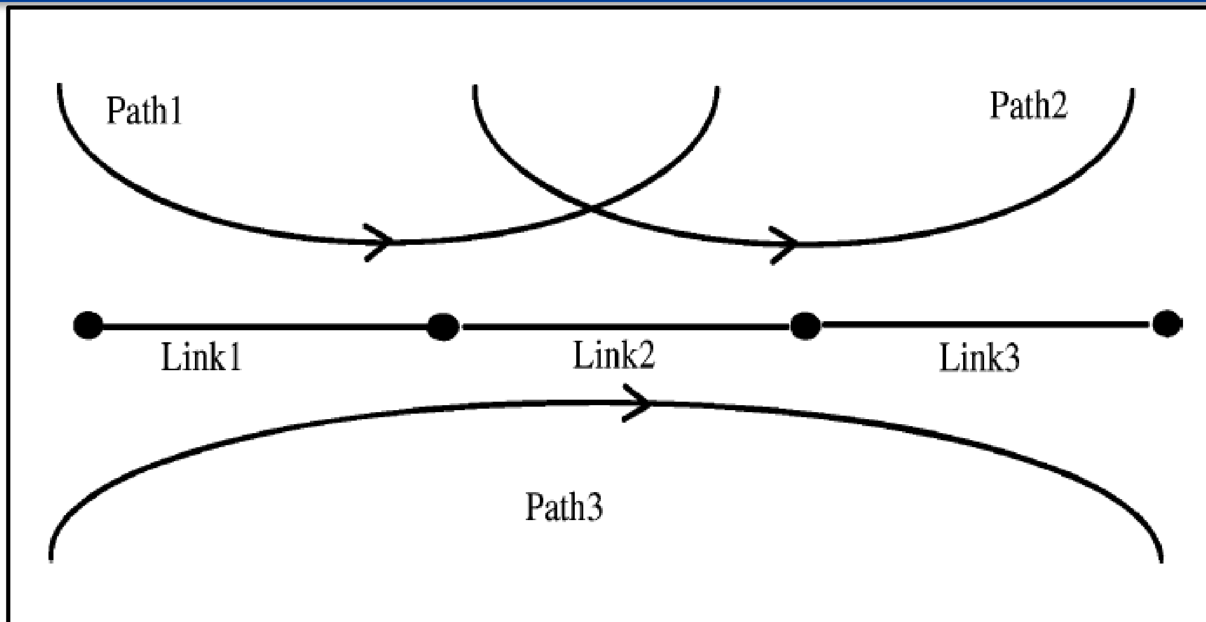
定义:

$$y_l(t) = \sum_s R_{ls} x_s(t)$$ 链路l处各个源速率的总和

$$q_s(t) = \sum_l R_{ls} \lambda_l(t)$$ 源s的端到端价格

对于上述模型，实际解决思路一般都会采用一个**分布式的算法**来解决这个问题。

参数的定义



L 链路

$c = (c_l, l \in L)$链路容量

$L(s) \subseteq L$ 每个源s使用的链路集合.

$\lambda_l(t)$ 拥塞的度量

$$R_{ls} = \begin{cases} 1, & \text{S使用了链路l} \\ 0, & \text{其他} \end{cases}$$

TCP拥塞控制回顾

慢启动：

慢启动初始启动时设置拥塞窗口值（cwnd）为1、2、4或10个MSS。拥塞窗口在每接收到一个确认包时增加，每个RTT内成倍增加。

拥塞避免：

当达到慢启动阈值（ssthresh）时，慢启动算法就会转换为线性增长的阶段，算法控制每个RTT内拥塞窗口只增加1个分段量。

快速重传：

TCP发送方在每发送一个分段时会启动一个超时计时器，如果相应的分段确认没在特定时间内被送回，发送方就假设这个分段在网络上丢失了，需要重发。

一般的拥塞控制分布式算法思路

源更新算法:

- 动态更新速率 $x_s(t)$
- 路径上的价格 $\lambda_l(t)$
- E.g. TCP Reno/Vegas…

链路更新算法:

- 更新价格 $\lambda_l(t)$
- 反馈给源
- E.g.主动队列管理机制(AQM)
- DropTail or RED

拥塞度量:

- 丢包率
- 排队延时

分布式算法(F,G,H)

源s可以观测到:

自己的速率 $x_s(t)$ and 端到端的价格$q_s(t)$

$$x_s(t+1) = F_s(x_s(t), q_s(t))$$

链路l可以观测到:

本地价格 $\lambda_l(t)$ and 流速率 $y_l(t)$

$$\lambda_l(t+1) = G_l(y_l(t), \lambda_l(t), v_l(t))$$

$$v_l(t+1) = H_l(y_l(t), \lambda_l(t), v_l(t))$$

$v_l(t)$ 一些内部变量, 例如链路l的队列长度.

案例一：TCP Reno/RED

TCP Reno 拥塞避免基本思想:

每个RTT(T_s)窗口增加一个包的大小

当发生丢包时窗口大小减半

$$F_s(t+1)=\left[x_s+\frac{1}{T_s^2}-\frac{2}{3}q_s(t)x_s^2(t)\right]^+$$

RED 维持两个内部变量:

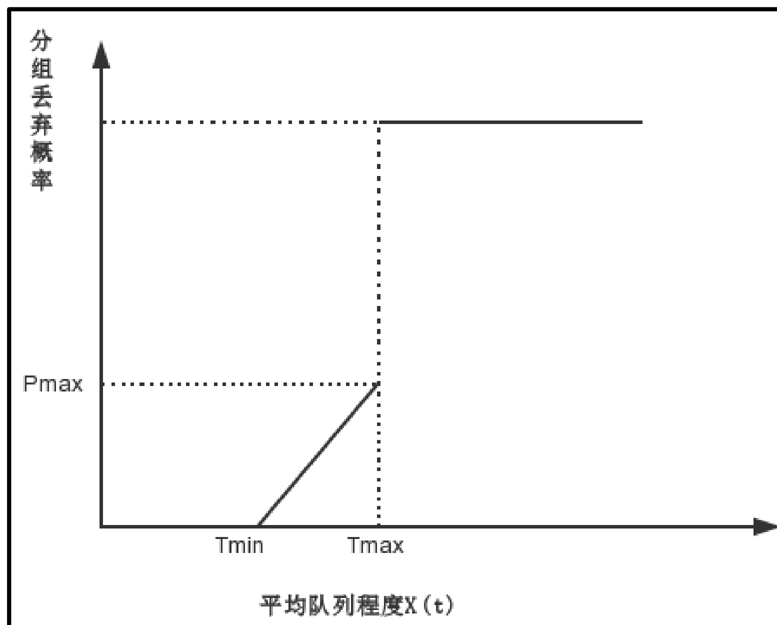
$b_l(t)$: 瞬时队列长度

$r_l(t)$: 平均队列长度

$$b_l(t+1)=[b_l(t)+y_l(t)-c_l]^+$$

$$r_l(t+1)=(1-\omega_1)r_l(t)+\omega_l b_l(t)$$

案例一：TCP Reno/RED



RED 以 $\lambda_l(t)$的概率丢包

$$\lambda_l(t) = \begin{cases} 0, & r_l(t) \le \underline{b}_l \\ \rho_1(r_l(t) - \underline{b}_l), & \underline{b}_l \le r_l(t) \le \overline{b}_l \\ \rho_2(r_l(t) - \overline{b}_l) + M_l, & \overline{b}_l \le r_l(t) \le 2\overline{b}_l \\ 1, & r_l(t) \ge 2\overline{b}_l \end{cases}$$

案例二：TCP Vegas/DropTail

TCP Vegas

- 假设 buffer 大小无限大
- 使用排队延时作为拥塞的度量

$$\lambda_l(t) = b_l(t)/c_l$$

$G_l\,, F_s$ 更新规则

$$\lambda_l(t+1) = \left[\lambda_l(t) + \frac{y_l(t)}{c_l} - 1\right]^+$$

$$x_s(t+1) = x_s(t) + \frac{1}{T_s^2(t)} l\big(\alpha_s d_s - x_s(t) q_s(t)\big)$$

d_s: 传播时延

案例三：Fast/DropTail

Fast

每个源更新其窗口大小根据：

$$W_s(t+1) = \gamma\left(\frac{d_s W_s(t)}{d_s + q_s(t)} + \alpha_s\right) + (1-\gamma)W_s(t)$$

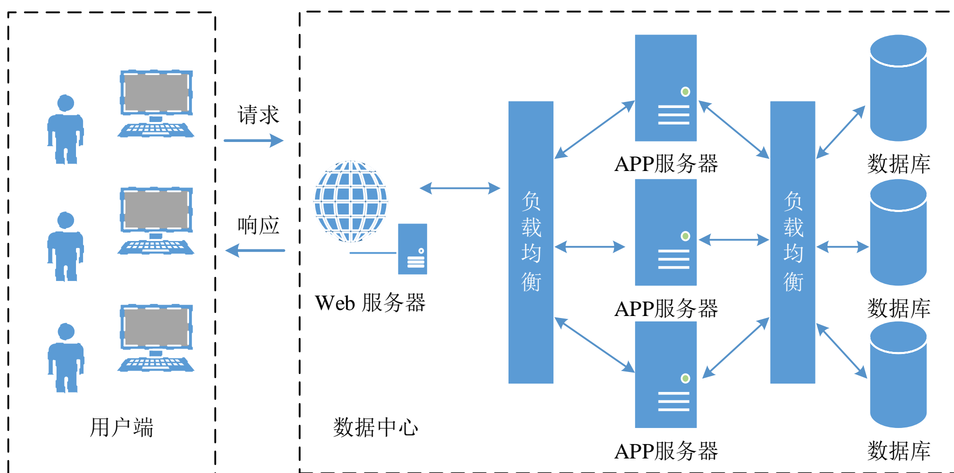
定义$x_s(t)$

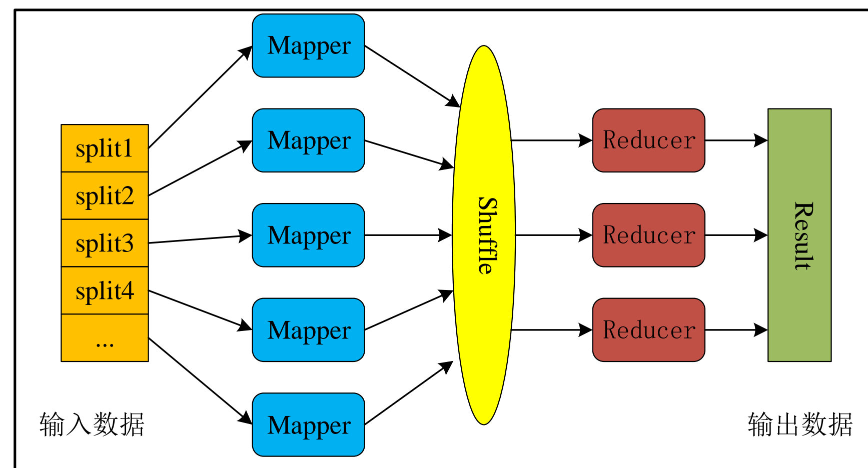
$$x_s(t) = W_s(t)/(d_s + q_s(t))$$

$$\sum_s R_{ls}\frac{W_s}{d_s + q_s(t)} \begin{cases} = c_l, & if\ \lambda_l(t)>0 \\ \le c_l, & if\ \lambda_l(t)=0 \end{cases}$$

背景

随着云计算和虚拟化技术的不断成熟以及云服务的不断丰富，越来越多的应用被部署在云数据中心虚拟化环境下，由于分布式的特点，数据中心应用的性能与网络传输性能密切相关，其网络传输需求通常表示为传输任务。根据应用对网络传输性能的不同需求，可将任务分为两大类：





延迟敏感型任务　　　　带宽敏感型任务

建模（回顾）

任务大小s_k：

$$s_k = \sum_{i=1}^{N} \sum_{j=1}^{N} s_k^{i,j}$$

每条流所分配的带宽$\boldsymbol{r}_{\boldsymbol{k}}^{\boldsymbol{i},\boldsymbol{j}}$满足：

$$r_k^{i,j} = \frac{s_k^{i,j}}{s_k} \times r_k = \frac{s_k^{i,j} \times r_k}{\sum_{i=1}^{N} \sum_{j=1}^{N} s_k^{i,j}}$$

建模

任务$\boldsymbol{task_k}$的完成时间**t**：

$$t = \frac{s_k}{r_k} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} s_k^{i,j}}{r_k}$$

效用函数转化为关于带宽$\boldsymbol{r_k}$的函数：

$$f_k(r_k) = u_k\left(\frac{s_k}{r_k}\right) = \frac{p_k^1}{1 + e^{p_k^2 \times \left(\left(\sum_{i=1}^{N} \sum_{j=1}^{N} s_k^{i,j}\right)/r_k - p_k^3\right)}}$$

建模

采用博弈论中的纳什讨价还价理论：

$$maximize \prod_{k=1}^{K}[f_k(r_k) - f_k^0]$$

$$s.t. \sum_{i=1}^{N} loc_{i,m} \times \left(\sum_{k=1}^{K}\sum_{j=1}^{N}\frac{s_k^{i,j}}{s_k} \times r_k\right) \le C, m \in \{1,2,\dots,M\}$$

$$\sum_{i=1}^{N} loc_{i,m} \times \left(\sum_{k=1}^{K}\sum_{j=1}^{N}\frac{s_k^{j,i}}{s_k} \times r_k\right) \le C, m \in \{1,2,\dots,M\}$$

上述最优化问题在数学上等价于求解：

$$maximize \sum_{k=1}^{K} ln[f_k(r_k) - f_k^0]$$

求解方法

算法 4-1: 效用感知的任务带宽分配算法

1 **输入**: 任务流量矩阵$s_k^{i,j}, k \in \{1,2,\dots,K\}, i,j \in \{1,2,\dots,N\}$
2 sigmoid 效用函数参数p_k^1, p_k^2, p_k^3
3 接入链路上下行带宽$C_m^{ul}, C_m^{dl}, m \in \{1,2,\dots,M\}$
4 **输出**: 为各任务中各条流分配的带宽$r_k^{i,j}, k \in \{1,2,\dots,K\}, i,j \in \{1,2,\dots,N\}$
5 根据效用函数计算各任务所能达到的最大效用f_k^{max}
6 计算每个任务的总大小s_k
7 t = 0
8 **while** t <= T
9 计算每个任务的效用值$f_k^* = (1-\theta)^t \times f_k^{max}$
10 根据当前的f_k^*得到为每个任务分配的总带宽r_k^*
11 根据r_k^*计算各任务中每条流的带宽$r_k^{i,j} = s_k^{i,j}/s_k \times r_k^*$
12 **if** $r_k^{i,j}$满足接入链路带宽容量约束
13 将剩余带宽分配给单位带宽效用最大的任务
14 **Break**
15 **End**
16 **End**

分布式求解

集中式的求解算法:

- 通过计算当前任务的总量，为各任务流分配带宽
- 再将计算结果发送给每个任务
- 必须获得全局的网络信息

源s可以观测到:

自己的速率 $x_s(t)$ and 端到端的价格$q_s(t)$

$$x_s(t+1) = F_s(x_s(t), q_s(t))$$

Coflow的特点:

- 只有最慢的流完成，任务才算完成
- 所以任务的各流之间应该有信息的交互，确定出瓶颈流
- 在已知一个任务信息的前提下，各流之间互发massage消息交互

分布式求解

分布式求解算法:

- 源根据当前速率，链路价格，massage信息m更新速率

$$x_s(t+1) = F_s(x_s(t), q_s(t), \mathrm{m})$$

算法: 任务流速率更新算法

输入: 任务流量矩阵$s_k^{i,j}, k \in \{1,2,\dots,K\}, i,j \in \{1,2,\dots,N\}$

接入链路上下行带宽$C_m^{ul}, C_m^{dl}, m \in \{1,2,\dots,M\}$

输出:各任务中各条流更新后的带宽$x_s^{i,j}, s \in \{1,2,\dots,K\}, i,j \in \{1,2,\dots,N\}$

1)源根据当前速率，链路价格，massage 信息 m 更新速率

$$x_s(t+1) = F_s(x_s(t), q_s(t), \mathrm{m})$$

2)链路根据当前价格，链路上流速率，排队队列长度更新价格

$$\lambda_l(t+1) = G_l(y_l(t), \lambda_l(t), v_l(t))$$

3)每个源更新速率后根据任务信息互发 massage 消息传递速率

End