排队论及其应用浅析

——Queueing Theory

网易杭研： 何登成

新浪微博：何_登成

H1391

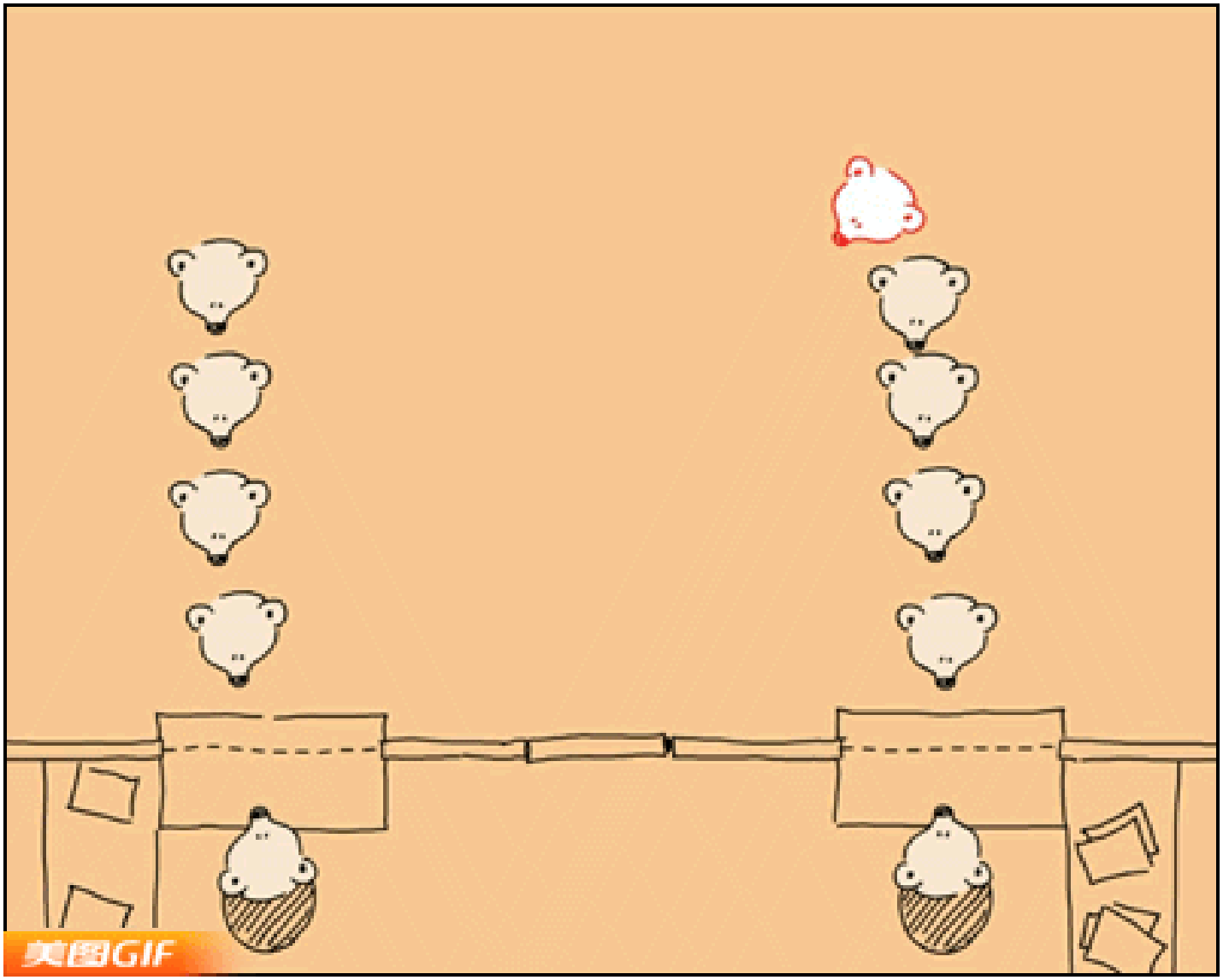
大纲(一)

- 排队
  - 现实生活中
  - 计算机领域
- 排队论浅析
  - 排队论问题分析
  - Operational Law
    - Utilization Law
    - Little’s Law
  - D. G. Kendall
  - Little’s Law
  - Erlang’s Formula

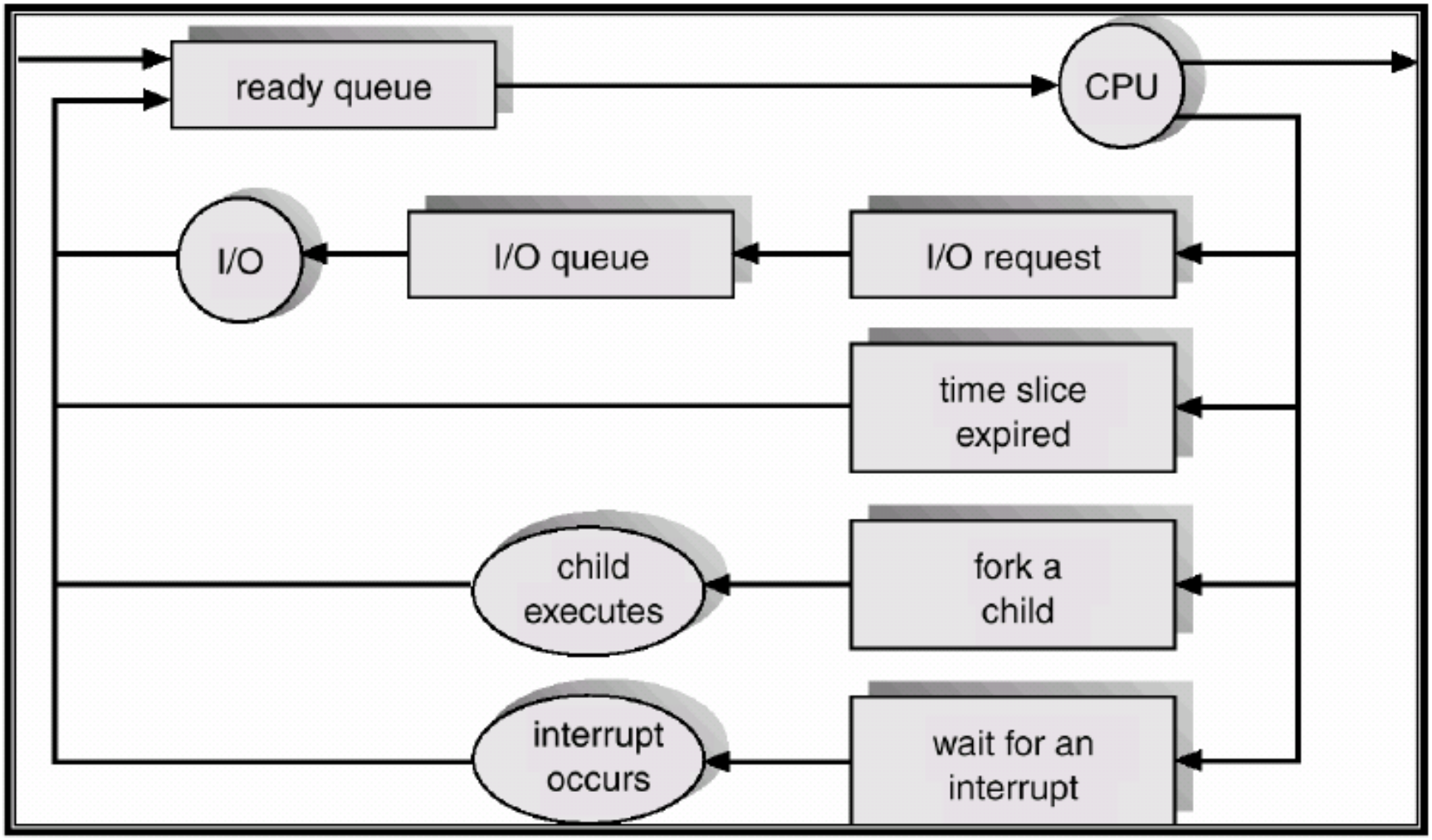
大纲(二)

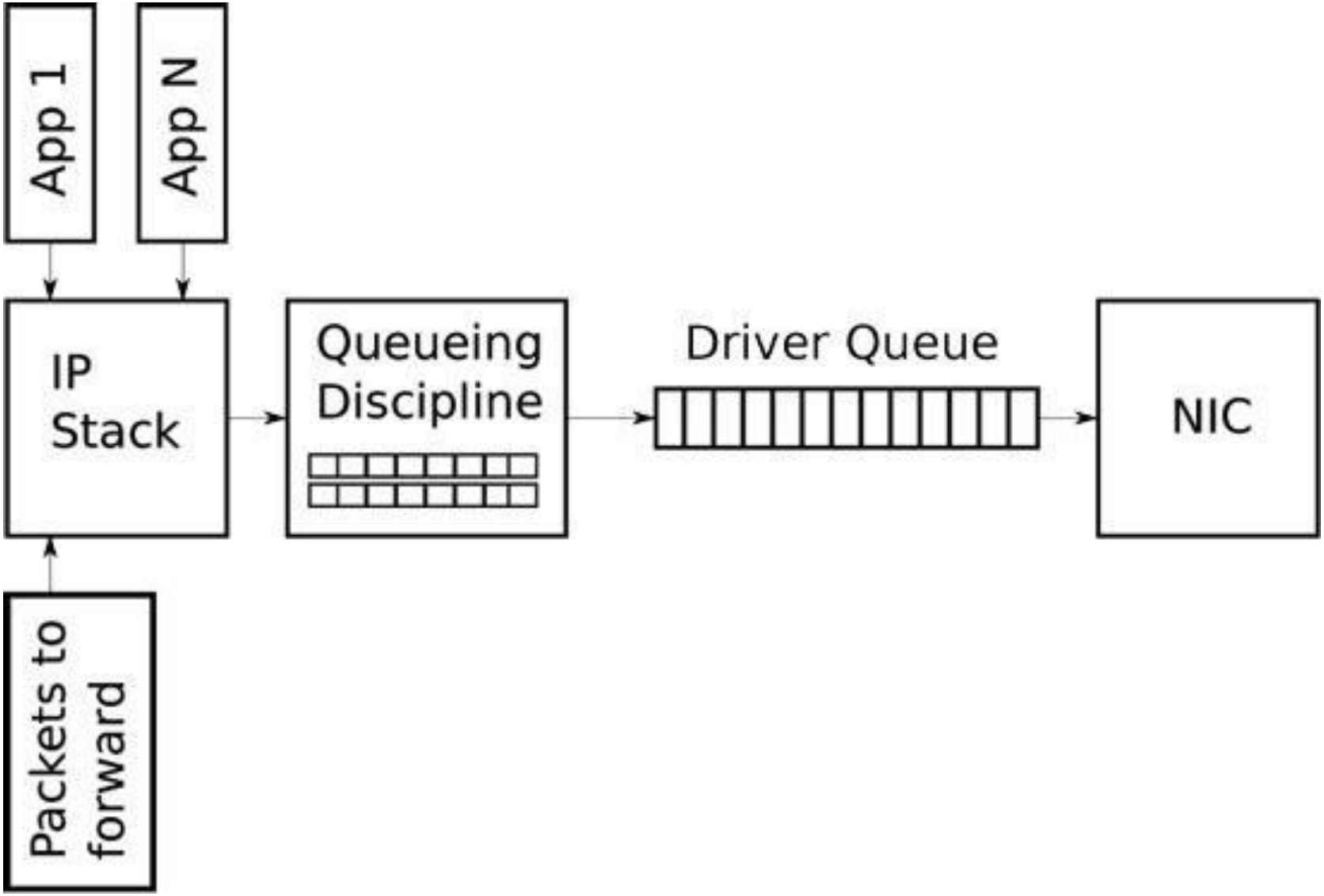
- 排队论应用分析
  - 更好的理解系统监控
    - Linux I/O Stats
  - 更好的架构选型
  - 排队论与Performance
    - 什么是Performance？
    - 何谓平衡的系统？
    - 如何监控系统？
    - 如何进行高性能程序设计？
  - 容量规划（入门）


珠江新城
体育西路


美图GIF


Boots


ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt


App 1
App N
IP Stack
Queueing Discipline
Driver Queue
NIC
Packets to forward

排队问题

- 在排队过程中，人们会关心哪些问题？
  - 什么时候过去，排队的时间会比较短？
  - 不排队的话，需要多少时间？
  - 是否需要排队？
  - (不)包括正在被服务的人，我前面一共还有几人？
  - 排到我需要多少时间？等我服务结束，又需要多少时间？
  - 这次排队是否合算？
  - 队伍长队达到什么程度，我就会放弃本次排队？
  - 系统有多少个窗口？是否已经高负荷运转了？

- 无论是现实中的排队，电话通讯领域的排队，还是计算机领域的排队，最后都能够抽象化为一个经典的理论——排队论(Queuing Theory)。

- 排队论起源于20世纪初的电话通话。1909—1920年丹麦数学家、电气工程师爱尔兰（A.K.Erlang）用概率论方法研究电话通话问题，从而开创了这门应用数学学科，并为这门学科建立许多基本原则 ...

- Erlang在电话通讯领域，解决了两个基本问题：电话损失率(Erlang-B Formula)，电话等待概率(Erlang-C Formula)。在介绍这些之前，让我们先来认识D. G. Kendall与J. D. C. Little ...

Queueing Theory

- 排队论用于解决什么问题？
  - 核心问题
    - 对用户来说：　　响应时间 (满意度)
    - 对服务提供者来说：　　利用率 (成本)
    - 在保障用户满意度的前提下，最大限度的控制成本，充分挖掘系统的潜力。

Queueing Theory(形式化)

- 什么时候过去，排队的时间会比较短？
  - 达到请求的分布；单位时间平均到达请求数量： λ
- 不排队的话，需要多少时间？
  - 服务时间 (Service Time)；单位时间平均完成的服务数量：μ
- 是否需要排队？

- (不)包括正在被服务的人，我前面一共还有几人？
  - 平均队列长度 L (不包括正在服务的人：L_q)
- 排到我需要多少时间？等我服务结束，又需要多少时间？
  - Wait Time W_q vs Reponse Time W or R
- 队伍长队达到什么程度，我就会放弃本次排队？
  - 服务丢失率： P_B ；队列最大长度；
- 系统有多少个窗口？是否已经高负荷运转了？
  - 窗口数量；利用率 U ；

Utilization Law

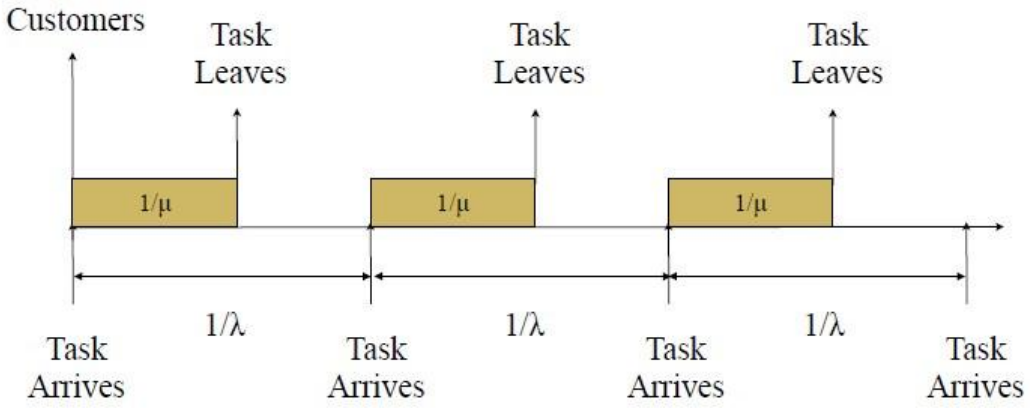
- Utilization Law

  $U = \lambda / \mu$

  - 排队系统中，Server的利用率，为平均到达速率与平均服务速率的比值；

  – Utilization Law解读
    - 保持Server的平均服务速率不变，平均到达速率越高 -> Server利用率越高；
    - 保持平均到达速率不变，Server的平均服务速率越高 -> Server利用率越低；

  – 一个简单的证明 (求下面这个系统的利用率)



Little's Law

- Little's Law

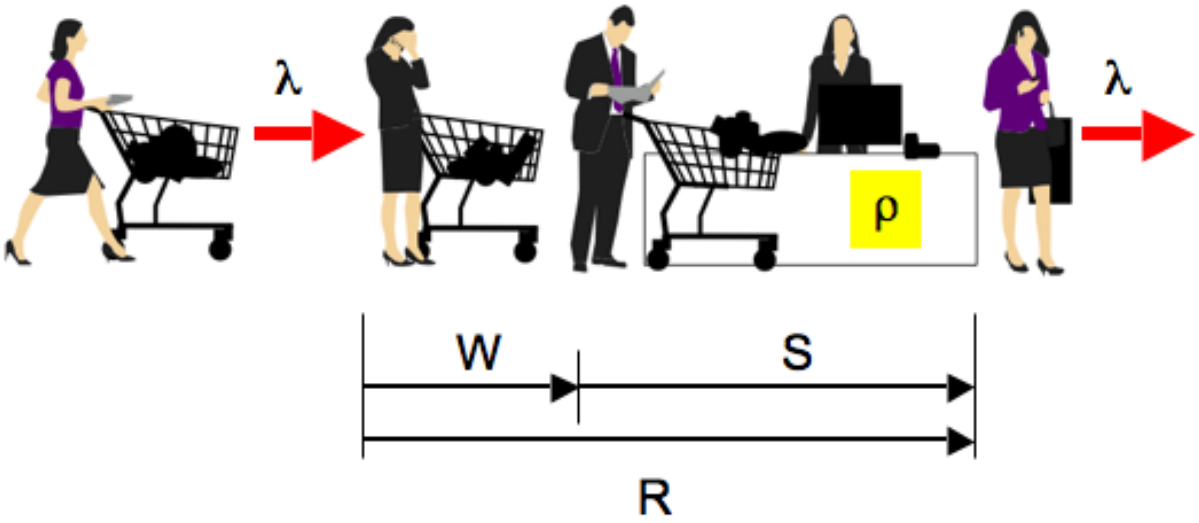
  $$L = \lambda W$$

  - 排队系统中，队列长度(包括正在接受服务的人) = 到达率 * 平均响应时间
  - 由J.D.C. Little在1961年的论文【A Proof for the Queuing Formula L = λW】中给出形式化的证明；
  - 证明：略
  - 变种： $L_q = \lambda W_q$ 排队长度 = 到达率 * 平均等待时间 （ $W_q = W - \frac{1}{\mu}$ ）

  – Little's Law解读

  - Little's Law/Utilization Law，均属于Operational Laws中的一种；
  - 平均响应时间越长，队列长度也越长；
  - 单位时间到达的请求越多，队列长度越长；
  - 符合日常的理解；
  - 平均请求数量 λ，已知 L, L_q, W, W 中的任何一个，都可以计算出另外三个；

Little’s Law（续）



- Little’s Law的应用场景
  - 最基础的Law，对请求到达/离开的分布没有特殊要求；
  - 超市、银行、KFC等管理
    - Six Sigma（六西格玛管理法）
  - 系统性能监控
  - …

Little's Law（例1）

- 一个小酒馆，客户平均访问频率为40人/小时；
  - 客户在小酒馆的平均消费时间是15分钟；
- 问题：请问小酒馆的平均客户数量是多少？
- 解答：
  - λ ：40人/小时
  - W ：0.25小时/人
  - $L = \lambda W$ ：40 * 0.25 = 10

Little's Law（例2）

- 一证券经纪公司，其网站有110万注册用户。高峰期，同时有20000用户在线。同时观察到，在高峰期网站一小时处理360万笔业务。

- 问题：请问处理每笔业务的响应时间是多少？

- 解答：
  - λ ：3600000 / 3600 = 1000笔/秒
  - L ：20000
  - $W = L / \lambda$ ：20000 / 1000笔/秒 = 20秒/笔

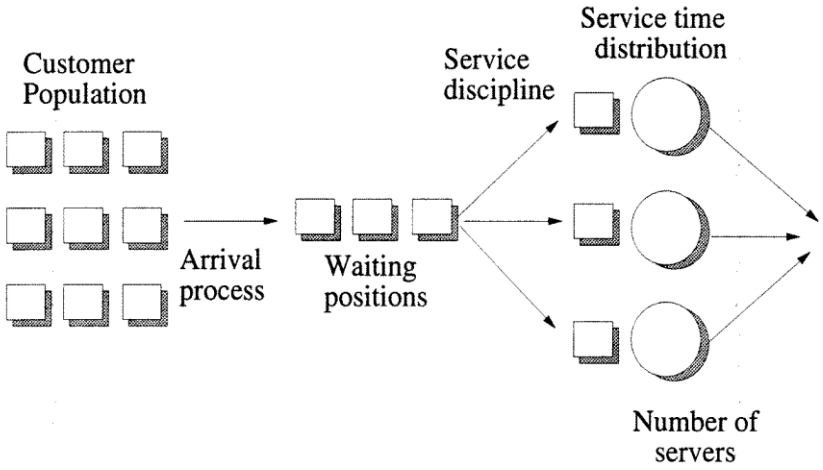
Little's Law（例3）

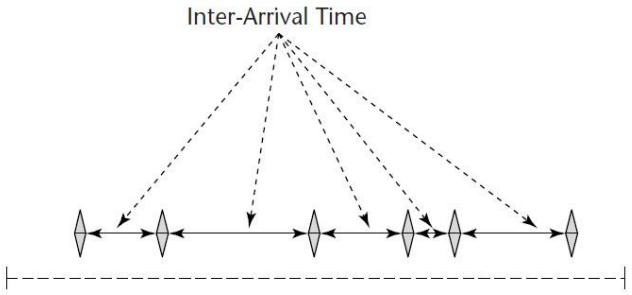
```
hzhedengcheng@app-66:~$ iostat -xm 60 10
Linux 3.2.0-3-amd64 (app-66.photo.163.org)      10/12/13        _x86_64_        (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          13.92    0.00    3.08    2.99    0.00   80.01

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.04   128.68    2.41    9.81     0.03     0.63   110.88     0.21   17.42    2.22   21.15   4.06   4.96
sdb               0.21   266.13  315.82  592.64     3.04    11.18    32.05     0.24    0.26    0.69    0.04   0.20  17.76

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           8.38    0.00   10.68    0.43    0.00   80.51

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     1.25    0.05    1.48     0.00     0.01    14.70     0.01    9.09    8.00    9.12   5.91   0.91
sdb               0.00     1.13  173.83   39.22     1.49     4.99    62.26     2.49   11.67    0.17   62.67   0.28   6.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          13.15    0.00   12.12    7.10    0.00   67.63

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     1.73    0.27    1.60     0.01     0.01    23.79     0.02   10.64    4.00   11.75   4.11   0.77
sdb               0.00    16.03 2555.38 1218.13    20.42    31.34    28.10    13.49    3.57    0.65    9.71   0.19  70.23

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          17.65    0.00   12.52   10.84    0.00   58.99

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00    19.75    0.00    1.72     0.00     0.08   100.04     0.02    9.51    0.00    9.51   4.66   0.80
sdb               0.00    20.73 4375.38 1500.22    34.18    26.19    21.04    11.27    1.92    0.64    5.66   0.15  89.77
```

- 公式：
  - avgqu-sz = (r/s + w/s) * await / 1000 = (173.83+39.22)*11.67/1000 = 2.49
  - %util = (r/s + w/s) * svctm / 1000 = (173.83+39.22)*0.28/1000 = 6%

Kendall Notation

- 六元组
  - A/S/c/K/N/D
    - 现实中所有系统的形式化模型；
  - 解读
    - **A**：到达请求的分布
      - M(exponential/Markov)；D(Deterministic)；G(General)；...
    - **S**：服务时间的分布
      - M(exponential/Markov)；D(Deterministic)；G(General)；...
    - **c**：排队系统中Server的数量
      - 1 到 无限
    - K：排队系统中队列最大长度(包括正在服务的队列)
      - 1 到**无限**
    - N：请求总数量
      - 有限 vs **无限**
    - D：请求的服务调度策略
      - **FCFS/FIFO**; LCFS/LIFO; SIRO; ...





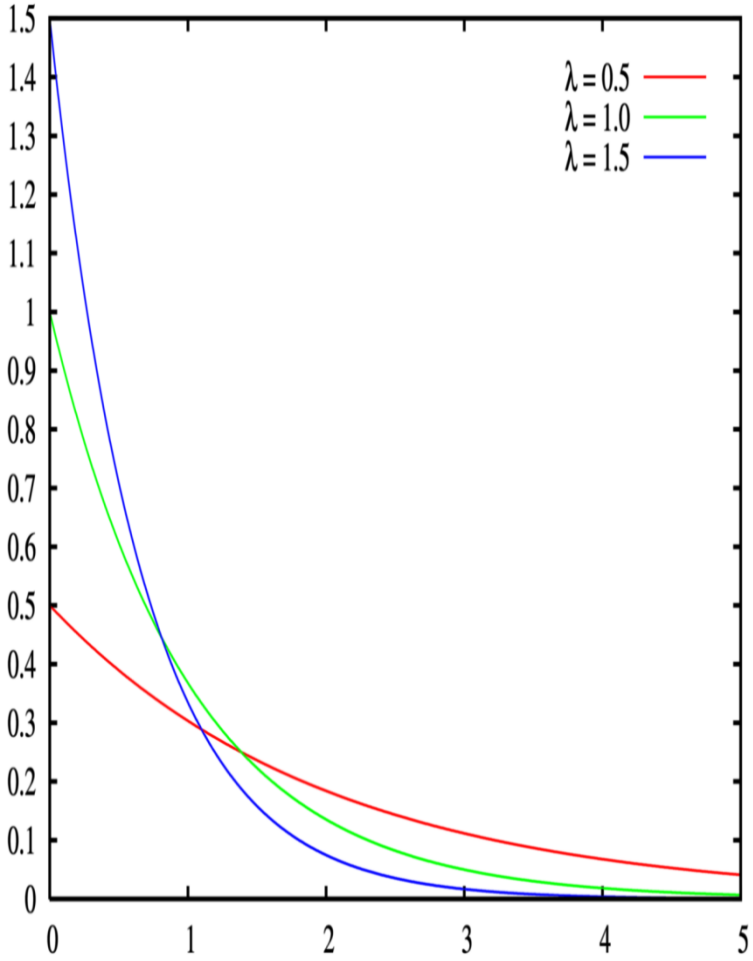
Kendall Notation(举例)

- M/D/5/40/200/FCFS
  - 到达请求的时间间隔指数分布/平均到达请求数量泊松分布 (M)
  - 服务时间确定 (D)；
  - 一个有5个Servers (5)；
  - 5个Servers，一共有40个Buffers (5个服务窗口，35个等待队列)；
  - 一共有200个请求；
  - 服务调度策略为先到先服务 (FCFS)；

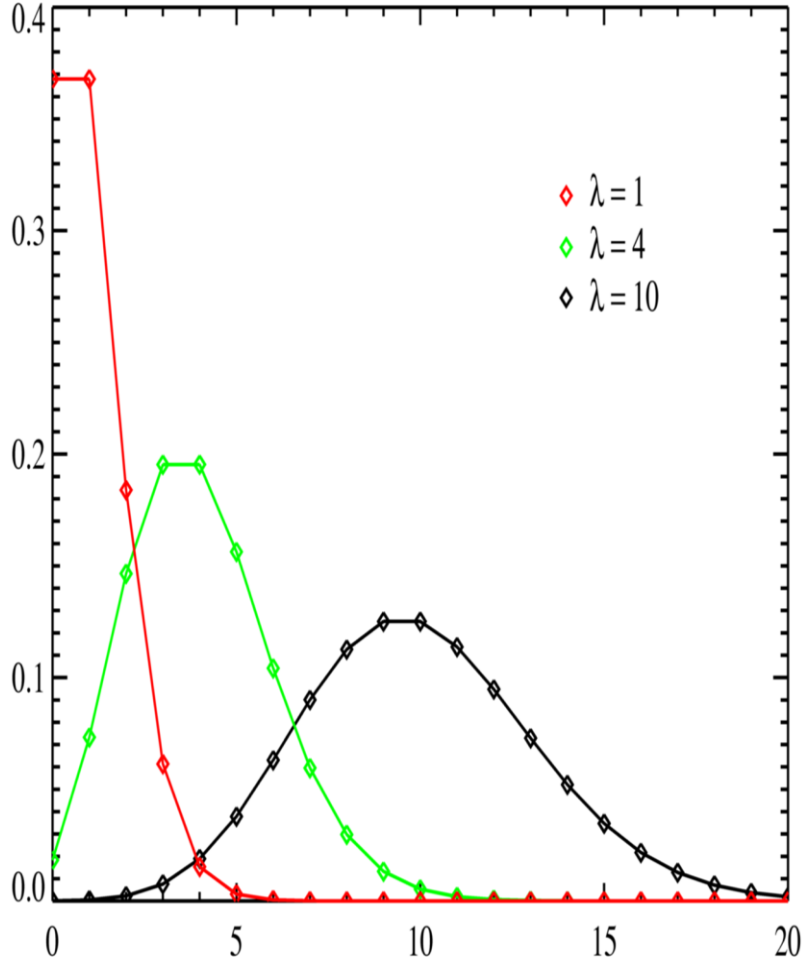
- M/M/1
  - 到达请求的时间间隔指数分布/平均到达请求数量泊松分布 (M)
  - 服务时间指数分布 (M)
  - 一个Server (1)
  - 无限等待队列长度 (默认)
  - 无限请求数量 (默认)
  - 先到先服务的调度策略 (默认)

概率分布

- 指数分布



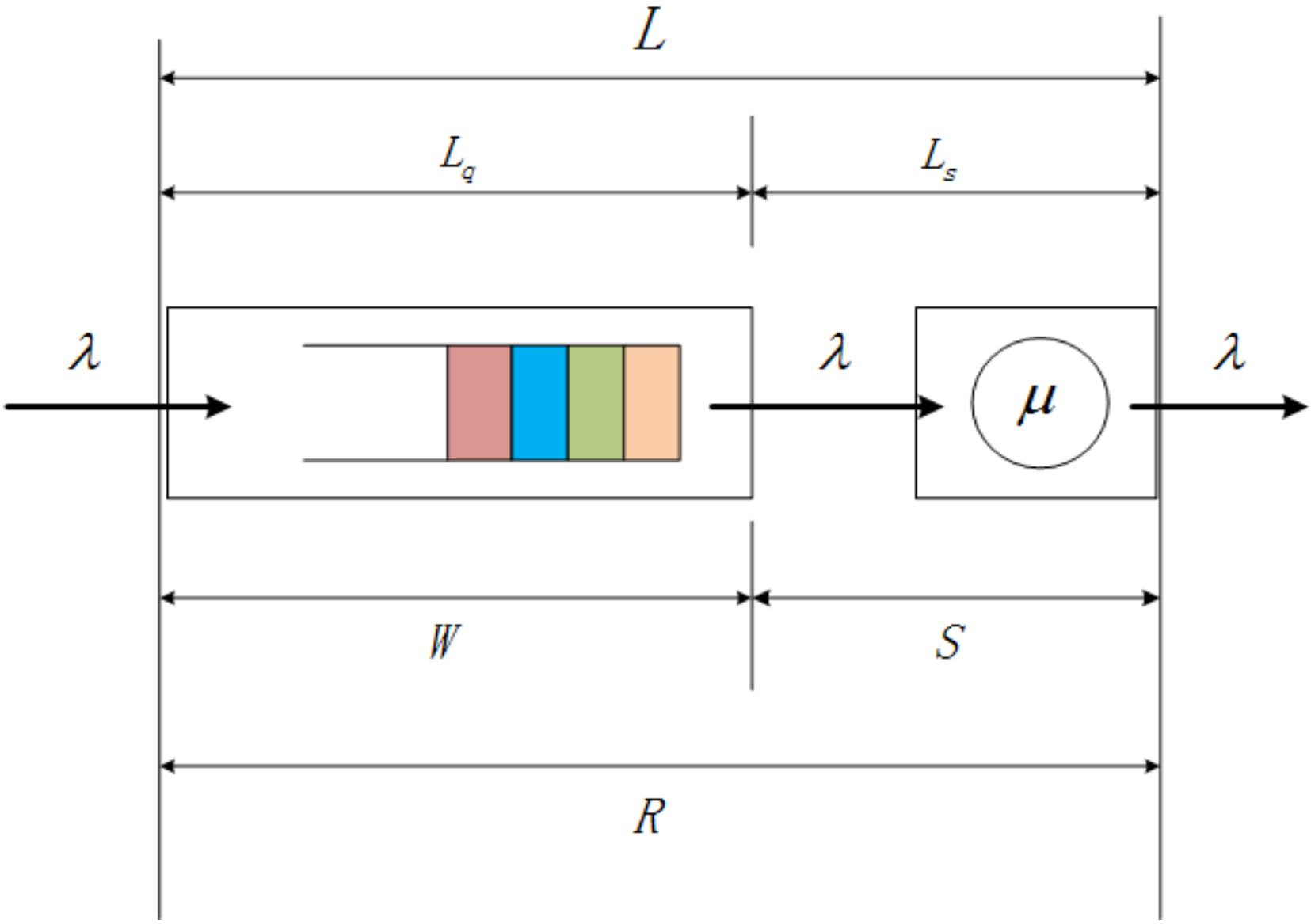
泊松分布



M/M/1

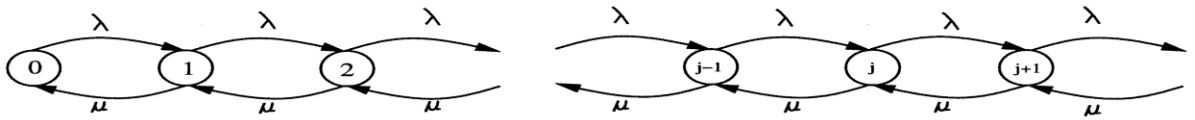
- 模型分析
  - M/M/1模型，是排队论中最典型、应用领域最广的模型；

- 给定此模型，需要解决什么问题？(仍旧是那些老问题)
  - 已知：
    - 单位时间平均达到的请求数量： λ
    - 单位时间平均处理能力： μ
  - 未知：
    - Server的利用率是多少： U
    - 平均服务时间是多少： S
    - 平均等待时间是多少： W
    - 平均响应时间是多少： R
    - 服务队列长度是多少： L_s
    - 等待队列长度是多少： L_q
    - 总队列长度是多少： L
    - Server空闲概率是多少： P_0
    - 一个请求，需要等待的概率是多少： P_b


L
L_q
L_s
λ
λ
λ
μ
W
S
R

M/M/1（原理1）

- 状态转移（birth-death process）



  - 马尔科夫过程(Markov Process)
    - 每一个状态，只跟他前后两个状态有关；
  - 每个状态，标识排队系统中有多少请求 L
    - j状态，有 λ 的概率来一个请求，进入j+1状态；同样，有 μ 的概率处理一个请求，进入j-1状态；
  - 每个状态，都有一个概率 P_n

- 状态转移公式
  - 每个状态的转入 = 每个状态的转出 (稳态系统：Steady State)

$$P_0\lambda = P_1\mu$$

$$P_1(\lambda + \mu) = P_0\lambda + P_2\mu$$

$$P_j(\lambda + \mu) = P_{j-1}\lambda + P_{j+1}\mu$$

M/M/1（原理2）

- 计算各状态概率

$$P_1 = \frac{\lambda}{\mu} P_0$$

$$P_2 = (\frac{\lambda}{\mu})^2 P_0$$

$$P_n = (\frac{\lambda}{\mu})^n P_0$$

- 利用率
  - 定义 $\rho = \frac{\lambda}{\mu}$ ，既为Server的利用率（Utilization Law）；

- Steady State（稳态）
  - 若 $\frac{\lambda}{\mu}$ > 1，到达速度快于处理速度，系统中的累积的未处理请求会越来越长，系统会最终崩溃 → 非稳态系统；
  - 稳态系统：$\frac{\lambda}{\mu}$ < 1 (= 1是特殊情况，也是不稳定的，后续会提到)
  - 稳态系统：请求不会累计，系统能够长时稳定运行：$\sum_{i=0}^{\infty} P_i = 1$

M/M/1（问题解答1）

- Server的利用率是多少？

$$U = \rho = \frac{\lambda}{\mu}(\lambda < \mu)$$

- 平均服务时间是多少？

$$S = \frac{1}{\mu}$$

- 服务队列长度是多少？

$$L_s = \lambda \bullet \frac{1}{\mu} = \frac{\lambda}{\mu}$$

- Server空闲概率是多少？

$$P_0 = 1 - U = 1 - \frac{\lambda}{\mu}$$

- 一个请求，需要等待的概率是多少？

$$P_b = 1 - P_0 = \frac{\lambda}{\mu}$$

M/M/1（问题解答2）

- 总队列长度是多少？

$$L = \sum_{i=0}^{\infty} iP_i = \sum_{i=1}^{\infty} i(1-\rho)\rho^i = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$$

- 等待队列长度是多少？

$$L_w = \sum_{i=1}^{\infty} (i-1)P_i = \frac{\rho^2}{1-\rho}$$

- 平均响应时间是多少？

$$R = \frac{L}{\lambda} = \frac{1}{\mu}\frac{1}{1-\rho} = \frac{1}{\mu-\lambda}$$

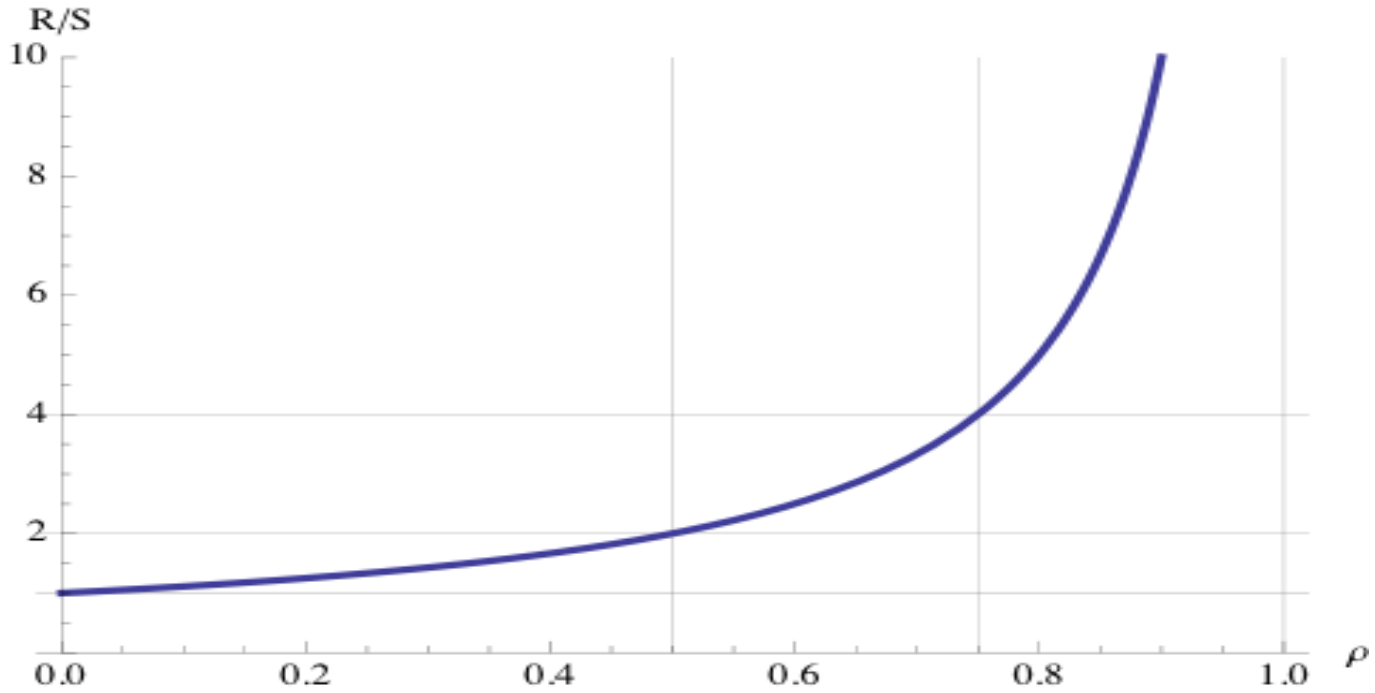
- 平均等待时间是多少？

$$W = \frac{L_w}{\lambda} = \frac{1}{\mu}\frac{\rho}{1-\rho} = R - S$$

M/M/1

- 利用率与响应时间之间的关系

$R = \frac{1}{\mu}\frac{1}{1-\rho}$（其中：$\frac{1}{\mu}$ 为常数）



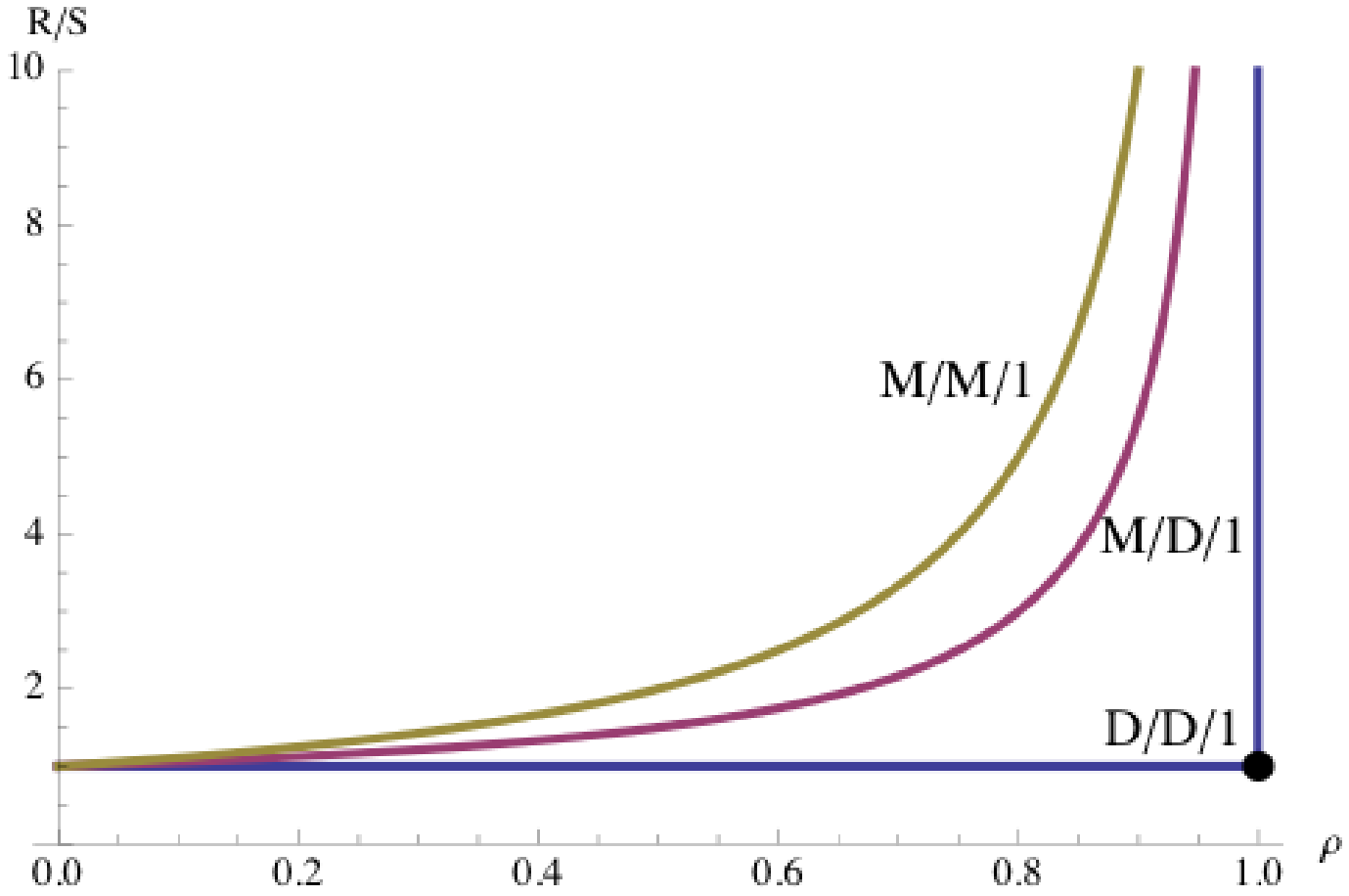
M/M/1（解读）

- OLTP模型
  - 请求随机，服务时间随机；
- 请求速度 < 处理速度：稳态系统
- 资源利用率与响应时间的关系
  - 控制资源利用率；

D/D/1

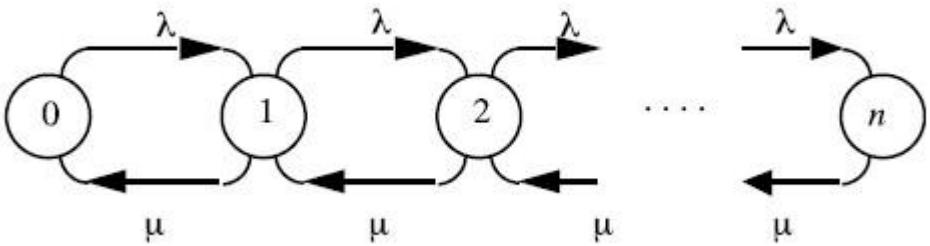
- 模型分析
  - D/D/1模型，是排队论中最简单的模型；
    - 到达请求的时间间隔分布：确定的；
    - 服务时间的分布：确定的；
- OLAP模型
  - 请求确定，服务时间确定，定时任务；
- 资源利用率与响应时间的关系
  - 可以人为调度各任务，做到完全的串行化；
  - 资源利用率可以达到100%，而不影响响应时间；

M/M/1 vs M/D/1 vs D/D/1



M/M/1/k

- Finite Buffer
  - 等待队列数量有限
    - 1个服务窗口，K-1个等待Buffers；
  - 若当前已有K个请求，则第K+1个请求被丢弃
- 状态转移(birth-death process)



M/M/1/k

- 现实意义
  - 大部分现实中的系统，都是有最大等待队列限制的；
  - 哪怕是没有等待队列限制的系统，用户发现队列长度过长时，也会主动放弃等待；
- 最重要的问题
  - 请求到达时，发现队列已满，而被丢弃的概率是多少？ P_B

$$P_B = P_k = \frac{1-\rho}{1-\rho^{n+1}}\rho^n$$

  - 问题：
    - 已知 ρ = 0.5，若要让请求被丢弃的概率小于0.1%，那么系统需要支持多大的队列k？

M/M/1/k

- 问题解答

$$P_B \le 10^{-3}$$

$$\frac{1-0.5}{1-0.5^{k+1}}0.5^k = \frac{0.5^{k+1}}{1-0.5^{k+1}} \le 10^{-3}$$

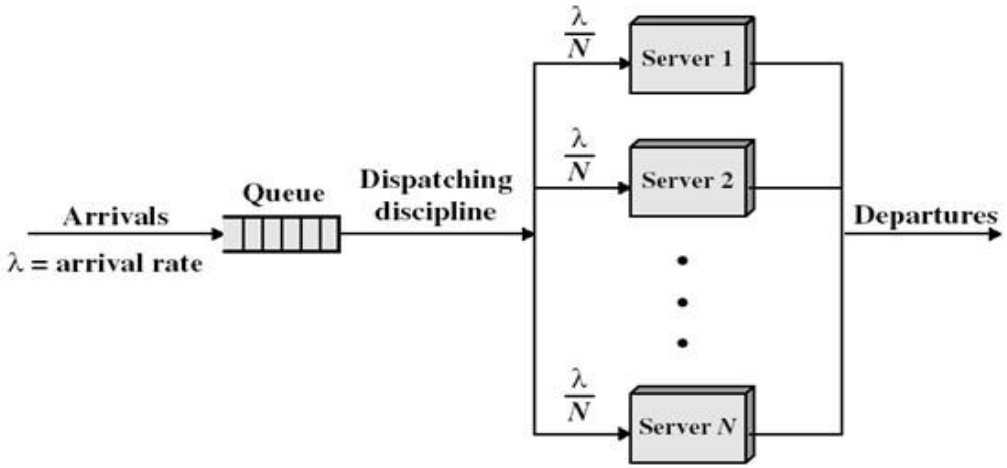
$$k+1 \ge 9.96$$

$$k = 9$$

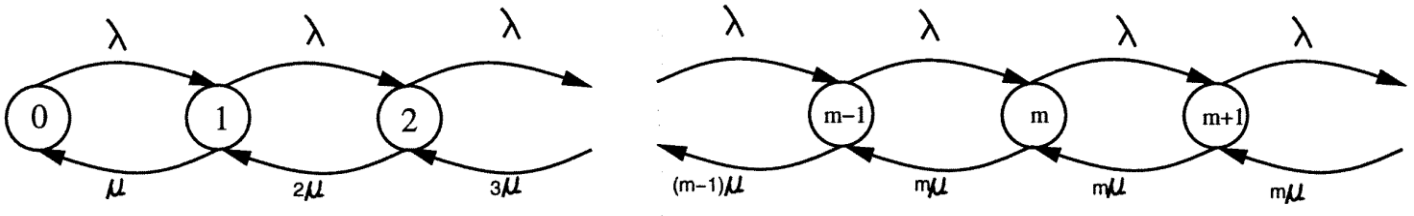
- 深入分析
  - M/M/1/k系统，不要求 $\rho < 1$，因为超过系统限制的请求，都被丢弃了；
  - M/M/1/k系统，实际进入系统的平均请求数量为 $\lambda' = \lambda(1-P_B)$

M/M/m

- M/M/m系统
  - m个Servers，一个排队队列（无限），每个Server的处理能力相同；



- 状态转移（birth-death process）



M/M/m

- M/M/m系统公式

$$U = \frac{\lambda}{\mu}$$

$$\rho = \frac{\lambda}{m\mu} < 0$$

$$R = \frac{1}{\mu}(1 + \frac{c(m, \rho)}{m(1 - \rho)})$$

其中，

$c(m, \rho)$

$$c(m, \rho) = \frac{(m\rho)^m}{m!} / [(1 - \rho)\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!}]$$

  - Erlang-C Formula
  - 代表了有m（或以上）个请求正在处理的概率。此时，新的请求进入系统需要等待；

- 响应时间R

$$R = \frac{1}{\mu}(1 + \frac{c(m, \rho)}{m(1 - \rho)}) \approx \frac{1}{\mu}\frac{1}{1 - \rho^m}$$

M/M/m

- 应用：
  - 一个电话呼叫系统，每分钟平均有10个电话打入，每个电话平均持续1分钟。
  - 电话的呼入间隔与服务时间均服从正态分布(M)。

- 问题：
  - 请问需要提供多少话务员，才能够满足用户的需求？使得用户不至于因等待而放弃。

- 分析：
  - M/M/m系统，话务员即为Server；
  - λ ：10个/分钟
  - μ ：1分钟/个
  - ρ ：$\rho = \frac{\lambda}{m\mu} < 1$ → m > 10

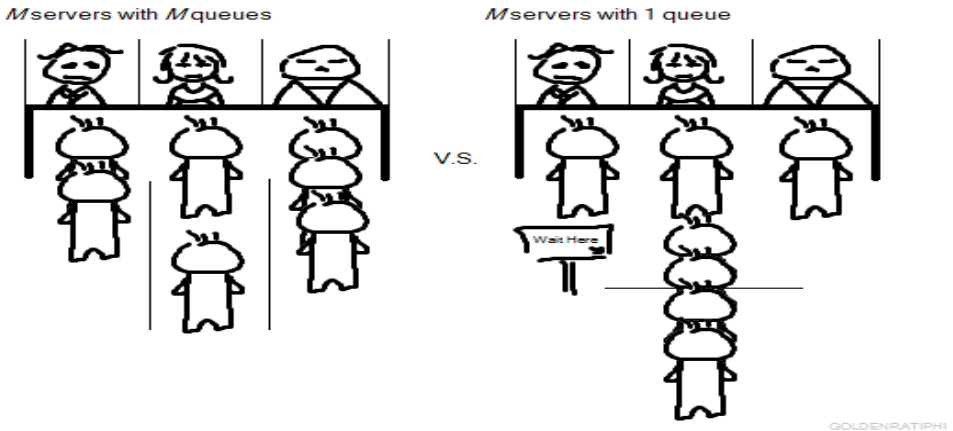
M/M/m

- 解答

衡量指标	$M/M/11$	$M/M/12$	$M/M/13$
等待队列L_q	6.821	2.247	0.951
等待概率W_q	0.682	0.225	0.095
利用率E	0.909	0.833	0.767

m M/M/1 vs M/M/m

- m M/M/1 vs M/M/m
  - 均为m个Servers，单队列与多队列系统，哪个更好？



- m M/M/1 $R = \frac{1}{\mu}\frac{1}{1-\rho}$
- M/M/m $R \approx \frac{1}{\mu}\frac{1}{1-\rho^m}$
- 随着 ρ（$\rho = \frac{U}{m}$） 的增大，响应时间哪个增加的更快？

m M/M/1 vs M/M/m

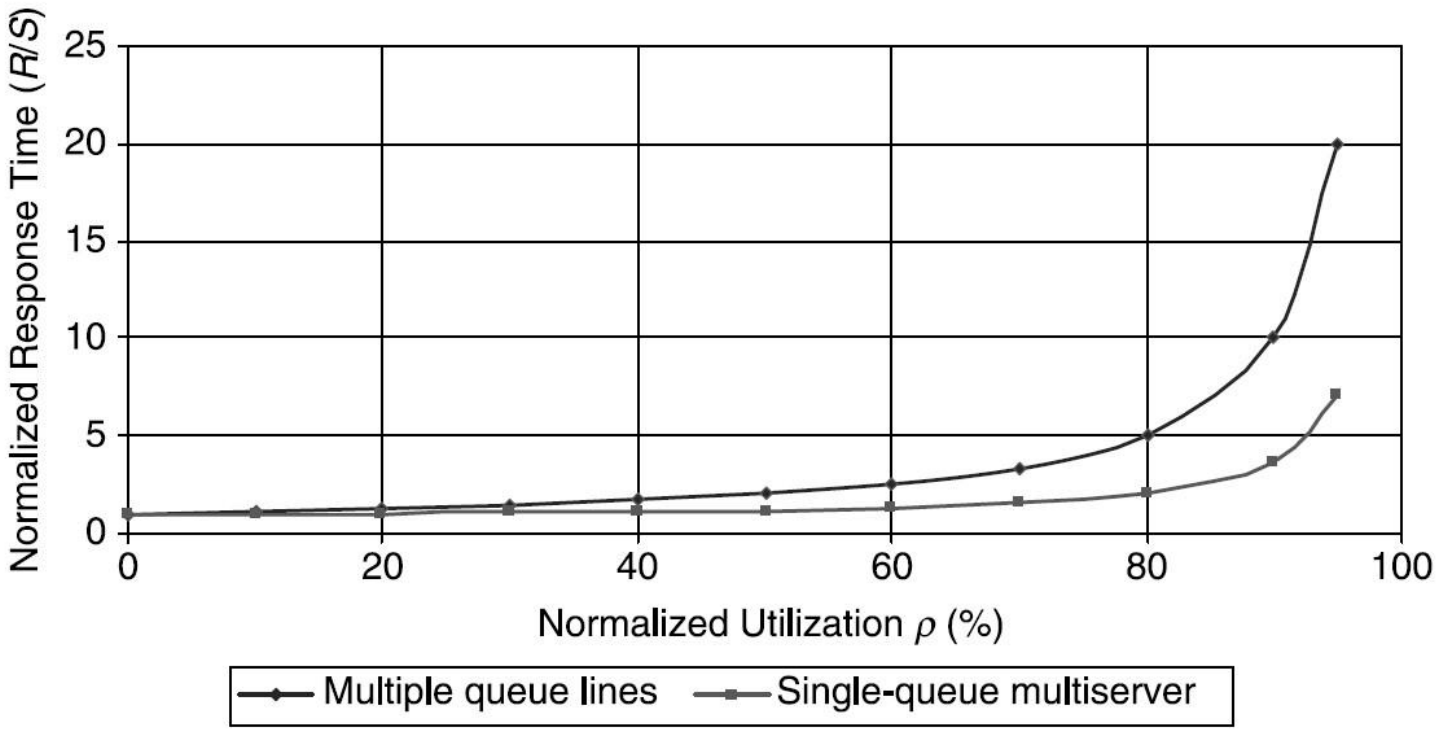


Figure 4.10 *Comparison of response time between multiple parallel queuing line scenario and single-queue multiserver scenario.*

M/M/m vs M/M/1

- M/M/m vs M/M/1
  - M/M/m：m个Servers，一个队列，平均处理速度 μ ，总速度 $m\mu$
  - M/M/1：1个Server，一个队列，平均处理速度 $m\mu$ ，总速度 $m\mu$

- M/M/m

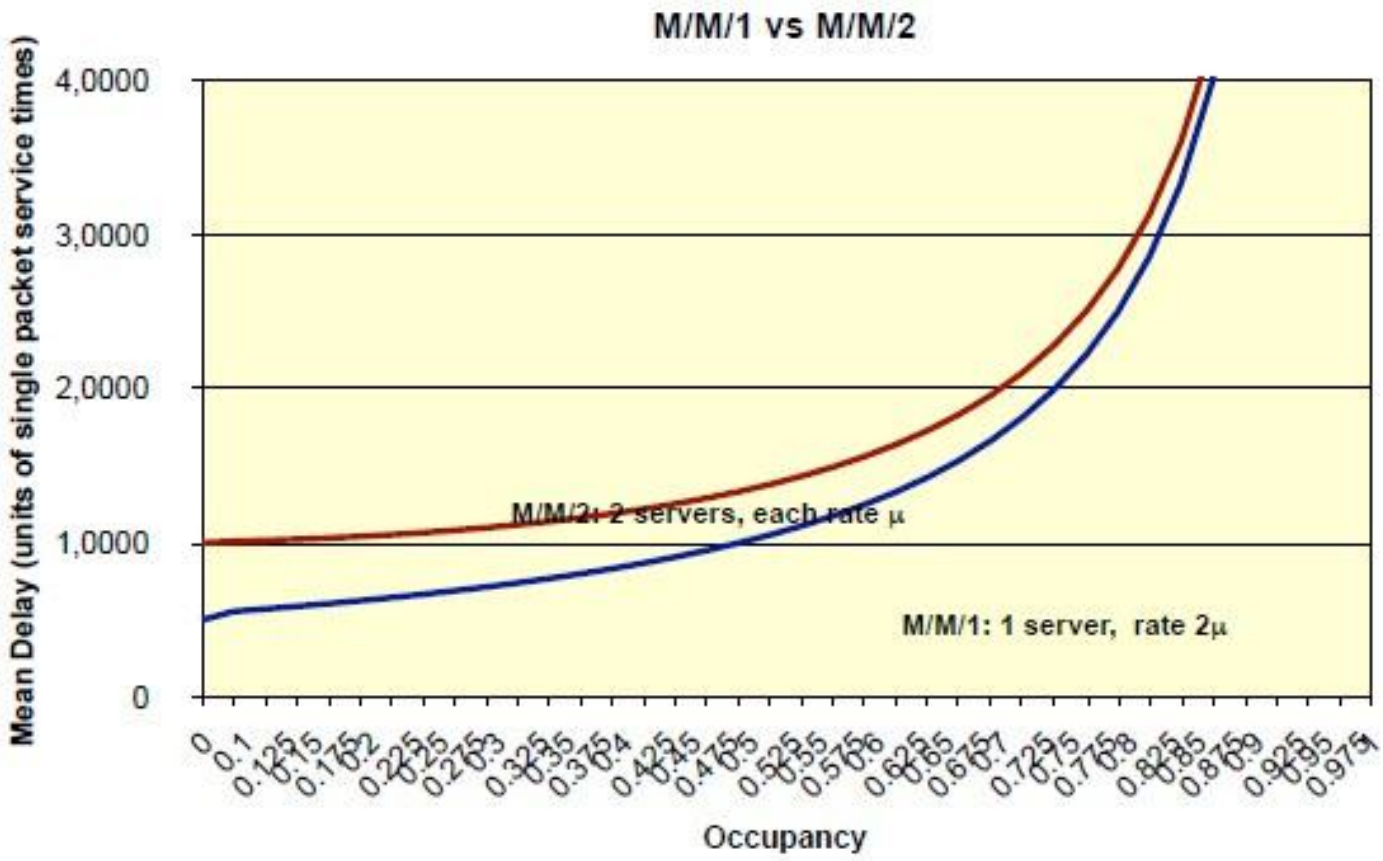
$$R \approx \frac{1}{\mu}\frac{1}{1-\rho^{m}}$$

- M/M/1

$$R = \frac{1}{m\mu}\frac{1}{1-\rho}$$

- 分析
  - 随着 ρ（$\rho = \frac{U}{m}$）的增加，M/M/m系统与M/M/1系统，哪个响应时间增加的更快？

M/M/m vs M/M/1


M/M/1 vs M/M/2
Mean Delay (units of single packet service times)
4,0000
3,0000
2,0000
1,0000
0
M/M/2: 2 servers, each rate μ
M/M/1: 1 server, rate 2μ
Occupancy

对比总结

- 结论
  - M/M/1 ($m\mu$) 优于M/M/m优于m M/M/1；
- 启发
  - 生活中
    - 一个熟练工的工资，要高于几个生手的工资和；
    - 银行排队，先领号，然后等待叫号 → M/M/m
  - 技术领域
    - 硬件(如CPU)的发展，先追求的是极致的性能，当单Server到达瓶颈之后，才考虑向多Servers发展；
    - 能将一个高性能Server资源使用到极致的系统，要优于堆积实例的系统；
      - Oracle vs MySQL；

M/M/m/k

- 略

- 此排队系统中，包含着Erlang-B Formula；
  - m个Servers，单一队列，队列长度为(k-m)时，请求的丢失率为多少？既为Erlang-B Formula；

大纲(一)

- 排队
  - 现实生活中
  - 计算机领域
- 排队论浅析
  - 排队论问题分析
  - Operational Law
    - Utilization Law
    - Little's Law
  - D. G. Kendall
  - Little's Law
  - Erlang's Formula

大纲(二)

- 排队论应用分析
  - 更好的理解系统监控
    - Linux I/O Stats
  - 更好的架构选型
  - 排队论与Performance
    - 什么是Performance？
    - 何谓平衡的系统？
    - 如何监控系统？
    - 如何进行高性能程序设计？
  - 容量规划（入门）

更好的理解系统监控

- 如何理解IOSTAT输出？（我已经告诉你了☺）

```
hzhedengcheng@app-66:~$ iostat -xm 60 10
Linux 3.2.0-3-amd64 (app-66.photo.163.org)      10/12/13        _x86_64_        (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          13.92    0.00    3.08    2.99    0.00   80.01

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.04   128.68    2.41    9.81     0.03     0.63   110.88     0.21   17.42    2.22   21.15   4.06   4.96
sdb               0.21   266.13  315.82  592.64     3.04    11.18    32.05     0.24    0.26    0.69    0.04   0.20  17.76

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           8.38    0.00   10.68    0.43    0.00   80.51

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     1.25    0.05    1.48     0.00     0.01    14.70     0.01    9.09    8.00    9.12   5.91   0.91
sdb               0.00     1.13  173.83   39.22     1.49     4.99    62.26     2.49   11.67    0.17   62.67   0.28   6.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          13.15    0.00   12.12    7.10    0.00   67.63

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     1.73    0.27    1.60     0.01     0.01    23.79     0.02   10.64    4.00   11.75   4.11   0.77
sdb               0.00    16.03 2555.38 1218.13    20.42    31.34    28.10    13.49    3.57    0.65    9.71   0.19  70.23

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          17.65    0.00   12.52   10.84    0.00   58.99

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00    19.75    0.00    1.72     0.00     0.08   100.04     0.02    9.51    0.00    9.51   4.66   0.80
sdb               0.00    20.73 4375.38 1500.22    34.18    26.19    21.04    11.27    1.92    0.64    5.66   0.15  89.77
```

更好的架构选型

- M/M/1 vs M/M/m vs m M/M/1

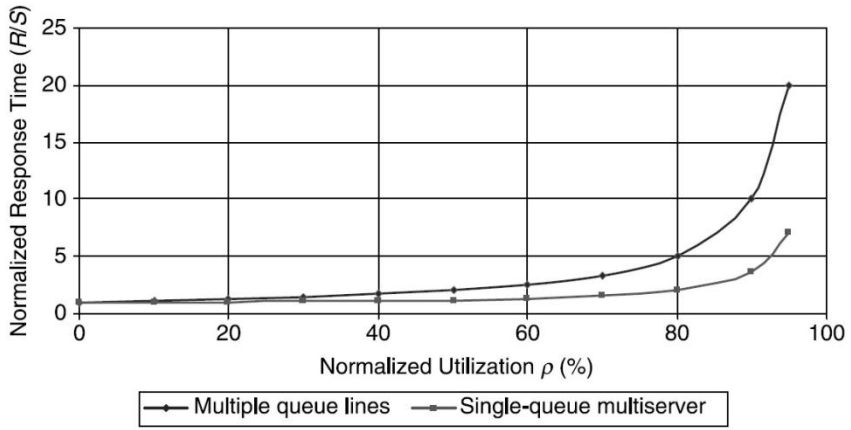
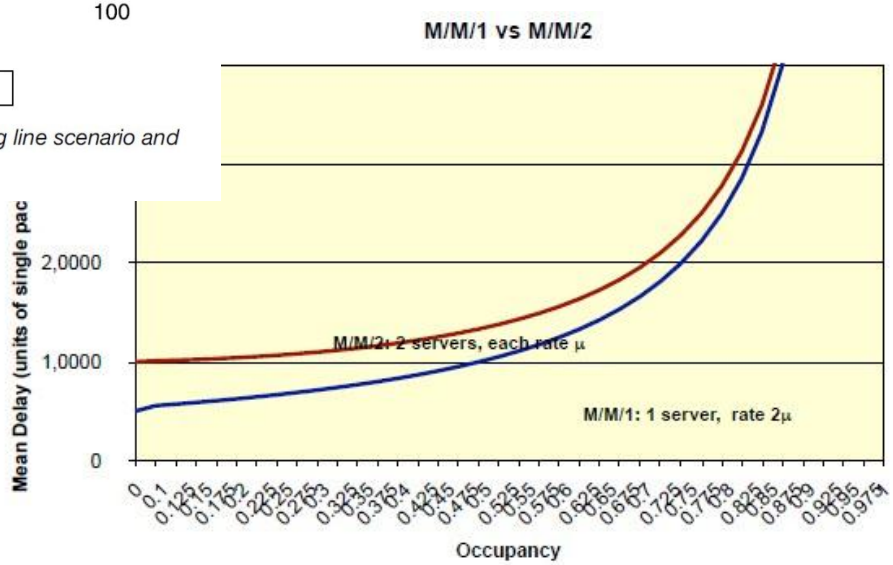


Figure 4.10 *Comparison of response time between multiple parallel queuing line scenario and single-queue multiserver scenario.*



什么是Performance？

- Performance：一体两面
  - **Response Time（用户层面）**
    - 关注一个操作需要多少时间——Response Time；
    - 响应时间越长——用户体验越差；

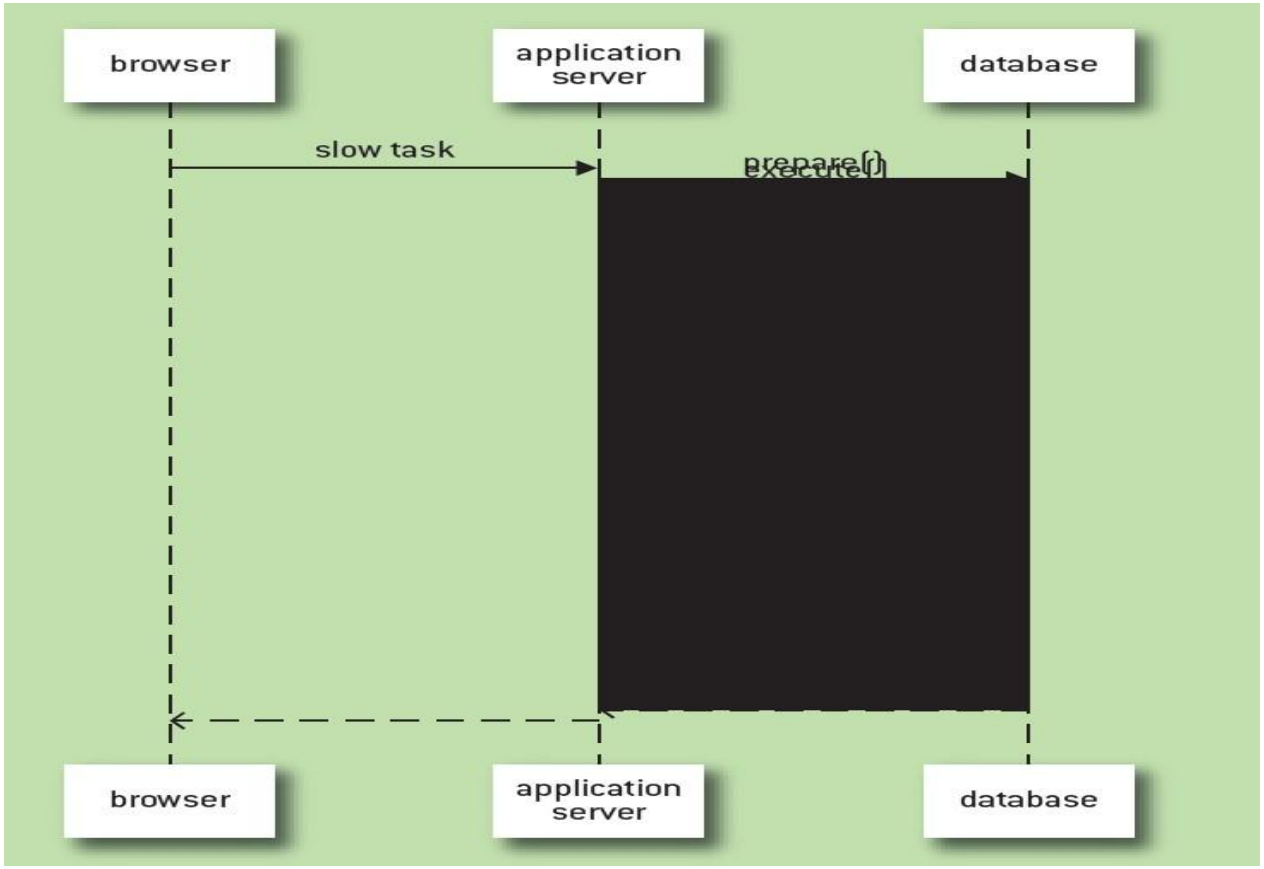
  - **Throughput（产品层面）**
    - 关注一批操作需要多少时间——Throughput
    - 相同配置下，单位时间完成的操作越多，资源利用率越高，投入的成本越少；

什么是Performance？

- 软件开发角度
  - 做高性能软件，在保证用户体验(Response Time)的同时，充分利用硬件资源(Utilization)，提供高吞吐率(Throughput)，减少资源投入；
  - 与排队论需要解决的问题，达到了高度统一；
- 排队论与Performance
  - Response Time
    - 响应时间 = 排队时间 + 服务时间 $R = W + S$
  - Throughput
    - 相同的硬件，系统提供高Throughput → 更高的资源利用率 U

理解Response Time

- 你点击一个页面按钮，响应时间过慢，该怎么办？



- 中间那坨黑乎乎的东东，究竟都做了些啥？

优化Response Time

- 优化Response Time的第一步 （方法论）
  - Profile
    - A *profile* is a tabular **decomposition of response time**, typically listed in descending order of component response time contribution.

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		819		53.81	
log file sync	11,585	139	12	9.15	Commit
db file sequential read	75,111	59	1	3.85	User I/O
latch free	670	10	15	0.64	Other
latch: cache buffers lru chain	421	7	17	0.46	Other

```
mysql> select * from TNT_NTSE_MUTEX_STATS order by waits desc limit 10;
+-------------------------------------------+------+-----------------------------+-----------+-------------+----------+---------+------------+
| FILE                                      | LINE | NAME                        | INSTANCES | LOCKS       | SPINS    | WAITS   | WAIT_TIME  |
+-------------------------------------------+------+-----------------------------+-----------+-------------+----------+---------+------------+
| src/ha_tnt.cpp                            | 1033 | TNT Prepare Commit Mutex    |         1 |     4316201 |  1872144 | 1786115 | 1469858980 |
| src/trx/TNTTransaction.cpp                |  749 | TNT Transaction System Mutex|         1 |   188574192 |  3894949 |  453084 |     529899 |
| src/misc/TxnLog.cpp                       |  279 | LogFlusher::lock            |         1 |    50635419 |  1221755 |  312733 |      39452 |
| ../storage/tnt/src/include/misc/LockManager.h |  78 | LockTableSlot::mutex      |    164864 | 13153953492 | 61429679 |  270998 |    1089828 |
| src/util/Thread.cpp                       |   70 | Thread::mutex               |        15 |    59646796 |   197300 |  160158 |       2792 |
| src/misc/Session.cpp                      |   37 | SessionManager::lock        |         1 |   124519383 |   964039 |   84739 |      72970 |
```

  - Profile的粒度越细，定位到的Performance瓶颈越准确；

优化Response Time

- 优化Response Time的第二步 （方法论）
  - Amdahl's Law
    - Performance improvement is **proportional** to how much a program uses the thing you improved.
    - Response Time能够改进的程度，跟你改进模块在Response Time中所占的权重正相关；
  - 优化模块的选择
    - 1. 根据Profile定位Response Time的分布；
    - 2. 根据Amdahl's Law，挑选潜在可优化的模块；
    - 3. 根据难易程度，选择投入产出比最大的模块进行优化；
    - 下图，该如何选择？

	Potential improvement % and cost of investment	R (sec)	R (%)
1	34.5% super expensive	1,748.229	70.8%
2	12.3% dirt cheap	338.470	13.7%
3	Impossible to improve	152.654	6.2%
4	4.0% dirt cheap	97.855	4.0%
5	0.1% super expensive	58.147	2.4%

优化Response Time

- 优化Response Time的第三步（实践）

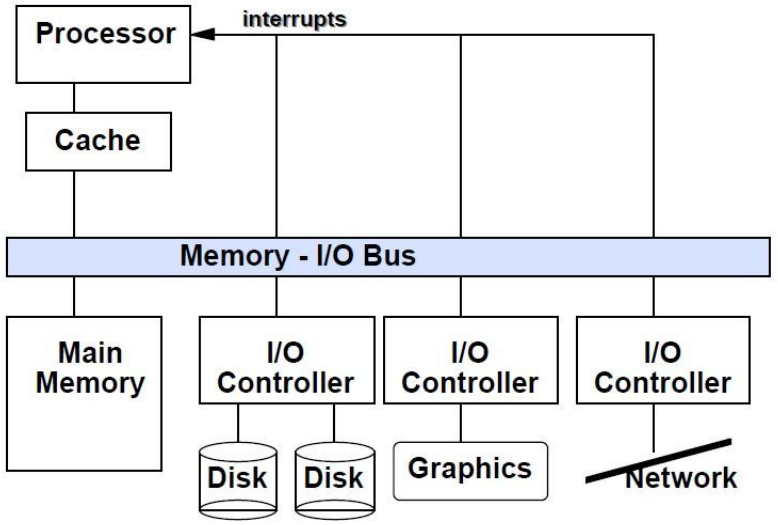
$$R = W + S$$

  - 优化服务时间 S
    - 硬件 （Hardware Delay）
      - 受硬件性能所限制，单次操作无法优化；
    - 软件临界区（也是一个排队系统：Coherence Delay）
      - 减少临界区长度；
      - 将耗时的操作提出临界区执行；
  - 优化等待时间 W
    - 可减少Server的访问次数；
      - Batch Process；
      - …

Throughput & Utilization

- Throughput & Utilization
  - Throughput
    - 系统能够提供高吞吐率；
  - Utilization
    - 高吞吐率的背后，是各种资源的合理利用；

- Balanced System
  - 一个系统会使用多种资源 →
  - 所谓Balanced System，就是所有资源都有相同的利用率；
  - 非Balanced System，必定有几种资源利用率较低 → 资源浪费



Utilization

- 问题
  - 是不是每个资源的利用率越高越好？没达到100%利用率就属于资源浪费？
- 解答
  - No，it depends.
- 区分应用类型
  - 根据前面提到的排队论，资源利用率与响应时间有关系。在固定响应时间下，不同的排队模型，能够达到的利用率是不同的。
  - 以数据库应用为例
    - OLAP系统
      - 对应于D/D/m模型，通过合理安排每个任务的调度，你可以获得极高的资源利用率；
    - OLTP系统
      - 对应于M/M/m模型，为了保证响应时间，必须将利用率限制在一定值之内；

Where is the Knee?

- M/M/m

$$R \approx \frac{1}{\mu} \frac{1}{1 - \rho^{m}}$$

- What is a Knee?
  - 最佳的Response Time；
  - 最佳的Throughput (Utilization)；
- Where is the Knee?
  - At some where in the curve；
- 结论
  - 无论理论上Knee是否存在，
    你都需要根据实际情况调整；

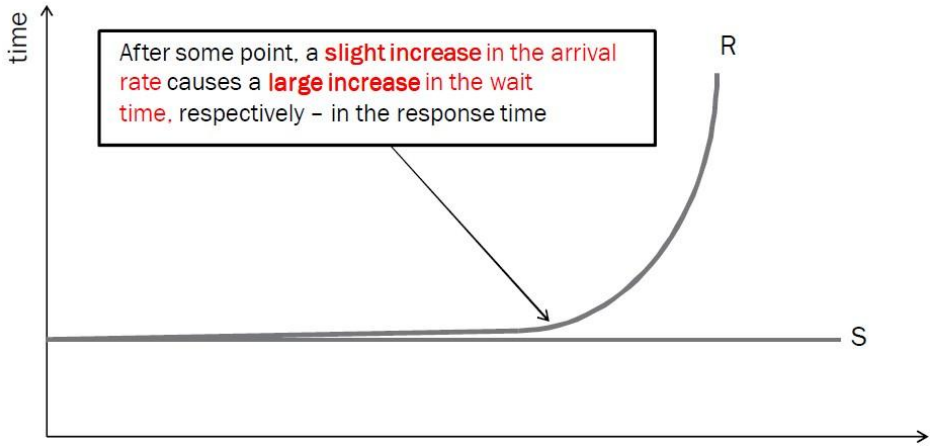


TABLE 5 **M/M/m Knee Values for Common Values of m**

Service channel count	Knee utilization
1	50%
2	57%
4	66%
8	74%
16	81%
32	86%
64	89%
128	92%

Performance： Optimization vs Tuning

- 性能优化的两条路
  - Performance Optimization
    - 从软件内部进行架构、代码级优化；
    - **硬件发展，推动软件进步**；
  - Performance Tuning
    - 在软件外部，根据软件暴露的参数，进行性能调优；
    - **给定软件，为软件选择合适的硬件**；
    - 以数据库为例：MySQL/Oracle能够发挥的硬件性能是不一样的，均有自己最合适的硬件环境；
  - Performance is a Feature
    - Performance is completely unknown until the production phase.
    - You need to write your application so that it’s easy to fix performance in production.

容量规划（入门）

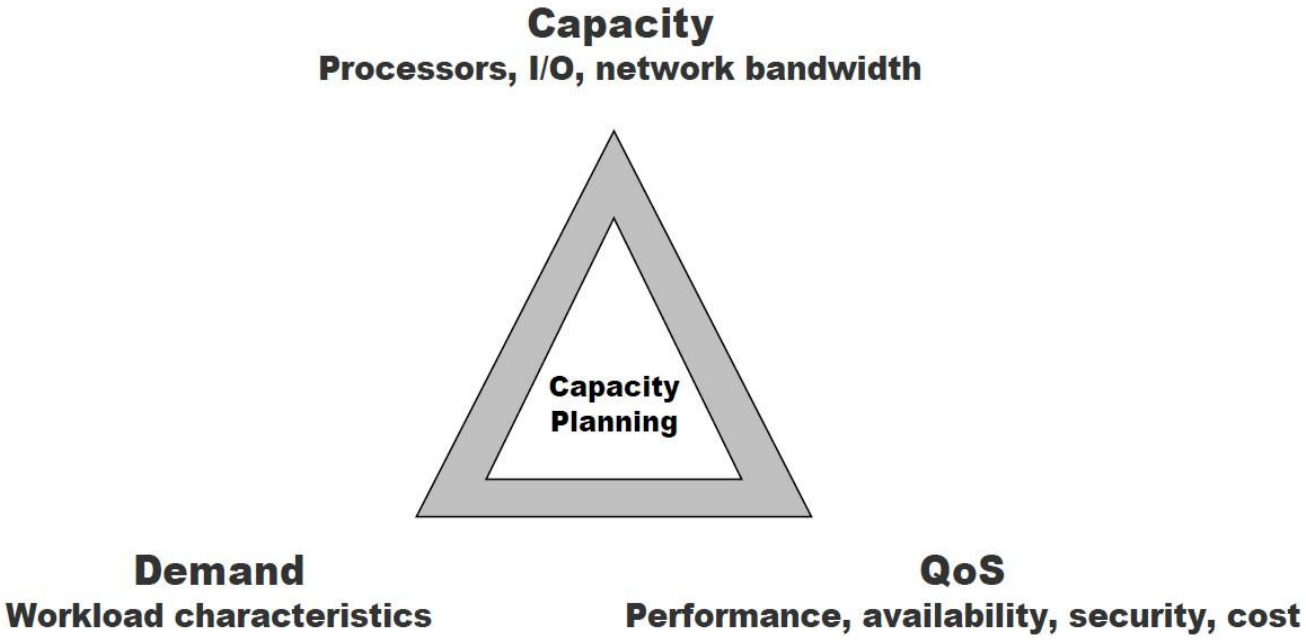
- 评估现有系统的容量是否合理？
  - 生活中
    - 一家KFC店，统计高峰期用户的平均客户数量，服务员的平均服务速度，就可以计算出每位顾客的平均等待时间，是否需要增加收银人员？
  - 技术领域
    - 监控线上的服务器是否存在利用率过低/过高的问题？
    - OLTP应用
      - M/M/m模型，在保证响应时间的前提下，观察利用率与拐点的关系(Knee)
      - 监控各种资源的利用率，是否为Balanced Systems，是否有资源会最早达到瓶颈(Knee)
    - OLAP应用
      - 可以使用较高的利用率，如果利用率不足，是否是任务调度不到位？

容量规划（入门）

- 预估未来系统的容量？
  - 生活中
    - 想在一个地方新开一家KFC，人流量是否够？需要多少收银人员？需要多少位置？才能满足用户的需求。
  - 技术领域
    - 预估系统Throughput；
    - 预估提供的Response Time；
    - 选择合适的软硬件组合；
    - 在系统压力极低的情况下，测试平均响应时间： $R_{\min}$
      - $R = W + S$
      - 系统压力极低 → $W \to 0$ $R_{\min} \approx S$
    - 根据以上得到的预估值，运用M/M/m模型，估算需要的Capacity；

容量规划

- To Be Continued ...



大纲（一）

- 排队
  - 现实生活中
  - 计算机领域
- 排队论浅析
  - 排队论问题分析
  - Operational Law
    - Utilization Law
    - Little’s Law
  - D. G. Kendall
  - Little’s Law
  - Erlang’s Formula

大纲（二）

- 排队论应用分析
  - 更好的理解系统监控
    - Linux I/O Stats
  - 更好的架构选型
  - 排队论与Performance
    - 什么是Performance？
    - 何谓平衡的系统？
    - 如何监控系统？
    - 如何进行高性能程序设计？
  - 容量规划（入门）

参考资料(排队论)

- *J.D.C Little.* A Proof for the Queuing Formula L = λW
- *Raj Jain.* operational law
- *Lund University.* Introduction to Queuing Systems
- *Moshe Zukerman*. Introduction to Queueing Theory and Stochastic Teletraffic Models
- *Daniel A. Menasce*. Performance Modeling – Part I Single Queues
- *Daniel A. Menasce.* Performance Modeling – Part II Queuing Networks
- *Daniel A. Menasce.* Performance of Multiprogrammed Operating Systems
- *Ivo Adan & Jacques Resing*. Queueing Theory
- *illinois.edu.* Introduction to Queueing Theory (Notation, Single Queues, Little's Result)
- *IAN ANGUS*. An Introduction to Erlang B and Erlang C
- *Randy H. Katz*. I/O—A Little Queuing Theory and I/O Interfaces
- *emory.edu.* Introduction to queueing theory
- *civil.iitb.* Queuing Analysis
- *GOLDENRATIΦ*. Comparison Between Single and Multiple Queues
- *Linux Journal*. Queuing in the Linux Network Stack.
- *J.D.C Little.* Littles Law as Viewed on Its 50th Anniversary

参考资料(Performance)

- *Cary Millsap*. Thinking Clearly about Performance
- *Brendan Gregg*. Thinking Methodically about Performance
- *Cary Millsap*. Performance Management: Myths & Facts
- *Neil J. Gunther*. Mind Your Knees and Queues
- *Michael Ley*. Does the Knee in a Queuing Curve Exists or is it just a Myth
- *Brendan Gregg*. Systems Performance: Enterprise and the Cloud
- *Brendan Gregg*. Open Source Systems Performance
- *Henry H. Liu*. Applying Queuing Theory to Optimizing the Performance of Enterprise Software Applications
- *Henry H. Liu*. Software Performance and Scalability: A Quantitative Approach
- *Anonymous*. Thinking of Performance (Forecasting Exercises)
- Virgilio A. F. Almeida. Capacity Planning: why, what and how.

Questions?

Thanks!

作者信息

- 何登成
  - 网易杭州研究院
- 新浪微博：何_登成
- 个人博客：深入MySQL内核
- 欢迎交流！