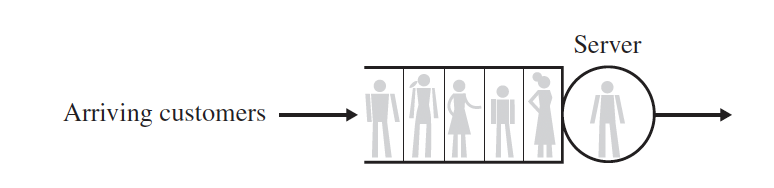
分析建模威力的激励性示例

第 1 章

1.1 什么是排队论？

排队论是研究当你有大量作业、资源稀缺、以及因此而产生的长队列和延迟时会发生什么的理论。它字面上的意思是“队列的理论”：是什么导致了队列的出现，以及如何让它们消失。

想象一下一个计算机系统，比如一个网页服务器，系统中只有一个作业。该作业到达，使用某些资源（如一些 CPU 和一些 I/O），然后离开。鉴于作业的资源需求，精确预测作业何时完成是非常简单的。由于没有队列，便不会有延迟。如果每个作业确实都能在自己的计算机上运行，就不需要排队论了。不幸的是，情况很少如此。

*图 1.1 排队示意图，其中顾客等待服务，而服务器提供服务。图中显示一名顾客正在服务器上接受服务，而其他五名顾客在队列中等待。*

排队论适用于任何出现队列的地方（见图 1.1）。我们都曾有过排队的经历，比如在银行里排队，心想为什么没有更多的柜员，或者在超市排队，心想为什么快速通道只限 8 件商品或更少，而不是 15 件或更少，或者是否最好设置两条快速通道，一条限 8 件商品或更少，另一条限 15 件商品或更少。队列也是任何计算机系统的核心。你的CPU使用时间共享调度器为等待CPU时间的作业队列提供服务。计算机磁盘为等待读写数据块的作业队列提供服务。网络中的路由器为等待路由的网络数据包队列提供服务。路由器队列是一个有限容量的队列，当需求超过缓冲空间时，数据包会被丢弃。内存条为请求内存块的线程队列提供服务。数据库有时会有锁队列，事务会等待获取记录的锁。服务器集群由多台服务器组成，每台服务器都有自己的作业队列。这样的例子不胜枚举。

排队论研究者的目标有两个。第一个目标是预测系统性能。通常，这意味着预测平均延迟、延迟的可变性，或者延迟超过某个服务等级协议（SLA）的概率。然而，这也可能意味着预测排队的作业数量或被使用的平均服务器数量（例如，总功耗需求），或者其他类似的指标。尽管预测很重要，更重要的目标是找到一种更优的系统设计来提高性能。通常这表现为容量规划，确定需要购买哪些额外资源以满足延迟目标（例如，是购买更快的磁盘还是更快的CPU，或是添加第二个较慢的磁盘）。然而，很多时候，即使不购买任何额外资源，仅通过部署更智能的调度策略或不同的路由策略来减少延迟，也可以提高性能。鉴于智能调度在计算机系统中的重要性，本书的第七部分全部用于理解调度策略。

排队论建立在一个更广泛的数学领域之上，称为随机建模和分析。随机建模将作业的服务需求和作业的到达时间间隔表示为随机变量。例如，UNIX 进程的 CPU 需求可以用帕累托分布（Pareto distribution）来建模，而繁忙的 Web 服务器上的作业到达过程可能可以用到达时间间隔呈指数分布的泊松过程（Poisson process）来建模。随机模型还可以用来建模作业之间的依赖关系，以及任何可以表示为随机变量的事物。

尽管通常可以提出一个随机模型来适当地表示系统中的作业或客户及其服务动态，但这些随机模型在求解性能问题时并不总是具有解析性(*译者关于解析性的解释: 在现实的计算机系统和网络建模中，有很多问题是非解析可解的，因为系统行为复杂，或者涉及大量变量的相互作用。然而，有些简化模型，比如假设服务时间是指数分布或到达过程是泊松过程，则可能变得解析可解。这就是为什么马尔可夫假设（如指数分布）经常被用来简化复杂问题，使其能够通过数学分析得到解的原因*)。正如我们在第四部分中讨论的，马尔可夫假设（如假设服务需求呈指数分布或作业到达过程为泊松过程）极大地简化了分析；因此，大多数现有的排队论文献都依赖于这些马尔可夫假设。在许多情况下，这些假设是合理的近似。例如，亚马逊上图书订单的到达过程可以合理地用泊松过程近似建模，因为有许多独立的用户，每个用户独立地以较低的频率提交请求（尽管这一假设在新《哈利·波特》书发布时会崩溃）。然而，在某些情况下，马尔可夫假设与现实差距很大；例如，当作业的服务需求高度可变或相关时。

*译者观点: 这就是我们为什么要做大量实验来建模, 获得性能数据的原因, 有这些数据做支撑, 模型就可以变得很简单*

虽然许多排队论书籍淡化了这些马尔可夫假设，但本书正好相反。我的很多研究都致力于展示工作负载假设对准确预测系统性能的影响。我发现，在许多情况下，简化工作负载假设会导致非常不准确的性能结果和糟糕的系统设计。因此，在我的研究中，我非常强调将测量的工作负载分布整合到分析中。与其试图掩盖所做的假设，本书将突出所有关于工作负载的假设。我们将具体讨论工作负载模型是否准确，以及我们的模型假设如何影响性能和设计，同时还会寻找更准确的工作负载模型。在我看来，计算机科学家迟迟不采用排队论的一个主要原因是标准的马尔可夫假设往往不适用。然而，通常有办法绕过这些假设，书中展示了许多方法，例如使用相位型分布和矩阵分析方法，这将在第21章介绍。

译者: 一些名词的解释[帕累托分布（Pareto Distribution)](https://zhuanlan.zhihu.com/p/67384223)和[泊松过程（Poisson Process)](https://zhuanlan.zhihu.com/p/53712363)都是在概率论和随机过程中的重要概念，但它们描述的现象和应用场景有所不同。以下是它们的主要区别：

1.2 排队论的威力示例

本章的其余部分将展示一些排队论实际应用的具体示例。请不要期望立即理解所有的示例，这些例子将在本书后面的章节中得到更详细的解释。像“泊松过程”这样的术语，如果你不熟悉，也会在本书中进一步解释。这些示例的目的是为了突出本书所涵盖的学习内容。

如前所述，排队论的一个用途是作为预测工具，可以用来预测给定系统的性能。例如，假设我们在分析一个具有一定带宽的网络，不同类别的数据包以不同的速率同时到达网络中的某些节点，并沿着不同的路径传输。排队论可以用来计算像是在某个路由器 (i) 等待的数据包的平均等待时间、路由器 (i) 队列积压的分布，或从路由器 (i) 到路由器 (j) 的平均总传输时间等数量。

现在我们将转向排队论作为设计工具的应用，用于选择最佳的系统设计，以最小化响应时间。以下示例展示了系统设计过程往往是反直觉的。

设计示例 1：到达率加倍

考虑一个系统，该系统由一个 CPU 组成，按照“先到先服务”（First-Come-First-Served, FCFS）顺序处理一个作业队列，如图 1.2 所示。作业按照某种随机过程以某个平均到达率到达，比如 ( \lambda = 3 ) 个作业每秒。每个作业有一些 CPU 服务需求，这些需求从某种作业服务需求的分布中独立抽取（在本例中我们可以假设任何分布）。假设平均服务率为 ( \mu = 5 ) 个作业每秒（即每个作业平均需要 1/5 秒的服务时间）。注意系统未处于过载状态（即 ( 3 < 5 )）。设 ( E[T] ) 表示系统的平均响应时间，其中响应时间是从作业到达时起直到完成服务的时间，也称为停留时间。

* ( \lambda = 3 )
* FCFS CPU
* ( \mu = 5 )
* 如果 ( \lambda ) 加倍为 ( 2\lambda )，那么 ( \mu ) 应该增加多少？

图 1.2 显示了一个 CPU 系统，按 FCFS 顺序处理作业。

*译者心声: 看到这里我就知道, 我在第一次看到问题时犯了什么错误. 作为性能工作者, 我习惯分析的是过载的, 不过载的时候, 前线的同事压根不会电话找我上场. 假设不同, 结论就不一样.*

**问题**：你的老板告诉你，从明天开始到达率将加倍。你被要求购买一个更快的 CPU，以确保作业体验到相同的平均响应时间 ( E[T] )。也就是说，客户不应注意到到达率增加的影响。那么你应该将 CPU 速度提高多少？(a) 将 CPU 速度加倍；(b) 将 CPU 速度提高到加倍以上；(c) 将 CPU 速度提高到小于加倍。

**答案**：( (c) ) 小于加倍。

这个示例展示了系统设计中的一个反直觉结果，虽然到达率加倍，但不需要将 CPU 速度加倍来保持相同的响应时间。这是排队论在系统设计中的一个典型应用。

**问题**：为什么答案不是 (a)？

**回答**：事实证明，将 CPU 速度加倍并同时将到达率加倍，通常会使平均响应时间减半！我们将在第13章中证明这一点。因此，CPU 速度不需要加倍。

*译者注:这个问题一开始就不是个多线程并行处理的问题, 而是排队问题. 这个区别非常大*

**问题**：你能立即给出一个不涉及任何排队论公式的粗略解释吗？如果我们将服务速率和到达率都加倍，会发生什么？

**回答**：想象有两种时间类型：联邦时间和克林贡时间。克林贡的秒比联邦的秒快。实际上，每一个克林贡秒相当于联邦时间中的半秒。现在，假设在联邦中，有一个 CPU 正在服务作业。作业以 ( \lambda ) 的速率（每秒作业数）到达，并以 ( \mu ) 的速率（每秒作业数）服务。克林贡人窃取了系统规格，并在克林贡世界中重新设计了相同的系统。在克林贡系统中，作业以 ( \lambda ) 个作业每克林贡秒的速率到达，并以 ( \mu ) 个作业每克林贡秒的速率服务。注意，两个系统的平均响应时间 ( E[T] ) 相同，区别在于克林贡系统的响应时间以克林贡秒计算，而联邦系统的响应时间以联邦秒计算。现在，设想 Kirk 船长同时观察联邦系统和克林贡重新设计的系统。从他的角度来看，克林贡系统的到达率和服务率都增加了一倍；然而，克林贡系统中的平均响应时间被减半了（因为克林贡秒在联邦时间里是半秒）。

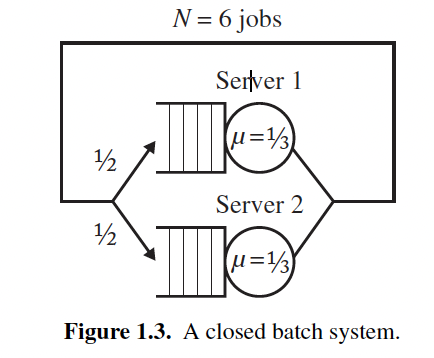
*译者注:恕我愚蠢, 我从数学上能搞明白为什么. 但是作者这个外星人比喻比数学更让我看不懂. 假设每隔1/3秒来一个任务, 我1/5秒就能处理掉, 剩下来2/15的时间CPU就在歇着, 平均响应时间为1/5秒. 现在给我一个1/10秒就能处理一个请求的处理器, 每隔1/6秒来一个请求, 我1/10秒就处理完了, 平均响应时间为1/10秒. 是不是我举的例子, 比较直观容易理解, 作者是大神, 思维太跳跃了*

**问题**：假设 CPU 使用时间共享服务顺序（简称为 PS，Processor-Sharing），而不是 FCFS。答案会改变吗？

**回答**：不会。相同的基本逻辑仍然适用。

设计示例 2 – 有时“改进”并没有作用

考虑图1.3中显示的批处理系统。系统中始终有 ( N = 6 ) 个作业（这称为多道程序级别）。一旦一个作业完成服务，新的作业就会开始（这称为“闭合”系统）。每个作业都必须经过“服务设施”。在服务设施中，作业以 ( 1/2 ) 的概率进入服务器1，以 ( 1/2 ) 的概率进入服务器2。服务器1的服务速率平均为每3秒1个作业。服务器2的服务速率同样为每3秒1个作业。作业的服务时间分布对于此问题无关紧要。响应时间的定义与平常一样，是指作业首次到达服务设施（在分叉点）到完成服务的时间。

*译者思考:N=6, 两个服务器, 意味着永远是饱和过载的*

**问题**：你将服务器1替换为一台速度加倍的服务器（新服务器的平均服务速率为每3秒处理2个作业）。这种“改进”会影响系统的平均响应时间吗？会影响吞吐量吗？（假设路由概率保持不变，仍为 (1/2) 和 (1/2)）

**回答**：几乎没有影响。平均响应时间和吞吐量几乎不受影响。这将在第7章中进行解释。 *译者思考:我并不理解为什么*

**问题**：假设系统的多道程序级别 (N) 较高，答案会改变吗？

**回答**：不会。随着 (N) 增加，响应时间和吞吐量的本来就很小的影响将趋于为零。

**问题**：假设系统的 (N) 值较低，答案会改变吗？

**回答**：会的。如果 (N) 足够低，那么这种“改进”确实有帮助。例如，考虑 (N=1) 的情况。

**问题**：假设系统变为开放系统，而不是闭合系统，如图1.4所示，其中到达时间与服务完成时间无关。现在这种“改进”是否会减少平均响应时间？

**回答**：当然！

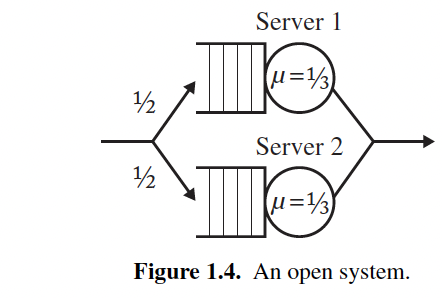


Fig. alt text

设计示例3 – 单台机器还是多台机器？

假设你有两个选择：一种是速度为 (s) 的快速CPU，另一种是 (n) 台速度为 (s/n) 的慢速CPU（见图1.5）。你的目标是最小化平均响应时间。首先，假设作业是不可抢占的（即每个作业必须运行至完成）。

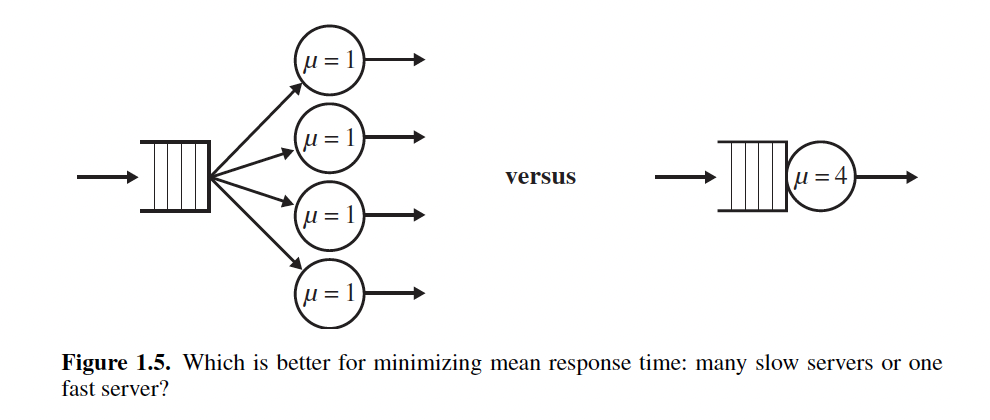


Fig. alt text

**问题**：哪个选择更好：一台快速机器还是多台慢速机器？

**提示**：假设我告诉你答案是：“这取决于工作负载。”你认为答案取决于工作负载的哪些方面？

**回答**：事实证明，答案取决于作业大小分布的可变性以及系统负载。

**问题**：当作业大小的可变性很高时，你更倾向于选择哪种系统？

**回答**：当作业大小的可变性很高时，我们更倾向于选择多台慢速服务器，因为我们不希望短作业被长作业阻塞。

**问题**：当负载较低时，你更倾向于选择哪种系统？

**回答**：当负载较低时，并不是所有服务器都会被充分利用，因此似乎选择一台快速服务器更好。

这些观察结果将在书中多次回顾。

**问题**：现在假设我们提出相同的问题，但作业是可抢占的，也就是说，作业可以被中断并在原地重新启动。在这种情况下，何时我们更倾向于选择多台慢速机器而不是一台快速机器？

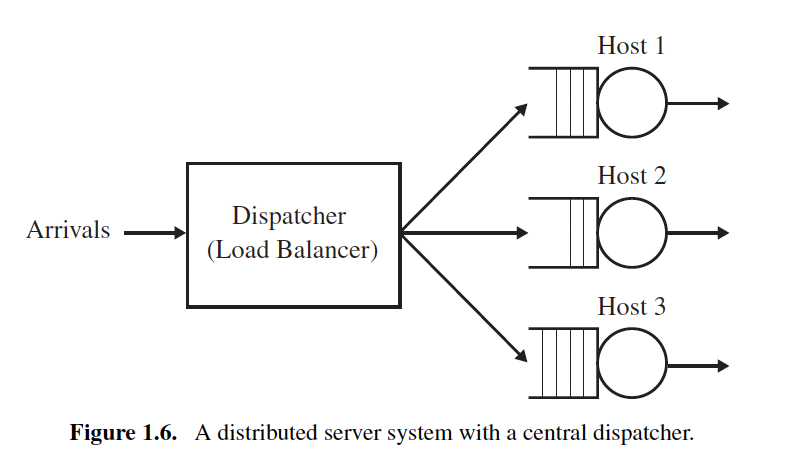
**回答**：如果作业是可抢占的，你可以使用一台快速机器模拟 (n) 台慢速机器的效果。因此，一台快速机器至少与多台慢速机器一样好。

多台慢速服务器与少数几台快速服务器的问题在广泛领域中有很大的应用潜力，因为任何资源（包括 CPU、功耗和带宽）都可以看作是计算资源。

以数据中心的功耗管理为例，考虑文献[69]中的问题：你有一个固定的功率预算 (P)，以及一个由 (n) 台服务器组成的服务器集群。你必须决定如何将功率分配给每台服务器，以最小化到达服务器集群的作业的整体平均响应时间。通常，分配更多功率给服务器会使其运行得更快（频率更高），但前提是受最大可能频率和开启服务器所需的最低功率限制。要回答如何分配功率的问题，你需要考虑是倾向于选择多台慢速服务器（每台服务器分配少量功率）还是少数几台快速服务器（将所有功率集中分配给少数服务器）。在文献[69]中，排队论被用来在多种参数设置下优化地回答这个问题。

另一个例子是，当带宽是资源时，我们可以问，什么时候将带宽划分为较小的部分是有利的，什么时候不应该划分。性能与价格结合时，问题变得更加有趣。例如，购买多台慢速服务器通常比购买少数几台快速服务器更便宜。然而，在某些情况下，多台慢速服务器的总功耗可能比几台快速服务器更高。所有这些因素都会进一步影响架构选择。

设计示例 4 —— 服务器集群中的任务分配

考虑一个带有中央调度器和多个主机的服务器集群。每个到达的作业会立即被分配到其中一台主机进行处理。图 1.6 展示了这样的系统。

这种服务器集群无处不在。Web 服务器集群通常部署前端调度器，例如 Cisco 的 Local Director 或 IBM 的 Network Dispatcher。超级计算站点可能使用 LoadLeveler 或其他调度器来平衡负载并将作业分配给主机。

暂时假设所有主机都是相同的（同质的），所有作业只使用单一资源。还假设作业一旦分配到主机后，将按“先到先服务”（FCFS）顺序处理，并且作业不可抢占。

有多种可能的任务分配策略可以用于将作业分配到主机。以下是一些示例：

* **Random**（随机）：每个作业通过抛硬币来决定它被路由到哪里。
* **Round-Robin**（轮询）：第 (i) 个作业分配到主机 (i \mod n)，其中 (n) 是主机数量，主机编号为 0, 1, ..., (n-1)。
* **Shortest-Queue**（最短队列）：每个作业分配到作业最少的主机。
* \*\*Size-Interval-Task-Assignment (SITA)\*\*（按区间任务分配）：根据作业的长短进行分配。“短”作业分配到第一个主机，“中等”作业分配到第二个主机，“长”作业分配到第三个主机，以此类推，具体定义由“短”“中”等区间决定。
* \*\*Least-Work-Left (LWL)\*\*（最少剩余工作量）：每个作业分配到剩余工作量最少的主机，主机的“工作量”是主机上所有作业大小的总和。
* **Central-Queue**（中央队列）：作业不再在每个主机上排队，而是在中央队列中集中排队。当主机完成一个作业时，它从中央队列中抓取下一个作业进行处理。

**问题**：上述哪种任务分配策略能带来最短的平均响应时间？

**回答**：鉴于服务器集群的普遍存在，令人惊讶的是我们对这个问题的了解还很有限。如果作业大小的可变性较低，那么**LWL**（Least-Work-Left）策略是最好的。如果作业大小的可变性较高，那么避免短作业被长作业阻塞就很重要，因此类似于**SITA**（Size-Interval-Task-Assignment）的策略，可以有效隔离短作业和长作业，表现会更好。事实上，长时间以来，人们认为在作业大小的可变性较高时，**SITA** 总是优于 **LWL**。然而，最近发现（参见文献 [90]），即使在作业大小可变性趋向无穷大时，**SITA** 可能比 **LWL** 差得多。事实证明，工作负载的其他属性，包括负载和作业大小分布的分数矩等，也同样重要。

**问题**：对于上一个问题，知道作业大小有多重要？例如，**LWL**（需要知道作业大小）与**Central-Queue**（不需要知道作业大小）相比如何？

**回答**：实际上，大多数任务分配策略并不需要知道作业的大小。例如，可以通过归纳法证明**LWL**与**Central-Queue**是等效的。即使是像**SITA**这样依赖于作业大小的策略，也可以通过不需要知道作业大小的其他策略很好地近似（参见文献 [82]）。

*译者观点: 对于磁盘, 网络, AI计算, 压缩, 这些其实都可以知道作业大小, 比如根据包和块的大小, 比如根据矩阵大小*

**问题**：现在考虑一个不同的模型，其中作业是可抢占的。具体来说，假设服务器是**Processor-Sharing (PS)服务器，作业在服务器上共享处理时间，而不是按“先到先服务”（**FCFS）顺序处理。现在哪个任务分配策略更好？答案是否与**FCFS**服务器相同？

**回答**：对于**FCFS**服务器来说最好的任务分配策略，在**PS**服务器下往往是灾难性的。对于**PS**服务器，**Shortest-Queue**（最短队列）策略接近最优（参见文献 [79]），而在作业大小的可变性较高时，这一策略在**FCFS**服务器下表现非常糟糕。

在任务分配策略方面仍然有许多未解的问题。例如，带有**PS**服务器的服务器集群几乎没有受到关注，甚至**FCFS**服务器的情况也仅部分得到了理解。还有许多其他任务分配策略没有被提及。例如，**cycle stealing**（利用空闲主机处理其他队列中的作业）可以与现有的任务分配策略结合，创造出改进的策略。还有其他指标需要考虑，如最小化响应时间的方差，而不是平均响应时间，或者最大化公平性。最后，当工作负载随着时间变化时，任务分配可能变得更加复杂和重要。

任务分配在第24章中有详细的分析，我们会在有机会研究实际工作负载之后讨论。

设计示例 5 —— 调度策略

假设你有一台服务器。作业按照泊松过程到达。你可以对作业大小的分布做任何假设。以下是几种可能的服务顺序（调度顺序）：

* \*\*First-Come-First-Served (FCFS)\*\*：当服务器完成一个作业时，它开始处理最早到达的作业。
* **非抢占式最后到达先服务（LCFS）**：当服务器完成一个作业时，它开始处理最后到达的作业。
* **随机调度**：当服务器完成一个作业时，它随机选择一个作业开始处理。

**问题**：哪种非抢占式服务顺序会导致最低的平均响应时间？

**回答**：信不信由你，它们的平均响应时间都是相同的。

**问题**：假设我们将非抢占式 LCFS 政策改为**抢占式 LCFS（PLCFS）**，其工作方式如下：每当有新作业到达时，它立即抢占当前正在服务的作业。这个政策的平均响应时间与其他政策相比如何？

**回答**：这取决于作业大小分布的可变性。如果作业大小分布具有中等及以上的可变性，那么 PLCFS 将带来巨大的改进。如果作业大小分布几乎没有可变性（基本恒定），那么 PLCFS 政策的表现可能差两倍。

我们将在本书的第 28 至 33 章中研究许多调度理论中的反直觉结果。

更多设计示例

在计算机系统设计中还有许多问题适合通过排队论解决。

一个例子是**设置成本**的概念。事实证明，启动一台关闭的服务器可能需要大量的时间和功率。在设计高效的电源管理策略时，我们通常希望将服务器关闭（以节省功耗），但当作业到达时，我们不得不支付设置成本来重新启动它们。鉴于响应时间和功耗方面的性能目标，一个重要的问题是是否值得关闭服务器。如果值得，那么接下来可以问，究竟应该留下多少台服务器继续运行。我们将在第 15 和第 27 章中讨论这些问题。

*译者注: CPU Parking策略*

还有一些问题涉及当作业具有优先级时的最优调度（例如，某些用户为他们的作业支付了更多费用以获得比其他用户作业更高的优先级，或者某些作业本质上比其他作业更重要）。排队论在设计正确的优先级方案以最大化完成工作的价值方面非常有用。

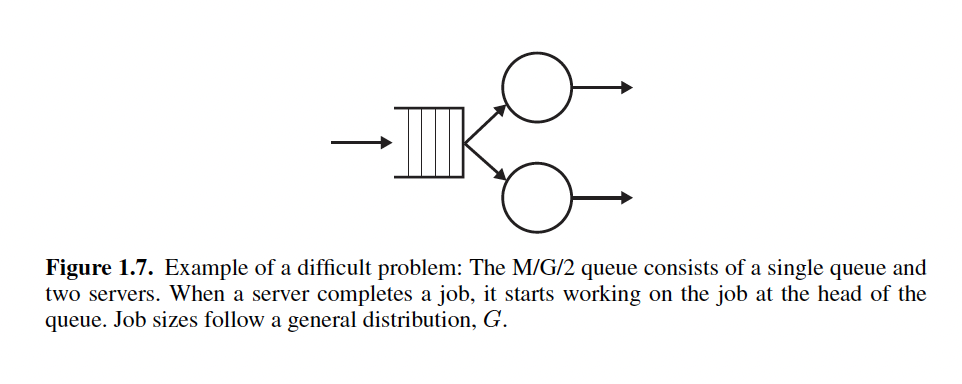


Fig. alt text

然而，排队论（更广泛地说，分析建模）目前并不是万能的！有许多非常简单的问题，我们最多只能进行近似分析。比如，考虑图1.7中显示的简单两服务器网络，其中作业大小来自于一般分布。没有人知道如何为这个网络推导出平均响应时间。虽然存在一些近似解，但它们相当差，尤其是在作业大小的可变性较高时（参见[76]）。本书中提到了许多这样的开放性问题，我们鼓励读者尝试解决这些问题！

[上一章](README.md)[下一章](02Terminology.md)