学号 221307020006

年级 2022 级

河海大学

《通信网理论》阅读报告

专　　业: 信号与信息处理

姓　　名: 吉普

联系电话: 19850052801

指导教师: 李旭杰

2023 年 01 月

使用 M/M/m 队列系统的云计算动态性能优化

摘　　要

云计算的成功引来了越来越多人和企业的使用。一方面，使用云计算降低了成本；另一方面，使用云计算提高了效率。由于用户非常关注服务质量 (QoS)，云计算的性能优化成为其成功应用的关键。为了优化云计算中多个请求和服务的性能，我们使用排队论对数据中心服务的各个参数进行了分析和推导。然后，通过分析排队系统的性能参数，提出了综合的优化模式、功能和策略。最后，我们建立了基于综合优化模式的仿真；我们还将仿真结果与经典优化方法 (短时优先和先进先出) 进行了比较和分析，表明所提出的模型可以优化平均等待时间、平均队列长度和服务客户数量。

一、引言

云计算是一种提供计算基础设施的新模式，旨在将计算基础设施的位置转移到网络上，以降低硬件和软件资源的管理和维护成本 [1]。这种云概念强调将管理、维护和投资从客户转移到提供商。云计算是一种模型，用于实现无处不在的、方便的、按需的网络访问可配置计算资源共享池 (例如网络、服务器、存储、应用程序和服务)，这使得资源可以通过最小的管理成本或与服务提供商的交互来快速提供和释放 [2]。一般来说，云计算提供了渠道让用户通过网络访问实时可扩展的资源，如文件、程序、数据、硬件、计算和第三方服务。这些用户通过定制的服务水平协议 (SLA) 获得计算资源和服务；他们只根据使用时间、使用方式或数据传输量支付费用。任何 SLA 管理策略都包含两个完全不同的阶段：合同的协商和履行情况的实时监控。因此，SLA 管理包含 SLA 合约定义：具有 QoS 参数的基本模式。在各个方面，QoS 是云计算向用户提供服务的基础。QoS 包括可用性、吞吐量、可靠性和安全性，以及许多其他参数，但性能指标如响应时间、任务阻塞概率、即时服务概率，以及系统中任务的平均数量 [3]，所有这些都可以通过使用排队论这一工具 [4] 来确定。因此，为了满足客户的 QoS 要求，优化 QoS 是很重要的。由于云计算动态地提供计算资源以满足不同客户对 QoS 的需求，优化资源利用率将是一项艰巨的任务。另一方面，一个数据中心拥有大量的物理计算节点 [5]；传统的排队分析很少涉及这种规模的系统。尽管前人的研究已经针对云计算中的关键研究问题提出了几种方法，包括云安全 [6-8]、隐私 [9-10]、能源效率 [11] 和资源管理 [12-14]，性能优化方面的研究仍然很少。

在本文中，数据中心被塑造成一个服务中心，可以被当成一个具有多任务到达和无限容量的任务请求缓冲区的 M/M/m 队列系统。通过 M/M/m 排队论，我们推导出各个参数的方程；然后，我们设计了一个优化函数和一个综合优化方法。仿真结果表明，与短时优先和先进先出的经典方法相比，所提出的优化方法提高了数据中心的性能。

本文的其余部分安排如下。第 2 节讨论了性能优化和分析的相关工作。第 3 节给出排队模型和优化策略。我们介绍了模拟设置和模拟结果，然后在第 4 节中分析和比较了其他经典方法的结果，例如短时优先和先进先出。在第 5 节中对我们的工作进行了总结，还概述了未来工作的方向。

二、相关工作

尽管云计算引起了研究的关注，但迄今为止只有一小部分工作解决了性能优化问题。在 [15] 中，Li 提出了一种面向云计算的差异化服务作业调度系统；然后，通过分析用户作业的差异化 QoS 需求，为本系统建立了相应的非抢占式优先级 M/G/1 排队模型。他们提供了相应的算法来获取具有不同优先级的每个工作的服务的近似优化值。在 [16] 中，使用云中心模型作为经典开放网络，假设到达时间和服务时间都是指数分布，获得了响应时间的分布。通过响应时间的分布，发现了任务最大数量、服务资源最小值和服务水平最高值之间的关系。在 [17] 中，他们使用线性预测方法和 FPR-R 方法从资源利用日志中获取有用信息，使 M/M/1 排队理论预测方法具有更好的响应时间和更低的能耗。在 [18] 中，他们使用排队模型研究单类服务和多类服务情况下的资源分配问题。此外，在每种情况下，他们优化资源分配以最小化平均响应时间或最小化资源成本。

此外，一些研究者还进行了性能分析的研究。在 [19] 中，作者提出了一个 M/G/m 排队系统，表明请求之间的间隔时间是指数分布的；服务时间是一般分布的，设施节点数量为 m。在另一篇论文 [20] 中，作者将云中心建模为一个有单个任务到达和有限容量的任务请求缓冲区的 M/G/m/m+r 排队系统。为了评估性能，他们使用了基于变换的分析模型和嵌入式马尔可夫链模型的组合，获得了系统中任务数量和响应时间的完整概率分布。模拟结果表明，该模型和方法能够准确地预测系统中任务数量的平均值、阻塞概率、及时服务的概率以及响应时间分布的特征，如平均值和标准差、偏度 (skewness) 和峰度 (kurtosis)。

在 [21] 中，作者针对云性能管理提出了一种基于排队的分析模型。在他们的研究中，Web 应用程序被建模为队列，虚拟机被建模为服务中心。他们应用排队理论模型动

态地创建和删除虚拟机，以实现扩展和缩减。

在 [22] 中，作者分析了云计算中资源配置的一般问题。为了在不同客户签订不同服务级别协议的情况下支持云资源提供商的资源分配决策，他们使用了带不同优先级类的 M/M/C/C 排队系统建模了云中心。他们分析的主要性能标准是可以被解析确定的不同客户类别的拒绝概率。

根据以上分析，我们知道在云计算中已经研究了使用排队理论进行性能评估和分析，但关于性能优化研究很少。此外，现有研究中已经分别研究了 QoS 的各个参数，但还没有一项工作同时处理过所有的参数。在本文中，我们使用具有多个任务到达和无限容量任务请求缓冲的 M/M/m 排队系统来优化性能。

三、云计算的排队模型和优化策略

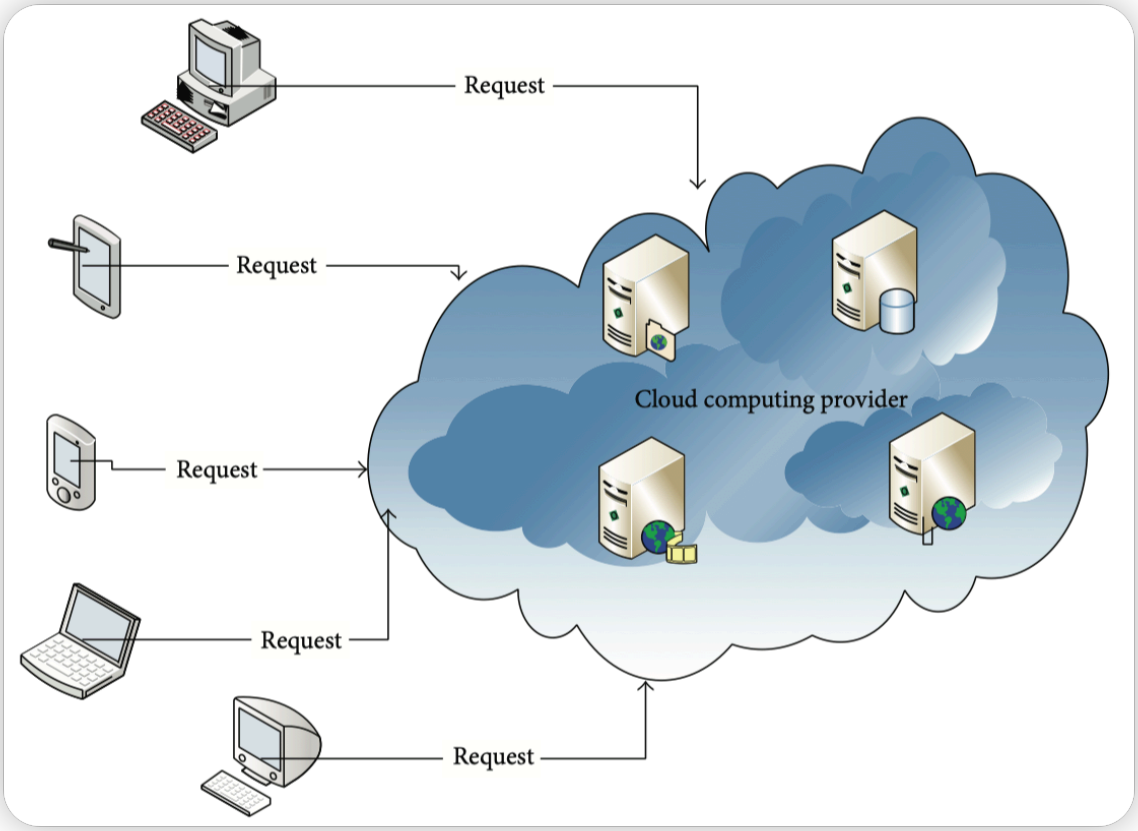

图 3.1 云计算服务请求模型图示

在云计算中，有很多用户访问服务。我们将云计算建模如图3.1所示。该模型由可以是服务中心的云体系结构组成。服务中心是全球各地客户的单一访问点。服务中心是

提供商提供的服务资源的集合，用于托管所有用户的应用程序。每个用户可以根据不同类型的请求申请使用服务并向服务提供商支付一些费用。

云计算提供商建立服务中心供客户使用，如提供多种方式供用户使用的亚马逊。在本文中，我们使用按需实例。按需实例允许您按小时计算计算能力的费用，无长期承诺。这使您免除了计划、购买和维护硬件的成本和复杂性，并将通常的大固定成本转化为小可变成本 [23]。

图3.1显示的云计算服务模型可以在图3.2中映射为排队模型。假设有 n 个请求和 m 个服务，且它们都是独立的。由于连续到达的请求可能来自两个不同的用户，因此到达间隔是一个随机变量，可以在云计算中用指数随机变量来建模。因此，请求的到达遵循到达率为 λ_i 的泊松过程。调度器队列中的请求分配到不同的计算服务器，调度速率取决于调度器。假设有 m 个计算服务器，标记为 Service_1，Service_2，Service_i 以及 Service_m；服务速率为 μ_i。所以，总的到达率为 $\lambda = \sum_{i=1}^{n} \lambda_i$，总的服务率为 $\mu = \sum_{j=1}^{m} \mu_i$。排队论已经证明了在 $\lambda/\mu < 1$ 的情况下，系统是稳定的。服务率跟到达率一样，遵循泊松过程。因此，M/M/m 排队模型适用于云计算模型。

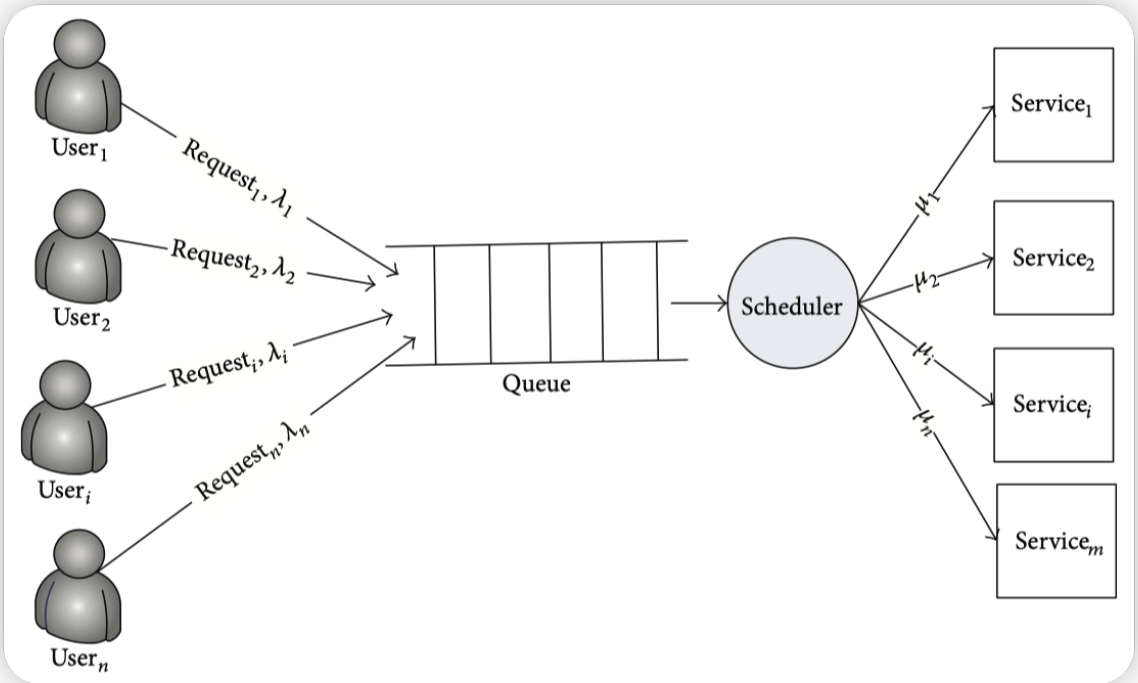

图 3.2 云计算中计算机服务的队列性能模式

3.1 状态平衡方程

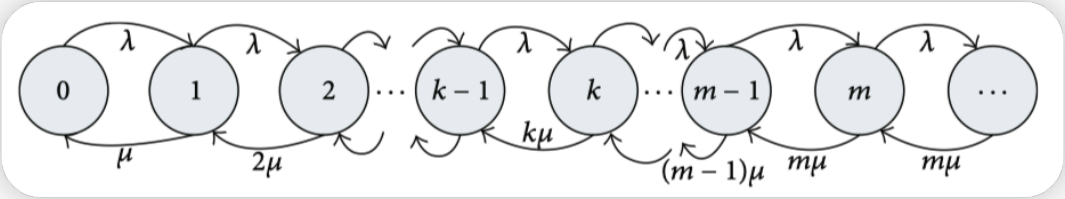

图 3.3 状态转移概率图

由于客户的请求来自世界各地，云计算可以提供无限的服务，因此客户的来源和排队模型的数量是不受限制的。系统的状态集合为 $E = \{0, 1, 2, ...\}$；因此这些平衡方程也可以由图3.3中的 M/M/m 状态转移图来描述。

当状态为 $k(0 \leq n \leq m)$，有 n 个服务在忙，其余的 $m - n$ 个服务空闲；当状态为 $k(m \leq n \leq \infty)$，有 m 个服务在忙，其余的 $n - m$ 个客户在等待。设 $\rho = \lambda/(m\mu)$，假定满足稳定条件 $\rho < 1$，则根据 M/M/m 队列有以下状态平衡方程：

$$
\begin{aligned}
p_1 &= m\rho p_0, \\
p_2 &= \frac{m^2}{2!}\rho^2 p_0, \\
p_3 &= \frac{m^3}{3!}\rho^3 p_0, \\
p_m &= \frac{m^m}{m!}\rho^m p_0, \\
&\cdots \\
p_{m+r} &= \frac{m^m}{m!}\rho^{m+r} p_0.
\end{aligned}
\tag{1}
$$

因此，可得通式：

$$
\begin{aligned}
p_n &= \frac{m^n}{n!}\rho^n p_0, \quad 0 \leq n < m, \\
p_n &= \frac{m^m}{m!}\rho^n p_0, \quad n \geq m.
\end{aligned}
\tag{2}
$$

为了获得 p_0，我们累加公式2的两边，因为 $\sum_{n=0}^{\infty} p_n = 1$，我们可以获得关于 p_0 的等式，可以得到 p_0 的解为：

$$
p_0 = \left(\sum_{n=0}^{m-1} \frac{\rho^n}{n!} + \frac{\rho^k}{k!} \times \frac{k}{k - \rho} \right)^{-1}. \tag{3}
$$

3.2 平均队列长度，延迟和等待时间

为了评估和优化性能，我们应该推导参数的方程。首先，我们定义以下符号：

- L_s 是一个随机变量，代表系统中客户的总数 (包含在队列中等待的和正在被服务的)；
- L_q 是一个随机变量，代表在队列中的客户数 (不包含正在被服务的)；
- N_s 是一个随机变量，代表正在被服务的客户数；
- W_s 是一个随机变量，代表系统中的延迟 (这包含用户排队和被服务的时间)；
- W_q 是一个随机变量，代表用户在队列中的等待时间 (不包含用户被服务的时间)；
- τ 是一个随机变量，代表服务时间。

根据上面的符号定义，我们可以得到：

$$\begin{aligned} E\left[L_s\right] &= E\left[L_q\right] + E\left[N_s\right], \\ E\left[W_s\right] &= E\left[W_q\right] + E[\tau]. \end{aligned} \tag{4}$$

很显然，

$$E[\tau] = \frac{1}{\mu}. \tag{5}$$

为了获得 M/M/m 队列的 $E[N_s]$，我们对系统使用 Little 公式。如果我们只考虑系统中的服务器而不考虑服务器外的等待室，可以发现，由于没有阻塞带来的损失，系统的到达率就是 λ，而平均服务时间就是 $E[\tau] = 1/\mu$。因此，根据 Little 公式，处于忙绿的服务器数可以表示为：

$$E\left[N_s\right] = \frac{\lambda}{\mu} = \rho \tag{6}$$

为了获得 $E[L_q]$，我们假设两个相互排斥和详尽的事件 $\{\, q \geq m \,\}$, and $\{q < m\}$，我们可以得到：

$$\begin{aligned} E\left[L_q\right] =& E\left[L_q \mid q \geq m\right] P(q \geq m) \\ &+ E\left[L_q \mid q < m\right] P(q < m). \end{aligned} \tag{7}$$

为了获得 $E[L_q|q \geq m]$，我们可以发现当 $q \geq m$ 时，M/M/m 队列就可以等价于一个服务速率 $m\mu$ 的 M/M/1 队列；所以，这个 M/M/1 队列的平均队列长度等于 $\rho/(1-\rho)$，

其中 $\rho = \lambda/(m\mu)$，因此：

$$E\left[L_q \mid q \geq m\right] = \frac{\rho/m}{1-\rho/m} = \frac{\rho}{m-\rho}. \tag{8}$$

由于 $E[L_q|q < m] = 0$ 并且 $P(q \geq m) = C_m(\rho)$，我们可以得到：

$$E\left[L_q\right] = C_m(\rho)\frac{\rho}{k-\rho}. \tag{9}$$

3.3 优化策略和优化算法

在云计算中，数据中心有很多服务。在3.2中，我们已经讨论了 M/M/m 队列系统的很多参数。队列中用户数量的参数为 L_s，L_q 和 N_s；在这三个参数中，L_q 起主导作用，因为正在被服务的用户数量由系统中服务器的数量决定，而 L_q 取决于服务器的性能。我们选用利用率，平均时间和等待被服务的用户数量作为优化参数。基于排队论系统的任务调度和性能优化模型如图3.4所示。

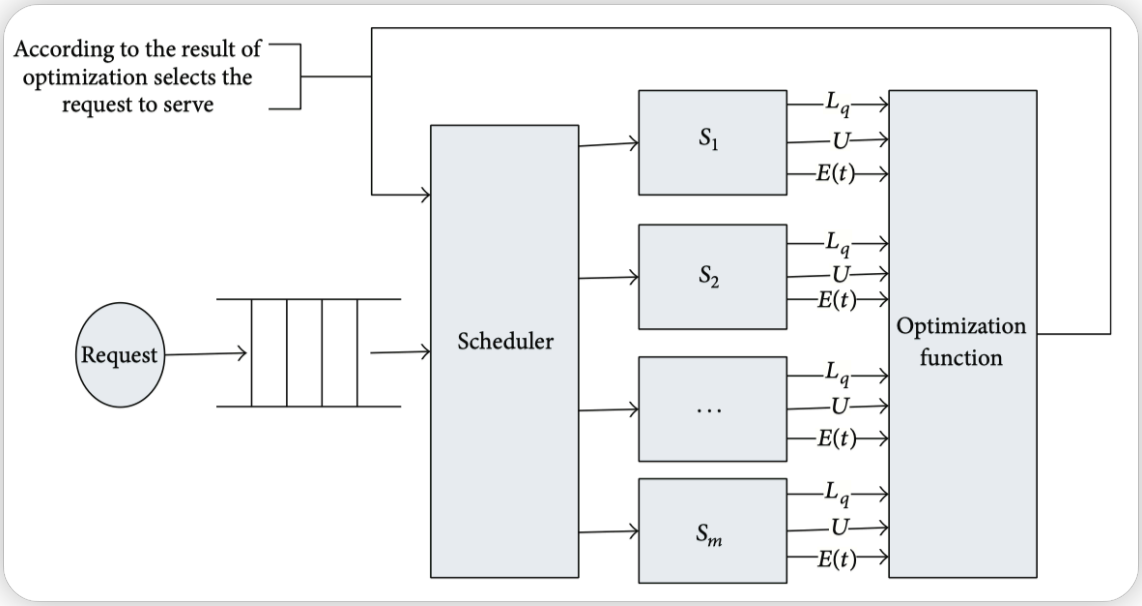

图 3.4 基于队列系统的任务调度和性能优化模型

优化策略如下：所有请求都会到达调度器，根据优化函数的结果选择将请求发送到服务器。优化函数的输入参数是云计算中每个服务器的时间平均值、利用率和等待客户总数。设 p_i 是 i 个用户被选择并执行的概率，t_i 是 i 个用户的服务时间，因此，平均时

间 $E(t)$ 如下：

$$E[t] = \sum_{i=1}^{m} p_i * t_i \tag{10}$$

为了获得优化结果，并保证负载均衡，优化函数如下：

$$\text{fcn} = \alpha * E[t] + \beta * L_q + \chi * U_i \tag{11}$$

其中，α, β 和 χ 是可以通过训练获得的系数，而 $E[t]$,L_q 和 U 分别是平均时间，队列中的用户数和每个服务器的利用率。利用率 $U_i = \lambda_i/\mu_i$，其中 λ_i 是优化函数选择的客户到达率，μ_i 是服务器 S_i 的服务率。优化函数根据填充参数计算函数值，然后按升序排序。调度器根据优化函数的结果选择服务器来执行服务。根据最优策略，优化算法在算法中进行描述。

根据最优策略，优化算法在算法1中进行描述。

Algorithm 1: 优化算法

input : request_list, server_list
output: server_number

for server in server_list do
    $E(t)$ = getvalue$E$(server_number);
    L_q = getvalueL_q(server_number);
    $U = \lambda_i$ / μ_i;
for for server in server_list do
    function_value = $\alpha * E(t)_i + \beta * (L_q)_i + \chi * U$;
ascending sort of the valued of function_value ;
getting the index of the first function_value ;
scheduler schedules the request to the select_server ;

四、仿真与结果分析

4.1 仿真设置

为了验证我们的优化策略的性能，我们使用 MATLAB 的离散事件工具。所有实验都在 AMD Phenom II X4 B95 3.0 GHz，2 G RAM 的微软 Windows XP 环境下进行测试，所有实验都在 MATLAB R2009b 中实现。我们按照亚马逊标准的按需实例将请求分为四类。

请求的到达时间间隔遵循指数分布，并且平均到达率为 $\{\lambda_1 = 30, \lambda_2 = 20, \lambda_3 = 15, \lambda_4 = 12\}$。服务也遵循指数分布，且平均服务率为 $\{\mu_1 = 3, \mu_2 = 2.4, \mu_3 = 2, \mu_4 = 1.71$

}。优化函数的系数分别是 $\{\alpha = 0.1, \beta = 0.4, \chi = 2\}$；总的测试时间是一天 (1440 分钟)，服务器的数量是 {4, 20, 40, 60, 80}。

4.2 仿真结果和分析

我们将我们的优化方法作为合成优化 (SO) 的优先队列。为了比较和分析优化策略的性能，我们使用经典的短时优先的优先级队列和先进先出 (FIFO) 排队策略作为测试度量。我们使用优先队列块来实现基于优化策略的排队策略，并将两种优先排队策略的性能与 FIFO 排队策略进行比较。

表 4.1 平均等待时间

Service number	FIFO	SSF	SO	
4	705.2	114.8	688.2	
20	475.5	43.42	304.6	
40	166.6	47.4	0.59	
60	2.64	2.64	48	0.025
			52	0.0010
80	0.06	0.06	60	0

表 4.2 平均队列长度

Service number	FIFO	SSF	SO	
4	892.3	892.3	890.6	
20	607.3	607.3	398.1	
40	206.2	206.2	0.76	
60	3.39	3.39	48	0.032
			52	0.0013
80	0.077	0.077	60	0

表 4.3 服务客户量

Service number	FIFO	SSF	SO
4	101	101	102
20	667	667	1083
40	1517	1517	1848
60	1848	1848	1848
80	1848	1848	1848

首先，我们测试了服务器数量分别为 {4, 20, 40, 60, 80} 时的平均等待时间。三种策略的结果如表4.1所示。根据表4.1，当服务器数量为 4 和 20 时，SSF 的平均等待时间最少；但是，当服务器数量为 40，60 和 80 时，SO 策略的平均等待时间最少。当服务器少的时候，许多用户被迫在队列中等待；另一方面，短时服务首先被完成，短时服务的服务时间小于其他策略的服务时间。因此，SSF 策略的平均等待时间是最佳的。然而，当服务增加时，即服务器数量为 40，60 和 80 时，SO 策略综合优化了利用率，平均等待时间和平均队列长度，并使每个服务器能力发挥最佳效果。因此，平均等待时间比其他策略短。与 SO 策略相比，当服务增加时，SSF 和 FIFO 策略无法顾全每个服务器的利用率和队列长度。因此，某些服务器可能无法完全运行，而其他服务器可能过载，导致更长的平均时间。

另一方面，表4.2和表4.3显示了三种策略的平均队列长度和服务客户数。上面两张表表示 SO 的平均队列长度和服务客户数量比其他两种策略更好。尽管 SSF 的平均等待时间在服务率为 4 和 20 时很好，但 SSF 策略的平均队列长度和服务顾客数量并不比 SO 策略更好。原因如下：服务器太少时，无法为到达的顾客提供足够的服务，大多数顾客不得不排队等候。当服务人员增加时，SO 可以优化平均等待时间、平均队列长度和利用率。因此，SO 能够获得实现更多的用户服务，平均队列长度更短。

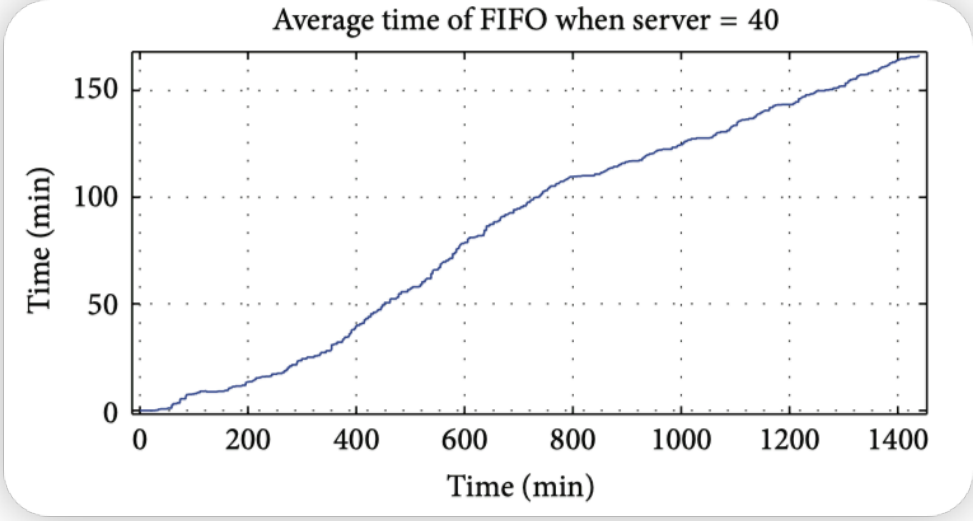

图 4.1 平均等待时间 [FIFO]

为了清楚地展示队列的平均等待时间，当服务器数为 40 时，三种策略的曲线图在图4.1、4.2和4.3中给出。当顾客请求服务太多而 FIFO 服务策略无法提供足够的服务时，图4.1显示平均等待时间几乎是随着时间的推移而线性增加的。

虽然平均等待时间也有所增加，但与图4.1相比，图4.2明显有所改善。图4.3表明，随着时间的推移，平均等待时间先上升后下降。原因如下：由于用户增加，首先服务器的利用率没有显著提高，因此等待时间会增加；随着服务器利用率的增加，等待时间也会减少。

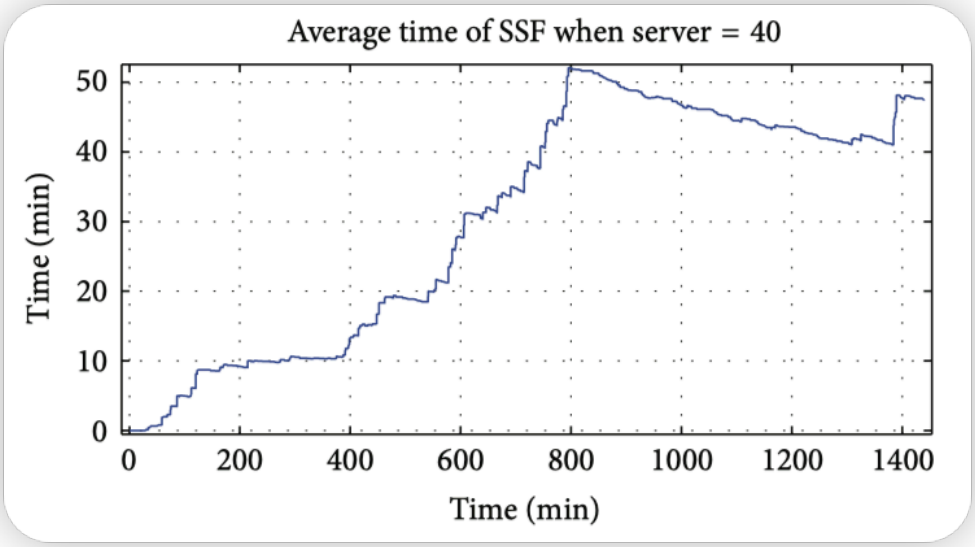

图 4.2 平均等待时间 [SSF]

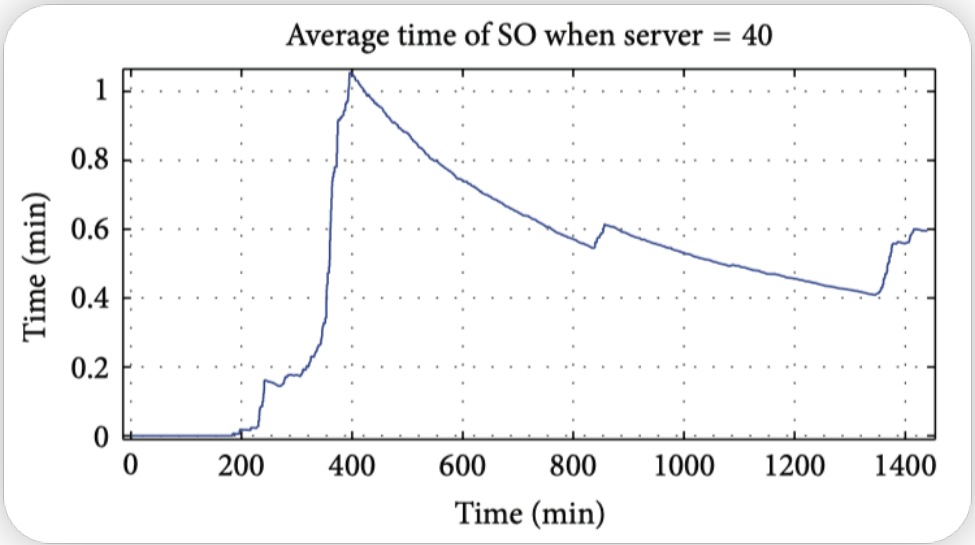

图 4.3 平均等待时间 [SO]

五、总结与展望

我们研究了在云计算中优化性能的方法，并给出了评估服务的参数。通过模拟，我们可以回答以下问题：(1) 对于给定的服务数量和客户到达率，可以获得多少水平的QoS？(2) 对于给定的 QoS 要求和客户到达率，用多少服务器能够满足 QoS？(3) 对于给定的服务器数量和客户到达率，有多少客户能够获得服务？在本文中，为了分析云计算中服务的性能，我们提出了一种排队模型，并开发了一种综合优化方法来优化服务性能。我们进一步进行了模拟以验证我们的优化方法。模拟结果表明，所提出的方法可以减少等待时间和队列长度，并使更多的客户获得服务。为了在真实的云计算环境中评估所提出的系统，我们计划通过扩展真实的云平台（如 OpenStack）来实现它。此外，如果将云计算建模为 M/M/G，则可以适用于不同的服务时间。我们将来会研究这方面。目前，在云计算中，数据中心的电力消耗非常巨大；因此未来研究的另一个方向是优化性能和能耗。此外，这项工作具有社会重要性，它不仅可以降低正在进行的运营成本，还可以减少二氧化碳。

参考文献

[1] VAQUERO L M, RODERO-MERINO L, CACERES J, et al. A break in the clouds: towards a cloud definition[J]. Computer Communication Review, 2008, 39(1): 50-55.

[2] MELL P, GRANCE T. The nist definition of cloud computing[EB/OL]. 2008. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[3] WANG L, VON LASZEWSKI G, YOUNG A, et al. Cloud computing: a perspective study[J]. New Generation Computing, 2010, 28(2): 137-146.

[4] KLEINROCK L. Queueing systems: Theory: Vol. 1[M]. New York, NY, USA: Wiley-Interscience, 1975.

[5] Amazon elastic compute cloud[M/OL]. Amazon Web Service LLC or its affiliate, 2010. http://aws.amazon.com/documentation/ec2.

[6] LU W, VARNA A L, WU M. Security analysis for privacy preserving search of multimedia[C]// Proceedings of the 17th IEEE International Conference on Image Processing (ICIP ’10). Hong Kong, 2010: 2093-2096.

[7] MENZEL M, WARSCHOFSKY R, THOMAS I, et al. The service security lab: a model-driven platform to compose and explore service security in the cloud[C]//Proceedings of the 6th IEEE World Congress on Services (Services-1 ’10). Miami, Fla, USA, 2010: 115-122.

[8] HUANG D, ZHANG X, KANG M, et al. Mobicloud: building secure cloud framework for mobile computing and communication[C]//Proceedings of the 5th IEEE International Symposium on Service-Oriented System Engineering (SOSE ’10). Nanjing, China, 2010: 27-34.

[9] LI J, WANG Q, WANG C, et al. Fuzzy keyword search over encrypted data in cloud computing[C]// Proceedings of the IEEE INFOCOM. San Diego, Calif, USA, 2010: 1-5.

[10] WANG C, WANG Q, REN K, et al. Privacy-preserving public auditing for data storage security in cloud computing[C]//Proceedings of the IEEE INFOCOM. San Diego, Calif, USA, 2010: 1-9.

[11] YUAN H, KUO C C J, AHMAD I. Energy efficiency in data centers and cloud-based multimedia services: an overview and future directions[C]//Proceedings of the International Conference on Green Computing. Chicago, Ill, USA, 2010: 375-382.

[12] LIN W, QI D. Research on resource self-organizing model for cloud computing[C]//Proceedings of the IEEE International Conference on Internet Technology and Applications (ITAP '10). Wuhan, China, 2010: 1-5.

[13] TENG F, MAGOULE'S F. Resource pricing and equilibrium allocation policy in cloud computing[C]// Proceedings of the 10th IEEE International Conference on Computer and Information Technology (CIT '10). Bradford, UK, 2010: 195-202.

[14] SHI H, ZHAN Z. An optimal infrastructure design method of cloud computing services from the bdim perspective[C]//Proceedings of the 2nd Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA '09). Wuhan, China, 2009: 393-396.

[15] LI L Q. An optimistic differentiated service job scheduling system for cloud computing service users and providers[C]//Proceedings of the 3rd International Conference on Multimedia and Ubiquitous Engineering (MUE '09). Qingdao, China, 2009: 295-299.

[16] XIONG K, PERROS H. Service performance and analysis in cloud computing[C]//Proceedings of the 5th IEEE World Congress on Services (Services-1 ' 09). Los Angeles, Calif, USA, 2009: 693-700.

[17] SHI Y, JIANG X, YE K. An energy-efficient scheme for cloud resource provisioning based on cloudsim [C]//Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '11). Austin, Tex, USA, 2011: 595-599.

[18] NAN X M, HE Y F, GUAN L. Optimal resource allocation for multimedia cloud based on queuing model[C]//Proceedings of the 13th IEEE International Workshop on Multimedia Signal Processing (MMSP ' 11). Hangzhou, China, 2011: 1-6.

[19] KHAZAEI H,´ J M,´ V B M. Modelling of cloud computing centers using m/g/m queues[C]// Proceedings of the 31st IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW ' 11). Minneapolis, Minn, USA, 2011: 87-92.

[20] KHAZAEI H, MISIC J, MISIC V B. Performance analysis of cloud computing centers using m/g/m/m+r queuing systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23 (5): 936-943.

[21] GOSWAMI V, PATRA S S, MUND G B. Performance analysis of cloud with queue-dependent virtual machines[C]//Proceedings of the 1st IEEE International Conference on Recent Advances in Information Technology (RAIT ' 12). Dhanbad, India, 2012: 357-362.

[22] ELLENS W, ZIVKOVIC M, AKKERBOOM J, et al. Performance of cloud computing centers with multiple priority classes[C]//Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD ' 12). Honolulu, Hawaii, USA, 2012: 245-252.

[23] Amazon ec2 pricing[EB/OL]. http://aws.amazon.com/ec2/pricing/.

附录

A仿真结果

- **‘code/distribute‘**: 请求的分布图
- **‘code/record‘**: 请求的信息记录表
- **‘code/waitTime‘**: 平均等待时间的时序图
- **‘code/result.xlsx‘**: 平均等待时间、平均队列长度、服务客户量

B仿真代码

B.1 main.py

```
import datetime
from lib import simulate, MixedRequestGenerator

startTime = datetime.datetime.now()
# 生成请求并绘制分布
mixedRequestGenerator = MixedRequestGenerator()
mixedRequestGenerator.plotDistribute(save=True)

# 仿真并记录数据
for method in ["FIFO", "SSF", "SO"]:
    for serverNum in [4, 20, 40, 60, 80, 48, 52]:
        simulate(method=method, serverNum=serverNum)
        print("============================================================")

endTime = datetime.datetime.now()
print(f" 运行时间: {endTime - startTime}")
```

B.2 lib.py

```
from functools import reduce, cache, cached_property

from colorama import Fore, Style
from scipy.stats import expon
import simpy
from simpy.core import Environment
from matplotlib import pyplot as plt
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']  #MacOS 自带字体
plt.rcParams['axes.unicode_minus']=False # 用来正常显示负号
import seaborn as sns
import numpy as np
import pandas as pd
```

```python
from pandas import Series, DataFrame

SIMULATION_TIME = 1440
REFER_RATIO = 0.1
RANDOM_SEED = 123
ACCURACY = 0.01

# 用每分钟表达的到达率、服务率
LAMBDA_LIST = [30, 20, 15, 12]
MU_LIST = [3, 2.4, 2, 1.71]
# 用每小时表达的到达率、服务率
LAMBDA_LIST = [each/60 for each in LAMBDA_LIST]
MU_LIST = [each/60 for each in MU_LIST]

KIND_LIST = ['A', 'B', 'C', 'D']

ALPHA = 0.1
BETA = 0.4
CHI = 2


def customPrint(actualTime, message, tip=None):
    if tip is None:
        print(Fore.YELLOW + f"{format(actualTime,'.2f')}: " + Style.RESET_ALL, message)
    else:
        print(Fore.YELLOW + f"{format(actualTime,'.2f')}: " + Style.RESET_ALL, message,
        Fore.GREEN + "\t"*3 + f"[{tip}]" + Style.RESET_ALL)

def generate_random_color():
    r = np.random.random()
    g = np.random.random()
    b = np.random.random()
    return (r,g,b)


class Request:
    def __init__(self, intervalTime, arrivalTime, serviceTime, kind: str, requestId: int):
        self.intervalTime = intervalTime
        self.arrivalTime = arrivalTime
        self.serviceTime = serviceTime
        self.kind = kind
        self.requestId = requestId

        self._serverId = None
        self._isScheduled = False # 被安排
        self._scheduledTime = np.nan
        self.inProcess = False # 服务中
        self.processingTime = np.nan # 开始处理的时刻
        self.isServed = False # 服务完成
        self._finishTime = np.nan

        # NOTE: 对于 SO, arrivalTime 非常接近 scheduledTime
        self._waitTime = SIMULATION_TIME - self.arrivalTime

    @property
    def isScheduled(self):
        return self._isScheduled
```

```
77
78      @property
79      def serverId(self):
80          return self._serverId
81
82      @serverId.setter
83      def serverId(self, id):
84          self._serverId = id
85
86      @property
87      def finishTime(self):
88          return self._finishTime
89
90      @finishTime.setter
91      def finishTime(self, time):
92          self._finishTime = time
93          self._waitTime = self._finishTime - (self.arrivalTime + self.serviceTime)
94
95      @property
96      def waitTime(self):
97          return self._waitTime
98
99      @property
100     def scheduledTime(self):
101         return self._scheduledTime
102
103     @scheduledTime.setter
104     def scheduledTime(self, time):
105         self._isScheduled = True
106         self._scheduledTime = time
107
108
109 class RequestGenerator:
110     def __init__(self, lambda_, mu, kind):
111         self.lambda_ = lambda_
112         self.mu = mu
113         self.kind = kind
114         self.rvIntervalTime = expon(scale=1/lambda_)
115         self.rvServiceTime = expon(scale=1/mu)
116
117     @cache
118     def generateRequestList(self):
119         np.random.seed(RANDOM_SEED)
120         size = int(SIMULATION_TIME * self.lambda_ * 1.1) # 留余量，待截取
121         intervalTimeArray = self.rvIntervalTime.rvs(size=size)
122         arrivalTimeArray = intervalTimeArray.cumsum()
123         serviceTimeArray = self.rvServiceTime.rvs(size=size)
124
125         # NOTE: 截断仿真结束时还未生成的请求
126         num = np.where(arrivalTimeArray <= SIMULATION_TIME)[0][-1]
127
128         return [ Request(intervalTimeArray[i], arrivalTimeArray[i], serviceTimeArray[i], self.kind, i)
129             for i in range(num) ]
130
131
132 class MixedRequestGenerator:
133     def __init__(self, lambda_list=LAMBDA_LIST, mu_list=MU_LIST, kind_list=KIND_LIST):
```

```
134         self.requestGeneratorList = []
135         self.requestLists = []
136 
137         for lambda_, mu, kind in zip(lambda_list, mu_list, kind_list):
138             requestGenerator = RequestGenerator(lambda_, mu, kind)
139             self.requestGeneratorList.append(requestGenerator)
140             requestList = requestGenerator.generateRequestList()
141             self.requestLists.append(requestList)
142 
143         mixedRequestList = reduce(lambda x,y: x + y, self.requestLists)
144         self.mixedRequestList = sorted(mixedRequestList, key=lambda x: x.arrivalTime)
145 
146     def plotDistribute(self, save=False):
147         for ind in range(len(KIND_LIST)):
148             kind = KIND_LIST[ind]
149             intervalTimeList = [request.intervalTime for request in self.requestLists[ind]]
150             serviceTimeList = [request.serviceTime for request in self.requestLists[ind]]
151             self.plotSingleDistribute(kind, intervalTimeList, serviceTimeList, save=save)
152 
153         kind = "Mixed"
154         intervalTimeList = [request.intervalTime for request in self.mixedRequestList]
155         serviceTimeList = [request.serviceTime for request in self.mixedRequestList]
156         self.plotSingleDistribute(kind, intervalTimeList, serviceTimeList, save=save)
157 
158     def plotSingleDistribute(self, kind, intervalTimeList, serviceTimeList, save=False):
159         """TODO: 绘制每个请求的 kde 和总的 hist"""
160         with plt.style.context('_mpl-gallery'):
161             colors = [ generate_random_color() for i in range(4) ]
162             fig, axes = plt.subplots(2, 2)
163             axes[0,0].hist(intervalTimeList, color=colors[0])
164             axes[0,0].set_xlabel(" 间隔时间")
165             axes[0,0].set_ylabel(" 频数")
166             sns.kdeplot(intervalTimeList, ax=axes[0,1], color=colors[1])
167             axes[0,1].set_xlabel(" 间隔时间")
168             axes[0,1].set_ylabel(" 概率密度")
169             axes[1,0].hist(serviceTimeList, color=colors[2])
170             axes[1,0].set_xlabel(" 服务时间")
171             axes[1,0].set_ylabel(" 频数")
172             sns.kdeplot(serviceTimeList, ax=axes[1,1], color=colors[3])
173             axes[1,1].set_xlabel(" 服务时间")
174             axes[1,1].set_ylabel(" 概率密度")
175             fig.set_size_inches(20,12)
176             fig.suptitle(f"{kind}类请求的间隔时间和服务时间分布", fontsize=20)
177             fig.tight_layout()
178             if save:
179                 plt.savefig(f"distribute/distribute{kind}.png")
180             else:
181                 plt.show()
182 
183 
184 class Server:
185     def __init__(self, env: Environment, serverId: int, maxQueueLength=None, prioritized=False):
186         self._queue = []
187         self.env = env
188         self.serverId = serverId
189         self.maxQueueLength = maxQueueLength
190         self.prioritized = prioritized
```

```
191
192     def isEmpty(self):
193         if len(self._queue) == 0:
194             return True
195         else:
196             return False
197
198     def isFull(self):
199         if self.maxQueueLength is None:
200             return False
201         if len(self._queue) == self.maxQueueLength:
202             return True
203         else:
204             return False
205
206     @property
207     def queueLength(self):
208         return len(self._queue)
209
210     @property
211     def currentRequest(self):
212         if self.isEmpty():
213             return None
214         else:
215             if not self.prioritized:
216                 return self._queue[0]
217             else:
218                 fastFirstRequest = min(self._queue, key=lambda request: request.kind)
219                 return fastFirstRequest
220
221     def run(self):
222         while True:
223             if not self.isEmpty():
224                 self.currentRequest.inProcess = True
225                 self.currentRequest.processingTime = self.env.now
226                 yield self.env.timeout(self.currentRequest.serviceTime)
227                 self.currentRequest.inProcess = False
228                 self.currentRequest.finishTime = self.env.now
229                 self.currentRequest.isServed = True
230                 kind = self.currentRequest.kind
231                 requestId = self.currentRequest.requestId
232                 # customPrint(self.env.now, f"request-{kind}{requestId} 完成于 server-{self.serverId}",
233                 #     f" 若无延迟: {format(self.currentRequest.arrivalTime + self.currentRequest.serviceTime, '.2f')}")
234                 self.remove(self.currentRequest)
235             else:
236                 yield self.env.timeout(ACCURACY)
237
238     def append(self, request: Request):
239         self._queue.append(request)
240
241     def remove(self, request: Request):
242         self._queue.remove(request)
243
244     def __len__(self):
245         return len(self._queue)
246
247
```

```python
class Cloud:
    def __init__(self, env: Environment, serverNum, method: str="FIFO"):
        # 方法可选：FIFO, SSF, 其他 (SO)
        self.env = env
        self.serverNum = serverNum
        self.method = method
        if self.method == "FIFO" or self.method == "SSF":
            self.serverList = [Server(env, serverId=i, maxQueueLength=1) for i in range(serverNum)]
        else:
            self.serverList = [Server(env, serverId=i) for i in range(serverNum)]
        self.mixedRequestGenerator = MixedRequestGenerator()
        self.recordInterval = 1
        self.queueLengthList = []
        self.averageWaitTimeList = []

    def getRequestQueue(self):
        """ 分配器端的队列"""
        requestList = self.mixedRequestGenerator.mixedRequestList
        return [request for request in requestList if request.arrivalTime < self.env.now and not request.isScheduled]

    def getServerRequestQueue(self):
        """ 服务器端的队列"""
        queueList = []
        for server in self.serverList:
            if len(server) > 1:
                queue = server._queue[1:]
            else:
                queue = []
            queueList.append(queue)
        return queueList

    def record(self):
        while True:
            yield self.env.timeout(1)
            if self.method == "FIFO" or self.method == "SSF":
                queueLength = len(self.getRequestQueue())
            else:
                queueLength = sum([len(queue) for queue in self.getServerRequestQueue()])
            self.queueLengthList.append(queueLength)

            now = self.env.now
            arrivedRequestList = [request for request in self.mixedRequestGenerator.mixedRequestList if request.arrivalTime < now ]
            # 处理完成的列表
            servedWaitTimeList = [ request.waitTime for request in arrivedRequestList if request.isServed]
            # 正在处理的列表
            processingWaitTimeList = [ request.processingTime - request.arrivalTime for request in arrivedRequestList if request.inPro
            # 已分配，未处理
            unProcessingWaitTimeList = [ now - request.arrivalTime for request in arrivedRequestList if not request.inProcess and not
            # 还未被分配的列表
            unScheduledWaitTimeList = [ now - request.arrivalTime for request in arrivedRequestList if not request.isScheduled ]
            waitTimeList = servedWaitTimeList + processingWaitTimeList + unProcessingWaitTimeList + unScheduledWaitTimeList
            if len(waitTimeList)==0:
                self.averageWaitTimeList.append(0)
            else:
                self.averageWaitTimeList.append(np.mean(waitTimeList))

    def arrive(self):
```

```
        lastArrivalTime = 0
        for request in self.mixedRequestGenerator.mixedRequestList:
            kind = request.kind
            requestId = request.requestId

            arrivalTime = request.arrivalTime
            mixedIntervalTime = arrivalTime - lastArrivalTime
            yield self.env.timeout(mixedIntervalTime)
            lastArrivalTime = arrivalTime

            # customPrint(self.env.now, f"request-{kind}{requestId} 到达")

    def schedule(self):
        if self.method == "FIFO":
            while True:
                requestQueue = self.getRequestQueue()
                for request in requestQueue:
                    kind = request.kind
                    requestId = request.requestId
                    while True:
                        for server in self.serverList:
                            if not server.isFull():
                                server.append(request)
                                request.scheduledTime = self.env.now
                                request.serverId = server.serverId
                                # customPrint(self.env.now, f"request-{kind}{requestId} 委派给 server-{server.serverId}")
                                break
                        if request.isScheduled:
                            break
                        yield self.env.timeout(ACCURACY)
                yield self.env.timeout(ACCURACY)
        elif self.method == "SSF":
            while True:
                requestQueue = self.getRequestQueue()
                if len(requestQueue) > 0:
                    fastFirstRequest = min(requestQueue, key=lambda request: request.kind)
                    kind = fastFirstRequest.kind
                    requestId = fastFirstRequest.requestId
                    while True:
                        for server in self.serverList:
                            if not server.isFull():
                                server.append(fastFirstRequest)
                                fastFirstRequest.scheduledTime = self.env.now
                                fastFirstRequest.serverId = server.serverId
                                # customPrint(self.env.now, f"request-{kind}{requestId} 委派给 server-{server.serverId}")
                                break
                        if fastFirstRequest.isScheduled:
                            break
                        yield self.env.timeout(ACCURACY)
                else:
                    yield self.env.timeout(ACCURACY)
        else:
            while True:
                requestQueue = self.getRequestQueue()
                if len(requestQueue) > 0:
                    firstRequest = requestQueue[0]
                    kind = firstRequest.kind
```

```
362                     requestId = firstRequest.requestId
363 
364                     # 获取最佳服务器 ID
365                     valueList = []
366                     for server in self.serverList:
367                         serverId = server.serverId
368                         requestQueue = self.getServerRequestQueue()[serverId]
369                         L = len(requestQueue)
370                         mixedRequestList = self.mixedRequestGenerator.mixedRequestList
371 
372                         endTime = self.env.now
373                         fromTime = endTime - SIMULATION_TIME*REFER_RATIO
374                         referredScheduledRequestList = [request for request in mixedRequestList
375                             if request.serverId == serverId and
376                             request.scheduledTime < endTime and request.scheduledTime > fromTime ]
377                         referredServedRequestList = [request for request in mixedRequestList
378                             if request.serverId == serverId and
379                             request.finishTime < endTime and request.finishTime > fromTime ]
380 
381                         if len(referredServedRequestList) == 0:
382                             E = 0
383                         else:
384                             E = np.mean([request.serviceTime for request in referredServedRequestList])
385 
386                         if len(referredServedRequestList) == 0:
387                             U = 0
388                         else:
389                             U = len(referredScheduledRequestList)/len(referredServedRequestList)
390                         value = ALPHA * E + BETA * L + CHI * U
391                         valueList.append(value)
392                         selectedServerId = valueList.index(min(valueList))
393                     # 获得最佳服务器
394                     for server in self.serverList:
395                         if server.serverId == selectedServerId:
396                             selectedServer = server
397 
398                     selectedServer.append(firstRequest)
399                     firstRequest.scheduledTime = self.env.now
400                     firstRequest.serverId = selectedServerId
401                     # customPrint(self.env.now, f"request-{kind}{requestId} 委派给 server-{selectedServerId}")
402                 yield self.env.timeout(ACCURACY)
403 
404     def start(self):
405         # 运行服务
406         for server in self.serverList:
407             self.env.process(server.run())
408         # 生成请求
409         self.env.process(self.arrive())
410         # 分配请求
411         self.env.process(self.schedule())
412         # 记录数据
413         self.env.process(self.record())
414 
415     def exportRecord(self):
416         mixedRequestList = self.mixedRequestGenerator.mixedRequestList
417         DataFrame({
418             "id": [request.kind+str(request.requestId) for request in mixedRequestList],
```

```
            "serverId": [request.serverId for request in mixedRequestList],
            "isScheduled": [request.isScheduled for request in mixedRequestList],
            "inProcess": [request.inProcess for request in mixedRequestList],
            "isServed": [request.isServed for request in mixedRequestList],
            "arrivalTime": [round(request.arrivalTime,2) for request in mixedRequestList],
            "scheduledTime": [round(request.scheduledTime,2) for request in mixedRequestList],
            "processingTime": [request.processingTime for request in mixedRequestList],
            "finishTime": [round(request.finishTime,2) for request in mixedRequestList],
            "serviceTime": [round(request.serviceTime,2) for request in mixedRequestList],
            "waitTime": [ round(request.waitTime,2) for request in mixedRequestList],
        }).to_csv(f"record/record{self.method}-{self.serverNum}.csv")
        for ind in range(len(KIND_LIST)):
            kind = KIND_LIST[ind]
            requestList = self.mixedRequestGenerator.requestLists[ind]
            DataFrame({
            "id": [request.kind+str(request.requestId) for request in requestList],
            "serverId": [request.serverId for request in requestList],
            "isScheduled": [request.isScheduled for request in requestList],
            "inProcess": [request.inProcess for request in requestList],
            "isServed": [request.isServed for request in requestList],
            "arrivalTime": [round(request.arrivalTime,2) for request in requestList],
            "scheduledTime": [round(request.scheduledTime,2) for request in requestList],
            "processingTime": [request.processingTime for request in requestList],
            "finishTime": [round(request.finishTime,2) for request in requestList],
            "serviceTime": [round(request.serviceTime,2) for request in requestList],
            "waitTime": [ round(request.waitTime,2) for request in requestList],
            }).to_csv(f"record/record{self.method}-{self.serverNum}{kind}.csv")

    def plotAverageWaitTime(self, save=False):
        fig = plt.figure(figsize=(20,12))
        with plt.style.context('ggplot'):
            plt.plot(self.averageWaitTimeList)
            plt.xlabel(r" 时间 ($min$)")
            plt.ylabel(r" 等待时间 ($min$)")
            plt.title(f" 平均等待时间-{self.method}-{self.serverNum}", fontsize=20)
            if save:
                plt.savefig(f"waitTime/waitTime-{self.method}-{self.serverNum}.png")
            else:
                plt.show()

    def getResults(self):
        mixedRequestList = self.mixedRequestGenerator.mixedRequestList
        # 保存记录
        self.exportRecord()
        # 绘制等待时间图
        self.plotAverageWaitTime(save=True)
        # 打印参数和结果
        mixedRequestNum = len(mixedRequestList)
        servedMixedRequestList = [request for request in mixedRequestList if request.isServed]
        servedMixedRequestNum = len(servedMixedRequestList)
        meanWaitTime = np.mean([request.waitTime for request in mixedRequestList])
        meanQueueLength = np.mean(self.queueLengthList)
        print(f" 优化方法: {self.method}")
        print(f" 服务器数: {self.serverNum}")
        print(f" 仿真时间: {SIMULATION_TIME}")
        for ind in range(len(KIND_LIST)):
            kind = KIND_LIST[ind]
```

```
            lambda_ = LAMBDA_LIST[ind]
            mu = MU_LIST[ind]
            requestNum = len(self.mixedRequestGenerator.requestLists[ind])
            print(f" 请求生成器{kind}", end=": ")
            print(f" 参数: 到达率-{format(lambda_,'.2f')}, 服务率-{format(mu,'.2f')}, 生成请求数-{requestNum}")
        print(f" 平均等待时间: {format(meanWaitTime,'.2f')}")
        print(f" 平均队列长度: {format(meanQueueLength, '.2f')}")
        print(f" 总服务客户数: {servedMixedRequestNum}/{mixedRequestNum}")


def simulate(method="FIFO", serverNum=4):
    env = simpy.Environment()
    cloud = Cloud(env, serverNum=serverNum, method=method)
    cloud.start()
    env.run(until=SIMULATION_TIME)
    cloud.getResults()


if __name__ == "__main__":
    simulate(method="SSF", serverNum=80)

```