CI Runner 性能分析

李明杰

摘要

分析复杂的计算机系统时，可以使用随机过程刻画任务到达和系统处理。本文整理了持续集成 (CI) 中的任务数据，使用排队论的 M/M/K 模型刻画 CI 实践中的任务处理过程。在现有任务负载下，为了控制等待概率在 5%，预计需要分配 8 个处理器核进行 CI 任务处理。Petri 网等工具可以更精细地刻画系统，但求解十分困难，本文未能得出进一步的结果。

1 背景

软件开发中，常常采用持续集成 (Continuous Integration, CI) 的实践，在代码检入版本控制服务器后，触发自动化的构建、测试、部署等操作，用于执行 CI 任务的程序称为 CI Runner。CI 的运行不像普通页面请求那么迅速，在使用过程中，会出现任务长时间排队得不到处理的情况。为此，本文试图确定合理的 CI 服务资源。

2 数据

2.1 数据搜集

在 2018 年秋季学期的软件工程课程中，使用 GitLab 提供版本控制服务，课程平台向学生开放公共 CI Runner，共有 39 支队伍的学生使用了公共 CI Runner，每支队伍 3-5 人。在 GitLab 的数据库中导出 ci_builds 表中公共 Runner 对应的数据，ci_builds 记录了 CI Runner 执行的每一项单独的任务。每一次代码检入到 GitLab 后，将根据配置文件创建若干个相互关联的任务，称为流水线 (Pipeline)，这些任务可能有依赖关系从而必须等待前置任务结束后才能执行，也可能没有依赖关系而可以同时执行。ci_builds 中的主要内容介绍如表1。

2.2 数据清理

提交 (Commit) 是版本控制系统中的基本单位，2018 年秋季学期的学生提交分布如图1，从图中可以看出，大量提交集中在周三下午到周四下午的时间段。这是由于每周周四下午是大部分项目的例会时间。为了确定最少需要的资源量，只需考虑任务高发期。另外，每一次任务可能会被重试多次，这常常是因为系统故障导致任务失败，学生尝试重新执行一次任务。系统内部的故障通常是性能下降的结果而非原因，系统外的故障无助于系统的性能评价，因此，系统故障导致失败的任务不予考虑。从而，数据的清理有如下标准，

1. 计算结束时间和开始时间的差值作为任务执行时间，去掉未开始的任务；
2. 选择项目的主要开发时间内的 CI 任务，9 月 28 日至 11 月 30 日；
3. 选择进入队列的时间在周三 16 点到次日凌晨 3 点，和周四 9 点到当天 17 点，两种时间段内的 CI 任务；
4. 同一次流水线的相同任务只保留一个，排队时间、开始时间以第一个任务为准，任务执行时间取最后一个任务。

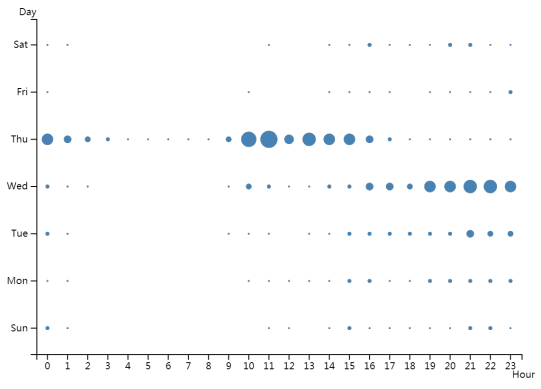


图 1: 提交分布图

数据库中共检索 47218 条任务，除去重试的任务后剩余 44280 条，而在高发期的任务数量有 16042 条。

表 1: GitLab 数据库的 ci_builds 表结构

列	类型	含义
project_id	整型	对应代码仓库的编号
user_id	整型	触发该任务的用户编号
commit_id	整型	该任务所属的流水线编号
name	字符串	该任务的名字，在一次流水线中不重复
status	字符串	任务的状态，取值有 “成功”、“失败”、“取消”
created_at	时间	创建时间，和流水线同时创建，相差不超过 1 秒
queued_at	时间	排队时间，有依赖任务时，依赖任务完成之后才开始排队
started_at	时间	开始执行时间
finished_at	时间	任务结束时间

3 建模

3.1 问题抽象

对于 CI Runner，输入为 CI 任务，工作内容为按照配置文件进行相应操作。不考虑计算机硬件的缓存，则限制 CI Runner 性能的因素主要包括如下几点，

计算 处理器频率越高，任务处理越快；处理器数量越多，同时处理的任务越多。

存储 存储空间不足将导致程序崩溃。

网络 CI 任务中会有网络通信，网络通信带宽和延迟会影响任务处理时间。

在上述因素中，储存空间必须严格大于运行时需要的上限以避免系统崩溃，因而直接使用历史数据估计需要的存储空间，在分析中认为存储空间不受限制。在获得的数据中，没有对 CI 任务需要的网络资源、存储进行记录，因而对这两个方面的分析也无法展开。假设使用的处理器有相同的频率，由于处理器数量限制了计算机同时运行的程序数量，之后的讨论中将只关注于处理器同时支持的 CI Runner 实例数量，$\#Server$。

图2a为 CI 任务到达时间间隔分布，其频率密度可以用指数分布进行刻画，如图2b。开发团队之间的工作相互独立，因而开发团队数量越多，CI 任务的到达越表现得和历史无关。

图3a为 CI 任务处理时间分布，在 3600 秒附近有密集分布，这些任务因为超时而被系统中断；忽略超时任务，其频率密度如图3b。每个代码仓库的 CI 任务的会有相对稳定的结构，例如，部署任务处理时间较短，构建任务大部分时间用于访问网络等；而不同代码仓库之间的 CI 任务的结构和用时会有差异。部分 CI 任务的执行时间小于 8 秒，其分布如表2，这部分的任务数量较少，且大部分失败。

表 2

成功 (success)	失败 (failed)	取消 (canceled)
52	156	49

此外，CI 任务的执行满足以下特点，

先入先出 先到达的任务优先执行。

持续工作 只要有任务到达，即开始工作。

非抢占 当前任务不会被新到达任务打断。

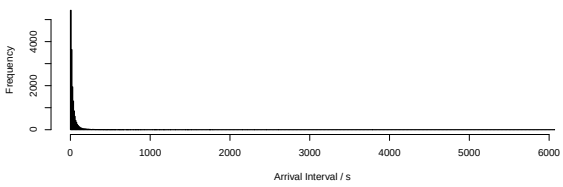
队列无限 CI 任务的队列信息存储在数据库中，在开始执行后才会大量占用资源。

3.2 M/M/K

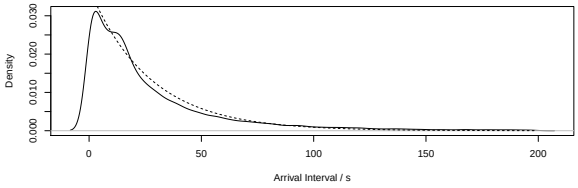
忽略 CI 任务之间的差异和依赖关系，以参数为 λ 的指数分布近似 CI 任务到达时间间隔分布，以参数为 μ 的指数分布近似 CI 任务处理时间分布，以未完成的 CI 任务数量为状态，设 CI Runner 实例数量为 K，则 CI Runner 的工作可以使用 M/M/K 模型来刻画，仅当 $\rho = \frac{\lambda}{K\mu} < 1$ 时稳态存在，式1为其稳态概率分布 [3]。稳态下，对于新到达的 CI 任务，等待概率为式2。

3.3 Petri 网

考虑不同队伍之间 CI 任务的分布特性差异，设共有 N 支队伍，使用 Petri 网刻画系统运行状态如图4。以参数为 λ_i 的指数分布近似队伍 i 的 CI 任务到达时间间隔分布，以参数为 μ_i 的指数

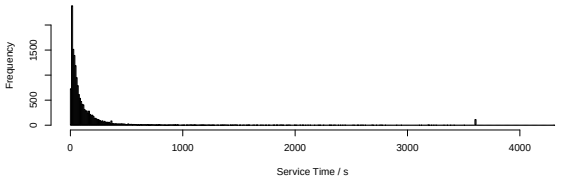


(a) 频数分布，$N = 16024$。

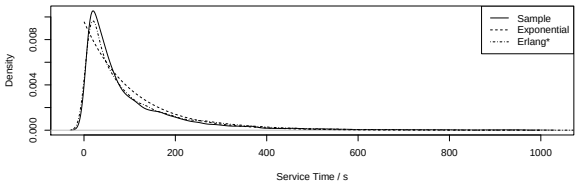


(b) 频率分布，$N = 15639$，0 附近曲线是绘图时平滑的结果。

图 2: 任务到达时间间隔分布。图2b中虚线为指数分布概率密度函数，参数使用到达时间间隔小于 200 秒的均值估计，$\lambda = 0.037$。



(a) 频数分布，$N = 16024$。



(b) 频率分布，$N = 15651$。

图 3: 任务处理时间分布。图3b中，指数分布概率密度函数的参数使用处理时间小于 1000 秒的均值估计，$\lambda = 0.0096$；使用相同部分数据的均值和方差估计 Erlang 分布的形状参数为 $\lambda_e = 0.63$，规模参数为 $\beta_e = 164.9$，Erlang 分布曲线向右平移 8 秒后得到图中曲线以拟合样本分布。

$$\pi_i = \begin{cases} \frac{\rho^i}{i!}\pi_0 & i \le K \\ \frac{\rho^K}{K!}\left(\frac{\rho}{k}\right)^{i-K}\pi_0 & i \ge K \end{cases}, \ \pi_0 = \left(\frac{\rho^K K^K}{K!}\frac{1}{1-\rho} + \sum_{i=0}^{K-1}\frac{(K\rho)^i}{i!}\right)^{-1} \tag{1}$$

$$P_{M/M/K,Wait} = \sum_{i=K}^{\infty}\pi_i = \sum_{i=K}^{\infty}\frac{\rho^j K^K}{K!}\pi_0 = \pi_0\frac{\rho^K K^K}{K!}\sum_{i=0}^{\infty}\rho^i = \pi_0\frac{\rho^K K^K}{K!}\frac{1}{1-\rho} \tag{2}$$

$$q_{(k,c,r),\eta} = \begin{cases} \lambda_i, & k > 0,\ \eta = (k-1, c, r+e_i) \\ \lambda_i, & k = 0,\ \eta = (k, c+e_i, r) \\ r_i\mu_i, & \sum_{j=1}^{N} c_j = 0,\ \eta = (k+1, c, r-e_i) \\ \alpha_{(k,c,r),\eta} r_i\mu_i, & \sum_{j=1}^{N} c_j > 0,\ \eta = (k, c-e_i, r) \end{cases} \tag{3}$$

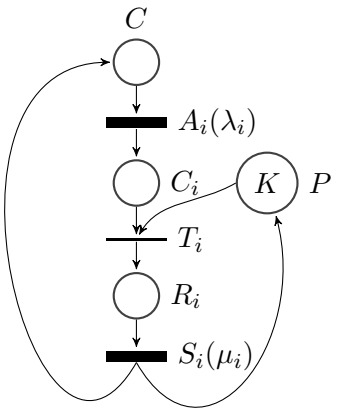


图 4: 每支队伍的 CI 任务执行模型。K 为 CI Runner 的实例数量；C 表示所有 CI 任务，拥有无限的标记；C_i 为这支队伍的 CI 任务等待队列，A_i 的延迟为任务到达时间，λ_i 为任务到达参数；R_i 为这支队伍 CI 任务的服务队列，S_i 的延迟为服务时间，μ_i 为任务服务时间参数。

分布近似队伍 i 的 CI 任务处理时间分布。需要注意，图中转移 T_i 是存在冲突的。

以 P 中空闲的 CI Runner 实例数量 k、各支队伍等待被执行的任务数量 c_i、各支队伍正在被执行的任务数量 r_i 构成向量 $\eta = (k, c, r)$ 为状态，初态为 $\eta_0 = (K, 0, 0)$，则转移速率矩阵定义如式3。其中，e_i 为一个有 N 个元素的行向量，第 i 个元素为 1，其余都为 0；当一个任务完成而队列中有任务等待时，$\alpha_{(k,c,r),\eta}$ 表示响应相应顾客的概率。

4 性能分析

4.1 M/M/K

对于 M/M/K 模型，使用数据估计参数，$\lambda = 0.037$，$\mu = 0.0096$。记

$$A = \frac{\rho^K K^K}{K!}\frac{1}{1-\rho},\ B = \sum_{i=0}^{K-1}\frac{(K\rho)^i}{i!}$$

$$P_{M/M/K,Wait} = \frac{A}{A+B}$$

以 $\frac{\lambda}{\mu}$ 和 CI Runner 实例数量为参数计算等待概率。

使用 M/M/K 计算不同实例数量下的等待概率，如图5a，在输入恒定的情况下，CI Runner 实例数量为 8 时可以将 CI 任务等待的概率控制在 5%。由 Possion 流的可加性，假定任务到达速率 λ 正比于队伍数量 N，以 $\lambda' = N\lambda$ 作为参数，估计固定 CI Runner 实例数量时，等待概率的分布，如图5b，该图用于估计队伍数量变化时，需要的 Runner 数量。

4.2 Petri 网

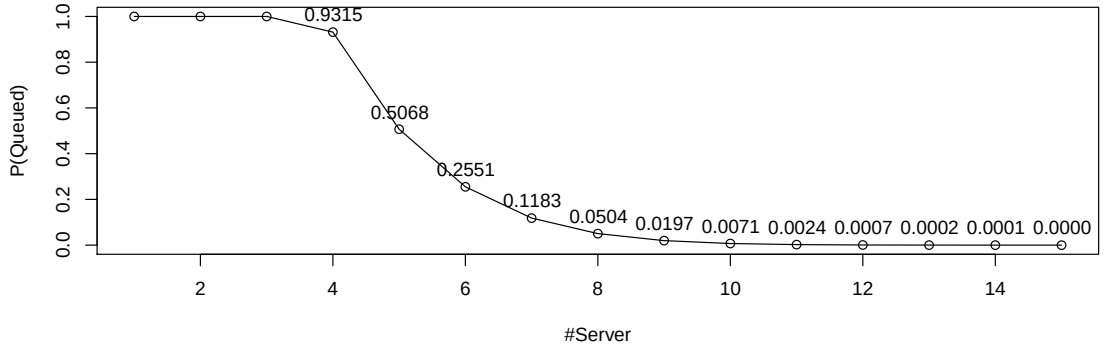
各队伍之间的统计特征差异如图6所示。通过对系统不同用户的区分，图4所示 Petri 网模型相比 M/M/K 模型更接近系统的真实情况。然而，建立起的 Petri 网中存在状态爆炸。仅考虑 $k + \sum_{i=1}^{N} r_i = K$ 这一限制条件，已经有 $\binom{K+N}{N}$ 个状态了，这使得模型的求解变得极为困难，模拟也难以在有限的时间内收敛。简单编程尝试迭代计算，在 $K = 1$ 时，从 $(1, 0, 0)$ 这一状态出发，仅迭代 5 轮即衍生出 4812991 个状态，继续计算将占用更多计算资源、耗费更多时间，而相邻两次迭代之间的差值变化很小。

5 总结

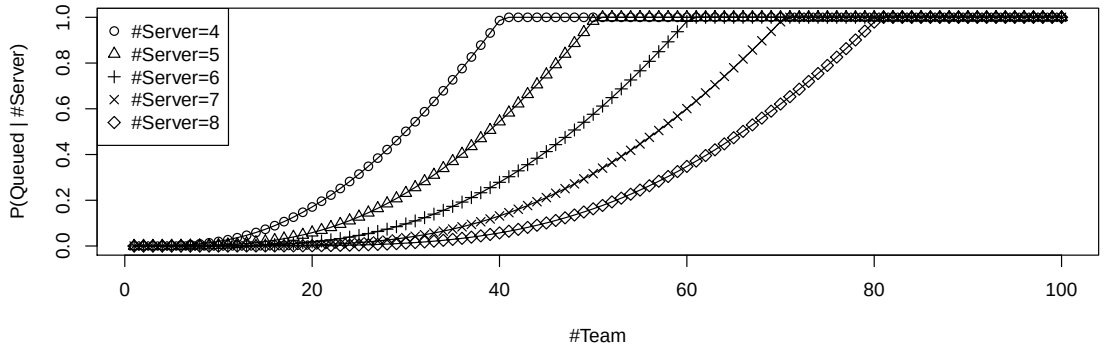
本文使用 M/M/K 队列模型刻画 CI 任务的到达情况，分析给定负载下需要的 CI Runner 实例数量，并进一步估计其它 CI 任务到达速率下需要的 CI Runner 实例数量。现有负载下，为了限制等待概率为 5%，需要为 CI Runner 至少提供 8 个核。在给定模型下，导致 CI 任务到达速率变化的队伍数量、软件开发的周期性，和导致 CI 处理时间变化的处理器频率等，均可以在已有模型的基础上计算 $\frac{\lambda}{\mu}$ 的变化系数，转化为队伍数量变化后，由图5b得到等待概率的估计。

该模型对问题的刻画是有偏差的。图3b表明，CI 任务处理时间与指数分布并不契合。不同队伍的到达速率也并不相同。

然而，当尝试使用 Petri 刻画系统行为时，面临无法求解的困难。用户的分类使得系统不再具有乘积解 [2]，而同一机器同时运行不同任务、用户特征多样却是分布式系统常见的特性。另一种尝试缩小模型和现实间差距的模型是分层排队网络 (LQN，Layered Queueing Network)，LQN 引入了对同步调用、异步调用、多进程等因素，并使用各种近似进行求解 [1]。当不得不使用运行时数据、并经过大量模拟计算才能得到期望的性能参数时，使用简单统计特征表征性能似乎是更合理的方式。

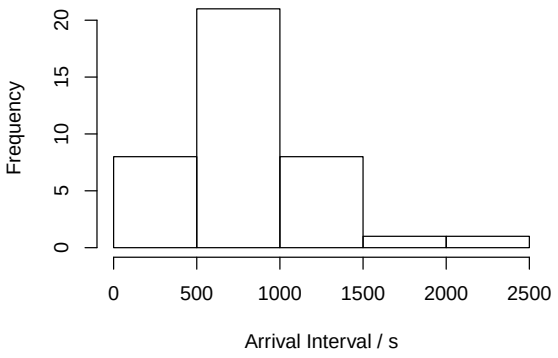


(a) 服务器数量-等待概率曲线

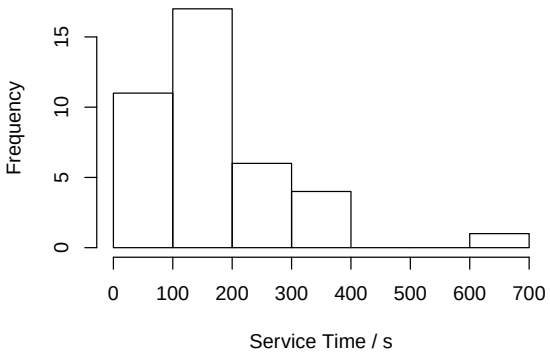


(b) 队伍数量-等待概率曲线

图 5: M/M/K 模型 CI 任务等待概率



(a) 各队伍任务到达平均时间间隔分布



(b) 各队伍任务平均服务时间分布

图 6: 队伍之间的统计差异

鸣谢

任丰原老师的课程对本文工作的开展提供了很大启发。

参考文献

[1] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering*, 35(2):148–161, mar 2009. doi: 10.1109/tse.2008.74. URL `https://doi.org/10.1109/tse.2008.74`.

[2] 任丰原. 计算机网络和计算机系统的性能评价, 2018. 清华大学研究生课程.

[3] 华兴. 排队论与随机服务系统, chapter 3.6 平衡方程式以及 M/M/1 队列的推广, pages 108–110. 上海翻译出版公司, 1987.