arXiv:2301.05770v1 [cs.DC] 13 Jan 2023

PESC – 顺序代码并行实验

预印本

Henrique C. T. Santos*
伯南布哥联邦学院 累西腓，巴西
henrique.santos@recife.ifpe.edu.br

Luciano S. de Souza†
统计与信息学系 伯南布哥联邦乡村大学 累西腓，巴西
luciano.serafim@ufrpe.br

Jonathan H. A. de Carvalho
信息学中心 伯南布哥联邦大学 累西腓，巴西
jhac@cin.ufpe.br

Tiago A. E. Ferreira
统计与信息学系 伯南布哥联邦乡村大学 累西腓，巴西
tiago.espinola@ufrpe.br

2023年1月18日

摘要

随着计算算法在科学界各领域日益普及，对计算资源的需求也随之增长。这种需求推动了多个云平台的发展，它们抽象化了计算基础设施的复杂性。遗憾的是，使用这些资源的成本可能会使一些研究无法开展，而这些研究原本只需较简单的基础设施即可完成。本文介绍了一个平台，可利用网络上的可用资源，通过容器分发计算机仿真；该平台抽象化了配置这些执行环境所需的复杂流程，使任何用户都能利用这一基础设施。仿真可使用任何编程语言（如 Python、Java、C、R）开发，并可满足特定的执行需求，从而让科学界普遍受益。我们将介绍运行仿真所取得的结果，其中有些仿真需要使用不同的初始参数运行 1000 多次，以及通过使用该平台而受益的其他实验。

关键词 桌面网格 · 并行编程 · 分布式计算 · 容器虚拟化

1 引言

随着亚马逊 AWS [亚马逊、 2022],Microsoft Azure [微软、 2022],和 Google Cloud [谷歌等公共云服务的出现，桌面网格解决方案 [Choi 等人， 2007] 逐渐被这些平台上的解决方案或利用这些平台功能扩展的解决方案所取代 [Krašovec 和 Filipˇciˇc， 2019]。此次迁移的原因之一，是获取和维护本地计算基础设施以及培训信息技术（IT）团队的成本高昂 [Taylor 等人， 2020]。然而，尽管这些公共云服务提供免费方案，但其访问权限受到限制，不利于耗时较长的仿真或需要专用资源的仿真 [Colab,2022]。因此，许多机构可能无法使用这些平台提供的付费资源，例如发展中国家的公立教育机构 [Njenga 等人， 2019]。

然而，即使在发展中国家，教育与研究机构也总是拥有计算机实验室，台式计算机被用于学习活动、课堂教学和研究实验。因此，该

*95，路易斯·弗雷雷教授大道500号 - 大学城，累西腓 - 伯南布哥州
†95，曼努埃尔·德·梅代罗斯街35号 - 两兄弟区，累西腓 - 伯南布哥州

机构可以利用所有这些设备搭建本地基础设施，创建能够运行耗时仿真的计算环境。不过，在这种情况下，最大的问题是配置和支持计算环境，因此可能需要培训 IT 团队。同时，即使计算基础设施已经搭建并完成配置，用户也需要学习计算并行和高性能技术，才能使用新环境的计算能力。

受上述场景启发，我们提出了一种充分利用现有基础设施的方法：开发一个计算平台，整合所有可用的计算能力，使当前的 IT 专业人员能够将解决方案作为本地服务保留，且易于维护和配置。此外，该平台使学术界能够运行研究代码和仿真，也可将其用作教授并行编程的教学工具。顺序代码并行实验（PESC）是一个旨在开放闲置计算资源的平台。该平台支持用户配置执行环境，并在这些资源上运行代码和计算机仿真。用户与 PESC 的所有交互均通过 Web 界面进行，确保用户在执行过程中以及监控正在处理的请求时都拥有自主权。与执行请求关联的每段用户代码都会被打包到 Docker [Merkel, 2014] 容器中，再由服务器分发至运行代码的客户端计算机。此外，服务器会监控每项执行的状态，以便重新分配工作负载；如果发生故障或通信中断，还会将任务重新分配给其他计算机。由于使用了容器，使用不同编程语言创建的程序都可以在 PESC 平台上运行。

近几十年来，为此出现了各种平台、框架和工具，例如 CONDOR [Litzkow 等人，1987] 以及 BOINC [Anderson, 2004]。它们利用计算机上的闲置资源来分发分布式编程任务。二者的区别在于，CONDOR 使用私有网格中的资源，而 BOINC 则通过访问公共计算机来实现志愿计算。近期还开发了其他工具，例如 Everest [Sukhoroslov. and Afanasiev., 2014],OSCAR [Pérez 等人, 2019], DLHub [Li 等人, 2021], 以及 ARC[Prila et al., 2021]。虽然这些工具可用于本文所述问题的情境，但它们专注于解决特定问题，会干扰用户代码，和/或无法让用户自主使用平台。此外，这些工具可能会增加基础设施管理方面的需求，而规模较小的 IT 人员团队可能无力应对。

PESC 的另一项特点是对用户代码的侵入性极低，使遗留程序无需对代码进行重大修改即可受益于该平台。之所以不侵入代码，是因为将最初按顺序执行编写的程序并行化并非易事 [Bhalla, 2014]。对此，PESC 通过用于标识当前运行实例的参数简化了这一过程，使使用独立循环的顺序代码能够轻松调整为并行运行，并让循环的每次迭代都在不同的容器中运行。

我们使用配置与教育机构实验室中常见机器相似的机器，在受控且真实的场景中评估了 PESC 平台。结果表明，PESC 平台显著加快了顺序代码的执行，节省了大量时间，即使没有修改（或仅对用户代码进行少量修改）也是如此。

本文其余部分安排如下。第 2 节介绍现有的并行执行分发计算方案。接着，第 3 节概述该平台。第 4 节介绍 PESC 架构及其组件之间的通信方式。第 5 节通过受控测试评估该平台。第 6 节介绍采用该平台并从本文提出的方案中受益的实际用例。最后，第 7 节总结本文的工作。

2 相关工作

已有多项关于桌面网格的研究，其中一些在综述论文中进行了分析 [Khan 等人，2017年，Ivashko 等人，2018]。如今，这些综述论文中评述的解决方案并非都仍在运行。此外，几乎所有经过分析的解决方案都不符合本文提出的方案：有些方案要求安装特定的库才能执行 [WISCONSIN–MADISON 的用户代码, 2022], 有些则是针对特定编程语言构建的 [Almeida 等人, 2019], ; 还有些要求用户根据项目定义的要求重写代码 [Aldinucci 等人, 2021]。随着技术的发展，我们可以重新思考如何实现这些桌面网格，着重实现易于用户采用、可移植，以及运行计算实验时无需（或尽量少）侵入用户代码。

有多种可用技术可用于配置环境，将正在运行的进程负载分配到不同的联网计算机上，例如 Docker Swarm [Docker、2022] 以及 Kubernetes [谷歌，2022年b]。其中许多方案依赖专业 IT 团队来搭建这些环境，或是专为在云基础设施中提供服务而创建。例如，Web 服务器或数据库服务器，如 [Docker、2022] 和 [谷歌，2022年b]中所述。

这类问题的一种解决方案基础见于 [Litzkow 等人， 1987], 其后发展成一种更强大且更新的工具，名为 HTCONDR。尽管这种方案简单易用，但采用该方案的用户必须懂得如何使用命令行工具，并学习一种任务描述语言来提交代码。PESC 平台的设计赋予用户自主性，即使用户没有计算机科学领域的专门培训，也能通过直观的 Web 界面请求执行代码。
另一个例子是 BOINC [BOINC, 2022]。BOINC 由加利福尼亚大学伯克利分校开发，是一个允许在本地环境中进行志愿计算和网格计算的平台。在本地网格计算环境中，BOINC 还利用现有计算机的空闲资源，执行由服务器分发的工作。用户必须针对每项需求创建一个项目，并将其代码配置为 BOINC 应用程序。PESC 平台借鉴了 BOINC 在本地环境中的一些网格计算概念。不过，PESC 对平台的复杂性进行了抽象，使其更易于采用，从而让任何用户都能运行自己的代码和仿真。
为实现这一目的而提出的另一个平台是 EVEREST [Sukhoroslov. and Afanasiev., 2014], 在该平台中，可以将一组计算机注册到服务器，由平台创建者集中管理，并开始以分布式方式接收执行任务。PESC 平台完全包含在机构内部，确保敏感研究的保密性和安全性。此外，由于它基于容器来创建程序的执行环境，因此无需在运行用户代码的计算机上安装与用户代码相关的依赖项。

其他解决方案基于与云基础设施相关的技术，例如无服务器和函数即服务[Hassan 等人， 2021]。其中，值得强调的是 OSCAR [Pérez 等人， 2019], ，它提出了一个框架，可在本地环境中高效支持函数即服务，用于通用文件处理计算应用程序。PESC 平台尽量减少创建生态系统时对其他产品或框架的使用，使其更易于 IT 人员管理，并且该平台旨在运行通用科学应用程序。
其他较新的研究工作针对特定问题提出了解决方案，但鉴于其架构的特性，这些方案也可用于在其他计算机上分发任务。例如，DLHub [Li 等人， 2021]是一个学习系统，为科学机器学习提供模板发布和服务功能。该服务基础设施基于 funcX [Chard 等人， 2020], ，这是一个专门用于支持远程和分布式函数执行的函数即服务平台。PESC 平台的目标类似，但其设计方式不会规定用户代码应如何编写，因此无需修改即可运行遗留程序。

3 系统概述

本节将介绍 PESC 平台，描述其主要组件，并概述系统工作流程。

PESC 平台旨在让用户利用空闲计算资源运行计算机程序。用户无需在计算框架中进行特定设置，也无需对程序进行重大修改；在许多情况下，甚至无需修改代码。此外，用户程序可以多次运行（例如，统计模拟需要进行多次重复），并且可以并行运行，也可以不并行运行，整个过程对用户透明。用户可通过直观的 Web 界面访问 PESC，上传并运行代码。
在这里，空闲计算资源称为客户端。房间用于对 PESC 平台上可用的客户端进行分组。这些房间可以表示用户环境中客户端的物理分配方式，从而生成工作模块，如图 1 所示。客户端首次连接到平台时，仅管理员用户可见。管理员用户可以将该客户端添加到公共房间，使所有用户都能访问该客户端；也可以将其添加到访问权限更严格的房间，由用户管理哪些人可以访问该客户端。任何用户都可以创建房间；管理员将客户端分配到某个房间后，该用户便可以在这些房间之间重新分配客户端。

PESC 平台采用非侵入式解决方案，以保持与现有计算机程序的兼容性。用户无需对代码进行重大修改（在许多情况下甚至无需修改）即可运行代码。用户代码将以命令行调用的方式运行，并附带与当前执行相关的参数。因此，用户代码必须包含头文件，才能使用执行时传入的参数值；不过，也可以忽略该头文件，在不知道这些参数值的情况下运行代码。PESC 平台会在其网站上提供头文件代码，从而简化用户的配置流程。值得强调的是，Python、Java、C/C++ 和 R 项目等多种编程语言均已有相应的头文件。如果用户代码中没有头文件，PESC 也可以自动添加。

该头文件定义了默认值，不会干扰在 PESC 平台之外执行代码。该头文件定义的参数列表如下：

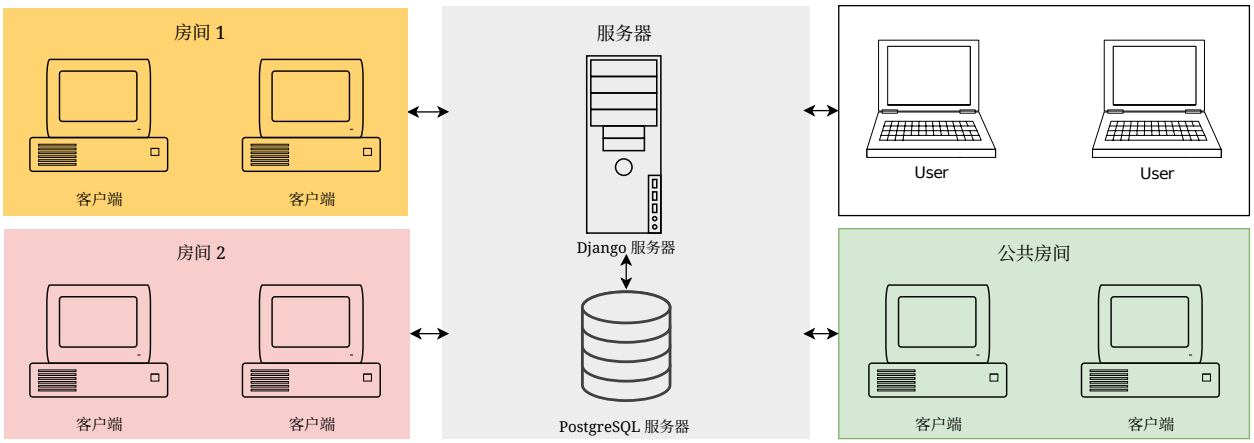


图 1：PESC 系统概述

- app_dir：客户端执行代码的当前目录；
- checkpoint_dir：用于存储崩溃恢复点的目录；
- output_dir：执行代码后将返回给服务器的目录。用户必须将代码所需的所有输出存储在此目录中；

- 秩：用户代码的每个实例都会获得一个身份值，该值从零开始；
- 重复次数：用户请求运行代码的重复次数；
- master_addr：运行当前用户代码中秩为 `= 0` 的实例的客户端 IP 地址；

- master_port：与运行当前用户代码中秩为 `= 0` 的实例的客户端 IP 地址关联的端口；

- 参数：执行请求表单中由用户定义、以向量形式传递给用户代码的值列表。

PESC 平台在 Docker 容器中执行用户代码。要配置容器执行环境，用户必须创建 Dockerfile 和 requirements.txt 文件。Dockerfile 定义容器环境，包括操作系统依赖项；requirements.txt 则描述用于定制 Python 环境的库列表。Python 是 PESC 平台的基础语言。用户可以自行提供这些文件来构建容器镜像。为了简化平台，用户也可以使用商店中提供的一组常用技术。如果商店中的选项无法满足用户需求，用户可以向平台提出请求。系统管理员或平台上的其他用户都可以响应此请求。不过，如果是普通用户响应请求，那么该内容只有经过系统授权用户（如版主）验证后，才会提供给提出请求的用户。在 PESC 平台中，所有这些执行环境配置统称为领域。

定义领域后，用户必须配置 PESC 平台将要执行的进程。该进程由将在容器中运行的用户代码文件组成，可以是使用 Python 等指定编程语言编写的单个文件；如果代码由多个文件组成，则可以提供一个 ZIP 文件。最后，服务器会将容器定义分发给客户端，由客户端控制 Docker 映像的整个构建流程，以及与用户代码关联的容器生命周期执行。

PESC 平台还允许用户上传文件并将其作为共享文件，供客户端在用户进程执行期间请求使用。这些文件会作为共享资源，供该客户端上运行的同一用户进程的所有实例使用。这样便无需向同一进程的每个实例重复传输相同文件。此外，所有这些共享文件都受到只读限制，因此不会影响容器运行，从而避免竞态条件。此类资源的一个使用示例是 CSV 文件格式的数据库，该数据库会在用户进程执行期间加载。例如，用户请求在一个有两个或更多客户端的房间中执行指定次数的代码。在这种情况下，每个客户端只会收到一份数据库文件副本，供每个进程实例在执行时使用。

完成领域和进程配置并上传共享文件（如有必要）后，用户必须提交执行请求，PESC 平台将其称为请求。平台会将该请求加入用户的执行队列，并在客户端有可用资源时分发该请求。创建请求时，用户必须提供数据，

- 领域：执行环境的定义；

- 进程：要执行的用户代码；
- 重复次数：平台将执行该进程的次数。客户端将在不同的容器中执行每次任务；
- 并行：代码是否使用并行资源。此值将告知客户端，在开始执行前等待所有请求副本分发完毕；
- 参数：以逗号分隔的值列表，进程实例将在运行时将这些值作为参数接收；
- 图形处理器：该进程是否需要图形处理器。此值表示应仅选择具备此特征的客户端；
- 同一台机器：用户必须说明所有进程实例是否必须在同一客户端上运行。用户代码应利用内存或图形处理器等本地资源共享功能；
- 共享文件：运行进程所需的文件。用户必须事先上传这些文件；
- 房间：应接收待执行进程的客户端组。

图 2 展示了如何在 PESC 平台的 Web 界面中提供这些数据。

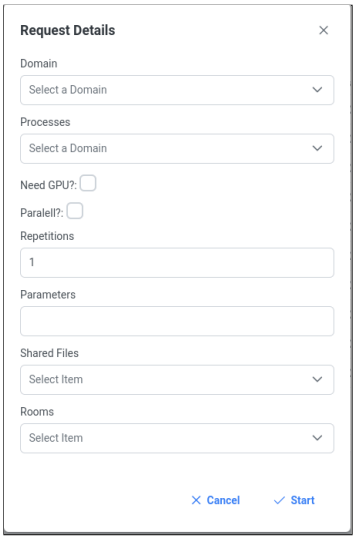


图 2：请求表单。

用户可以通过 Web 界面中的客户端状态面板实时监控请求的执行情况，如图 3 所示。监控器仅显示执行开始和结束的消息。不过，如果进程与 PESC 平台的资源集成，监控器还可以发送执行详情，例如自定义消息和执行百分比。用户必须实现一个包含特定定义的 Python 类，才能将代码与平台资源集成。平台的 Web 界面中提供了该类的模板。不过，值得注意的是，与平台集成并非运行用户计算实验的必要条件。

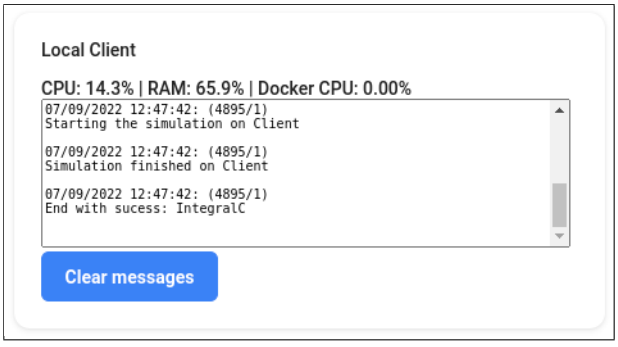


图 3：客户端状态面板

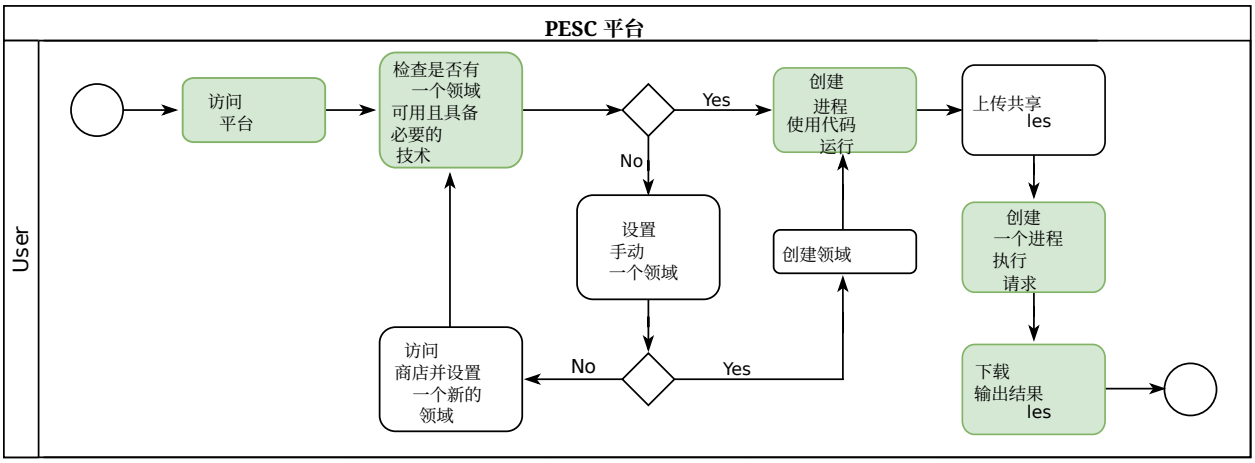


图4：PESC平台流程。绿色元素表示必须与用户交互的步骤。

按照 PESC 的序列流程，客户端会接收请求中定义的用户进程实例，并检查是否已为该进程构建 Docker 镜像。如果尚未构建，则开始构建镜像。在容器执行之前，客户端会检查请求中配置的共享文件是否已接收；如果尚未接收，则向服务器请求这些文件。之后，正式开始计算实验。

在请求执行期间，客户端会定期向服务器查询用户是否取消了请求。如果用户取消请求，客户端就会中断与该请求关联的所有进程实例的执行。执行完成后（即用户未取消请求），客户端会压缩输出目录并将其发送回服务器。该目录中会创建一个 output.txt 文件，其中包含程序在屏幕上输出的所有内容。例如，对于 Python 程序，这些内容包括 print 或 printf 命令的执行结果。
请求执行完成后，用户可以开始下载参与运行该请求的各个客户端所创建的所有输出文件。最后，系统会将这些文件压缩成一个文件，把 output.txt 的内容按各进程实例的秩排序后汇总到一个新文件中，再将该文件发送给用户，以响应其下载请求。图4总结了这一流程。

4 架构

本节介绍 PESC 平台的架构，并说明其模块及模块间的通信方式。
如图1所示，PESC 是一个由三个模块组成的系统，这些模块托管在本地基础设施中，分别是客户端、管理器和前端模块。前端模块采用 VueJS JavaScript 框架开发 [Street 等人， 2018]并通过 REST API 调用与管理器通信，但本文不讨论前端模块。尽管 PESC 平台采用开源技术开发，其代码在当前测试阶段尚未开放。不过，此后该平台将作为开源工具向社区开放。客户端模块和管理器模块各由一组组件构成，如图5和图6所示，具体如下。

4.1 管理器模块架构

管理器模块（MM）是一个使用 Django Python 框架开发的 Web 服务器 [Django 软件基金会， 2021] 它为所提出的系统开发提供了适用的一组资源。这些功能包括与数据库集成（对象关系映射）、集成管理模块、用户身份验证，以及对系统功能的访问控制。此外，通过与 Django 集成，Django REST 框架 [基金会，2021] 模块已配置为通过 REST API 调用与客户端和前端模块建立通信端点。

平台管理的数据存储在 PostgreSQL [Group、 2019] 数据库中。不过，Django 对象关系映射抽象使得在不更改系统代码的情况下使用其他关系型数据库成为可能。此外，系统还开发了一组监控器，用于管理用户请求和客户端通信。每个监控器都运行在不同的线程上，并持续更新应用程序状态。

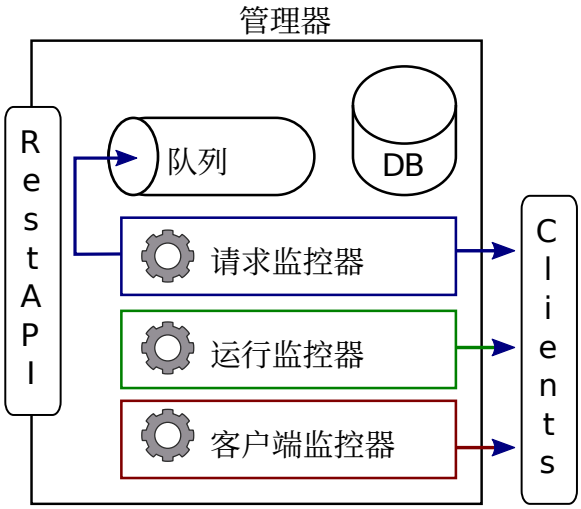


图5：服务器架构

4.1.1 客户端监控器

客户端监控器的主要功能是验证已连接的客户端是否可用于接收并运行用户进程。尽管客户端已经连接，但由于可能存在网络不一致或某个客户端意外关机的情况，仍有必要执行此检查。例如，在这种情况下，如果可以通过网络访问客户端，但它没有响应 REST API 调用，客户端监控器将尝试重启此机器上的客户端模块。每个客户端都可以通过其配置文件配置这一启动选项。

4.1.2 请求监控器

每个已连接的用户都有一个用于存储其请求的队列。请求监控器负责管理这些队列，并分析每个请求。请求可以设置为单次执行，也可以设置为重复执行多次。在这种情况下，请求会继续留在用户队列中，直到将全部进程实例转发给可用客户端。客户端的选择取决于请求的特征，例如是否需要图形处理器，以及已分配给每个客户端的工作负载。

4.1.3 进程运行监控器

发送给客户端的每个请求进程实例都称为一次进程运行。进程运行监控器会定期向接收进程实例并执行它的客户端查询其执行状态。无法评估的进程运行必须转移到另一个客户端，并向当前客户端发送取消通知。离线客户端会在下次与 MM 建立连接时收到取消通知，并暂停执行。因此，只要客户端在平台上可用，进程实例就会根据其配置文件中设定的限制分配给该客户端，并且请求应得到充分满足。

4.2 客户端模块架构

客户端模块（CM）是一个使用 Flask 框架开发的应用程序 [Grinberg，2018]，可通过 REST API 调用创建与 MM 和 Docker 容器通信的端点。与 MM 交换的数据存储在 SQLite3 数据库 [Hipp，2020] 中。不过，系统使用 SQLAlchemy 软件包 [Bayer，2021]，它提供了一个抽象层，使系统无需修改代码即可使用其他关系型数据库。此外，系统还开发了一组监控器，用于管理镜像构建操作和容器执行。每个监控器都在不同的线程上运行，并持续更新应用程序状态。

CM 作为后台服务运行，由为此目的创建的操作系统用户启动。因此，CM 必须在使用计算机资源时，兼顾可能同时使用该操作系统的其他用户。之所以需要保持这种平衡，是因为客户端并非专供 PESC 平台使用。

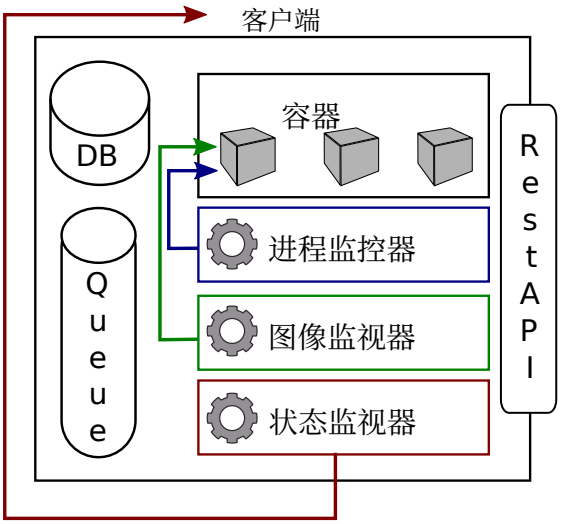


图 6：CM 架构

4.2.1 状态监视器

每个客户端都必须向 MM 告知其运行特性。状态监视器负责收集这些信息，以确认客户端处于正常运行状态，且能够执行收到的进程。如果 MM 未收到这些信息，就可以将正在运行的进程转移到其他可用客户端。此外，状态监视器还会验证以下信息：

- 操作系统的其他用户同时使用该机器；
- 内存使用百分比；
- 处理器使用百分比；
- 图形处理器内存使用率（如有）。

用户连接到客户端时，操作系统分配给 PESC CM 的资源会减少到 10%，以免影响用户使用机器的体验。不过，当处理器使用率达到 70% 时，客户端会通知 MM，表示无法再获取新的进程来运行。这些百分比在客户端配置文件中定义。

4.2.2 图像监视器

Docker 容器由映像构建而成，必须在进程实例开始运行之前预先构建。图像监视器负责监控映像构建需求，并向整个系统通知此进程的进度。如果客户端上缺少映像，便会向 MM 发送包含必要信息的请求。任何依赖某个映像构建的进程实例，都必须等待映像成功构建完成后才能开始执行。

4.2.3 进程监控器

用户创建的进程运行在 Docker 容器中，而这些容器需要文件和目录构成的基础设施才能正常运行。进程监控器负责监控进程实例的生命周期，并为其运行准备必要的基础设施。容器以分离模式运行，因此可以同时启动其他容器，同时遵守客户端配置文件中定义的操作系统资源消耗限制。进程实例通过容器中的一个包装器运行，该包装器通过 REST API 端点与客户端通信。

用户进程可以使用检查点目录，该目录的信息包含在接收参数的头文件中。如果容器执行过程中发生故障，系统将检查是否存在恢复点。如果文件中存在恢复点，进程监控器将从该点重新开始执行。

5 评估

评估 PESC 平台对于确保其正确运行并衡量采用该平台后运行时间的收益至关重要。

5.1 评估环境

为契合 PESC 平台的主要目标，我们创建了一个评估环境，用于模拟一个配备一台服务器和六台客户端计算机的小型实验室。表 1 列出了评估环境的软件要求，其中所有计算机均使用 Ubuntu 18.04.6 LTS（仿生海狸）作为操作系统，并配备 500 GB 硬盘。表 2 总结了其他计算机配置。

表 1：软件要求。

机器类型	**数据库**	**主要软件包**
服务器（管理器）	PostgreSQL 12	Django、DRF
客户端	SQLite 3	Flask、SQLAlchemy

表 2：客户端机器配置。

机器	RAM	系统处理器
客户端 1 和 2	32GB	Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
客户端 3	8GB	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
客户端 4 和 5	32GB	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
客户端 6	16GB	Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz

5.2 评估执行

为了评估 PESC 平台，我们创建了多个场景。在每个场景中，平台都必须执行用户创建的请求，并得出预期结果。下面将更详细地介绍每个场景：

- 运行一段简单代码；
- 重复运行这段简单代码 N 次；
- 运行专用代码；
- 运行经过并行化的专用代码；
- 在客户端模块（CM）或管理器模块（MM）发生故障后恢复；
- 运行使用具有原生并行功能的框架的代码。

5.2.1 场景 1——运行简单代码

在此场景中，用户必须运行一段使用 Python 编写、仅使用内置函数的简单代码。该代码是一个生成 10.000.000 个数字的高斯随机数生成器。用户将使用商店中提供的领域（Simple Python），该领域会抽象出所需的全部容器基础设施知识，用户无需对其代码进行任何更改。此场景是 PESC 平台最基本的用法，算法 1 展示了用户伪代码的一部分。

算法 1场景 1——验证代码

$x \leftarrow 0$
对于 $rank = 0$到 10000000 **执行**
  $u1, \quad x \leftarrow get_uniform(x)$
  $u2, \quad x \leftarrow get_uniform(x)$
  $z1, z2 \leftarrow get_gaussian(u1, u2)$
  print 秩+“: ”+z1 +“, ”+z2
结束循环

此场景要求将用户代码分配给一个新的进程（验证1_1）。为了运行这一新进程，用户需要根据以下参数的值创建一个新的请求：

- 领域：Simple Python
- 进程：验证1 1_
- 重复次数：1
- 房间：公共

PESC 平台会选择一个客户端来分发并运行此进程。该进程在此客户端上按预期运行，并将输出以压缩文件的形式返回给用户。我们还验证了此场景下 R 脚本和 Java 代码版本的执行情况。在这些情况下，用户必须为每种技术创建领域，或在商店中选择相应领域。此外，要执行用 Java 编写的代码，用户必须上传可执行 JAR 格式的进程文件。两次仿真均已运行，并返回了预期的输出文件。

5.2.2 场景 2 - 将简单代码重复执行 N 次

假设用户需要生成更多随机数。在这种情况下，无需更改代码，只要将重复次数参数设为大于 1 的值即可，如下所示。

- 领域：Simple Python
- 进程：验证1 1_
- 重复次数：10
- 房间：公共

PESC 会将此进程的每个实例分发给可用客户端，并在请求中附加一个包含所有生成输出的压缩文件。此外，还会为每次进程运行单独附加一个压缩文件。

5.2.3 场景 3 - 运行专用代码

在此场景中，用户需要运行一段使用 Scikit-learn [Pedregosa 等人， 2011] 库功能编写的 Python 仿真，但无法更改机器的操作系统以安装所需软件。在这种情况下，使用 PESC 平台相当于使用容器。为了对转换为 CSV 格式的 MNIST 手写数字数据集进行分类，用户编写了一个 k近邻（kNN）算法 [Cunningham 和 Delany， 2021], 而算法 2 展示了部分用户伪代码。

算法 2场景 2 - 验证代码

$train \leftarrow read_csv('./mnist_test.csv')$
$test \leftarrow read_csv('./mnist_train.csv')$
$rank = 0$到 10 **为止**
  $model \leftarrow knn(n = rank)$
  $model.fit(train.x,\ \ train.y)$
  $accuracy \leftarrow model.score(test.x,\ \ test.y)$
  print "k="+秩+"==>"+准确率
结束循环

此场景所需的文件包括 Dockerfile、requirements.txt、用户代码以及包含数字基数的 CSV文件。用户使用前两个文件创建领域（场景_3），或者使用商店中的 Python环境，并设置必要的软件包以进行定制，然后使用用户代码创建进程（验证_3）。我们在此次执行之前，已在一个进程中创建了测试数据库和训练数据库；它们将作为共享文件上传至 PESC 平台，并可供代码从相同的目录路径访问。这样，本节介绍的其他验证场景就可以使用这些共享文件，无需为已参与此场景执行的客户端再次传输文件。最后，用户使用以下参数创建请求：

- 领域：场景 3_
- 进程：验证 3_
- 重复次数：1

- 共享文件：minst_test、mnist_train
- 房间：公共

代码按预期运行，并返回输出文件，其中显示了清单 1 中的结果以及每个 k 值对应的准确率：

```
k=1==>0.9416833333333333 k=2==>0.
93195 k=3==>0.9428333333333333 k
=4==>0.94115 k=5==>0.
9425166666666667 k=6==>0.
9404333333333333 k=7==>0.
9401333333333334 k=8==>0.93905 k
=9==>0.9380166666666667 k=10==>0.
9371666666666667
```

清单 1：场景 3 - 输出

5.2.4 场景 4 - 并行化专用代码

如第 3 节所示，PESC 平台通过命令行调用传递一组参数来执行用户代码。其中一个参数是运行实例的 ID，称为秩，其值从 0 开始。用户可以利用该参数将顺序循环转换为一组并行执行。在这种情况下，我们不再使用包含 N 次重复的单个实例和循环，而是可以使用 N 个程序实例，每个实例只执行一次循环内容，如图 7 所示。这样，每个实例都可通过秩变量表示循环的一次迭代。

在此场景中，用户修改了前一场景中使用的代码，使 k 的每个值都在不同的进程实例中执行。这样，只需对用户代码进行最少的修改，便可并行执行顺序代码，如算法 3 所示。在这种情况下，用户对平台的利用比前一场景更加高效。

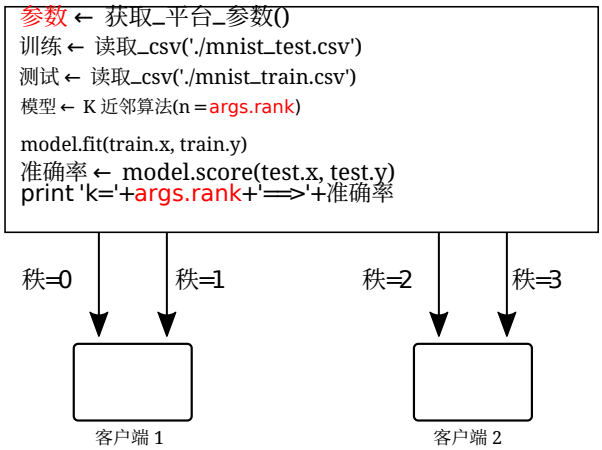


图 7：秩分布

算法 3场景 4 - 验证代码

$args \leftarrow get_platform_parameters()$
$train \leftarrow read_csv('./mnist_test.csv')$
$test \leftarrow read_csv('./mnist_train.csv')$
$model \leftarrow knn(n = args.rank)$ ▷ 循环从这里开始
$model.fit(train.x, train.y)$
$accuracy \leftarrow model.score(test.x, test.y)$**print** "k="
+秩+"==>"+准确率

用户使用以下参数的值创建一个新请求：

- 领域：场景 3_
- 进程：验证 4_
- 重复次数：10
- 共享文件：minst_test, mnist_train
- 房间：公共

此场景所需的要求与上一个场景相同。代码按预期运行，并返回每个客户端的输出文件，其中包含 k 值对应的结果。输出文件会合并为一个文件，以便用户分析结果。

使用 $k = 1$、5、10、15 和 20，反复执行验证_3和验证_4 进程，以验证使用 PESC 平台所带来的性能提升。执行时间进行了比较，如表 3 和表 4 所示。我们没有考虑客户端构建 Docker 镜像所用的时间。图 8 展示了针对该场景所做代码更改带来的性能提升。

表3：场景 3 执行时间

K	开始时间	结束时间	秒
1	13:48:42	13:48:59	17
5	13:50:25	13:51:44	79
10	13:56:19	13:59:00	161
15	14:08:03	14:12:06	243
20	14:13:57	14:19:22	325

表4：场景 4 执行时间

K	开始时间	结束时间	秒
1	14:32:02	14:32:20	18
5	14:33:55	14:34:56	61
10	14:39:23	14:40:47	84
15	14:48:04	14:49:35	91
20	14:53:26	14:54:59	93

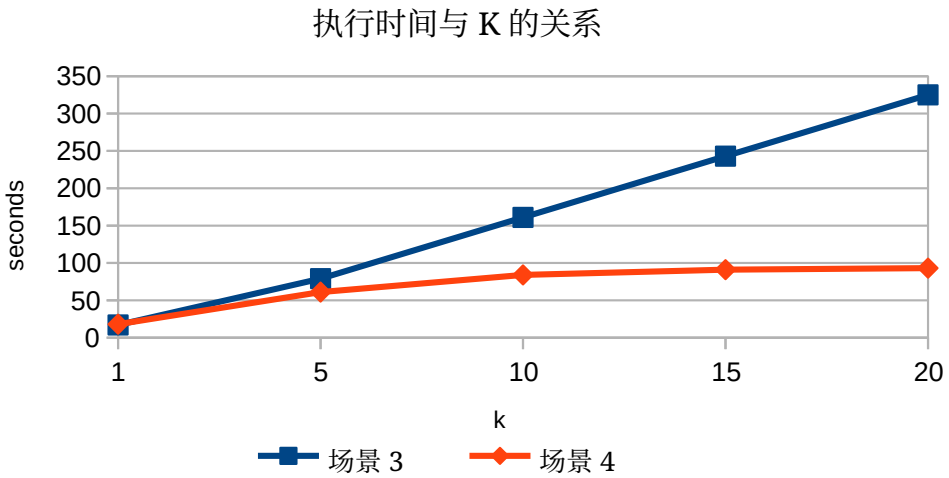


图8：顺序场景与并行场景的比较

5.2.5 场景 5 - 客户端模块（CM）或管理器模块（MM）发生故障后恢复

在此场景中，我们将重复上一个场景中创建的相同请求，但在收到待执行的进程实例后，将两个客户端与 MM 断开连接。如果其中一个客户端发生通信故障，MM 将取消发送进程，并将其重新分配给另一个客户端。代码运行符合预期，我们可以在清单 2 所示的数据库表中查看重新分配的进程。用户可以通过 Web 界面验证此行为。

id	秩	客户端_id	状态	obs
23547	0	11	3	成功
23548	1	12	3	成功
23549	2	8	3	成功
23550	3	10	5	已取消
23557	3	12	3	成功
23558	4	8	3	成功
23551	4	7	5	已取消
23552	5	12	3	成功
23553	6	8	3	成功
23554	7	7	5	已取消
23559	7	11	3	成功
23555	8	11	3	成功
23556	9	11	3	成功

清单 2：场景 5 - 数据库查询

如我们所见，使用 $id = 23550$ 和 $rank = 3$ 执行的进程在具有 $id = 10$的客户端上被取消了，但具有 $id = 12$ 的客户端完成了相同的秩值。进程 ID 为 23558 和 23554 的进程也发生了同样的情况。这种重新分配保证，只要客户端保持连接且资源可用，PESC 平台就能成功执行用户代码并完成请求，无需任何用户迭代。这种行为也是 PESC 平台的主要目标之一。
如果 MM 发生故障，客户端会继续执行收到的进程实例。MM 恢复后，它们会发送执行状态，并在彼此之间建立新的连接。进程实例分配完成后，这一过程不会改变最初的执行流程。

5.2.6 场景 6 - 运行使用原生并行功能框架的代码

一些框架为并行程序提供了原生功能，例如 Pytorch 分布式 RPC 框架（DRF）[PYTORCH, 2022]。在此场景中，我们会执行一个进程，其中每个实例都将根据该资源的要求承担不同的角色。此场景基于 PyTorch 官方教程中的代码构建 [Li, 2002], ；该教程使用 torch.distributed.rpc 包进行分布式训练。这里仅使用了通过 RPC 和 RRef 实现分布式强化学习的代码。在此场景中，一个客户端将充当代理，负责协调向其他客户端发送数据；其他客户端则充当观察者。代理起着核心作用，观察者需要识别该客户端，才能建立彼此之间的通信。PESC 平台会告知为每个进程实例创建的客户端的 IP 地址和端口 $rank = 0$ for each process instance。这样，其他实例$rank > 0$便能与该实例通信，并满足 Pytorch DRF 所用资源的要求。用户代码会检查秩参数，并进行必要的调整，如算法 4 所示。最后，用户使用以下参数创建请求：

- 领域：场景 6_
- 进程：验证 6_
- 并行：是
- 重复次数：3
- 房间：公共

算法 4场景 6 - 验证代码

```
args ← get_platform_parameters()
...
如果 args.rank == 0则                                              ▷ 秩=0 是代理
    name ← agent_name
    rpc.init_rpc(name, args.rank, args.world_size)
    agent ← Agent(args.world_size, gamma)
    重复
        agent.run_episode()
        last_reward ← agent.finish_episode()
    直到 agt.running_reward > agt.reward_threshold
    print "已解决！"
else                                                               ▷ 其他秩是观察者
    name ← observer_name + args.rank
    rpc.init_rpc(name, args.rank, args.world_size)
    观察者被动等待代理的指令
结束条件判断
阻塞直到所有远程过程调用完成，然后关闭远程过程调用实例
rpc.shutdown()
```

代码按预期运行，请求表单中的重复次数参数决定客户端将执行的实例数量。并行参数可防止进程未同步便启动。这一信息会使所有收到进程实例的客户端在开始执行前等待 MM 发出的信号。此选项有助于减少分配给未运行其进程的容器的资源。

6 实验结果——真实案例

尽管验证测试得出了预期结果，我们仍采用一个用例来评估该平台对真实用户的价值。此用例涉及对量子搜索过程进行仿真：该过程使用懒散量子游走算法，在一个 n维超立方体中搜索多个解[Souza 等人， 2021]。

仿真量子系统的计算成本往往呈指数级增长，并且需要大量内存和计算机运行时间 [Acheson， 2001]。因此，经典系统可能无法高效地仿真量子系统 [费曼， 1982]。

这种量子游走算法的特点是在用于表示搜索空间的结构的每个顶点上设置自环，本例中的结构是超立方体。仿真分为三个场景，具体取决于被标记为解的顶点类型：第一个场景中的非相邻顶点、第二个场景中的相邻顶点，以及第三个场景中的相邻和非相邻顶点。

懒散量子游走的另一个特点是其对自环权重值的依赖 [Carvalho 等，2020，Souza 等，2021，de Carvalho 等， 2022]。该权重值会调整游走者停留在顶点上的概率。在三个场景中，分别使用四种自环权重值进行了实验。为了根据非相邻标记顶点的相对位置，获得量子游走的平均行为，需要进行仿真。因此，在包含非相邻标记顶点的场景中，每组标记顶点都进行了100次仿真。

因此，对于标记顶点仅彼此不相邻的场景，执行了 1200 次仿真。对于标记顶点既有相邻也有不相邻的场景，执行了 1100 次仿真。对于标记顶点彼此相邻的场景，仅执行了 12 次仿真。每次仿真都在一个秩上执行。每个秩负责执行 30 次懒散量子游走，迭代次数从 1 到 200 不等。进程结束时，每个秩都会返回 30 次游走中的最大成功概率。要运行标记顶点既有相邻也有不相邻、自环权重为 3 的场景，用户会创建一个新请求，并将以下参数设置为相应的值：

- 领域：量子游走 • 进程：相邻-不相邻 • 并行：否
- 1200 重复次数:参数: 3
- 房间：公共

在这种情况下，没有使用共享文件，而是将顶点文件和用户代码打包成 ZIP 文件来创建进程。借助平台，仿真执行流程得以简化，因为平台会管理进程的状态和生命周期。共有四个注册客户端参与了此次执行。因此，可以在用于验证测试的同一实验室环境中运行仿真。

客户端节点并非 PESC 平台专用。它们可能会与其他系统竞争计算资源，而且客户端设置也各不相同，如表 5 所示。因此，实例分配并不均衡，这一点可从表 5 中的平均时长列看出。

表 5：按客户端划分的实例分配。

机器	平均时长	数量
客户端 1	01:20:38.42278	207
客户端 2	01:22:21.366742	202
客户端 3	01:01:13.74276	224
客户端 6	00:31:24.892872	567

考虑到实例执行的最低平均时间为 00:31:24s，如果在该客户端上以原始代码格式执行全部 1200 次顺序重复，执行时间将约为 600 小时。因此，考虑到第一个实例开始执行到最后一个实例执行结束的时间，PESC 平台上的总执行时间约为 12:39:14s。

7 结论

将计算基础设施用作计算网格，以分配并行执行的负载，已证明是开展科学研究的可行选择。我们提出了一个通用平台，重点关注参与进程的各方——用户和 IT 团队，也就是说，平台易于使用、维护和配置。对创建该平台所需技术的抽象，使任何用户使用任何编程语言开发的程序几乎无需任何修改即可运行。它还使最初为顺序执行设计的代码能够在用户只需进行最少调整的情况下并行运行。此外，利用机构的闲置资源，可以让教学与研究机构更充分地利用这些资源，而无需为此购置新设备。我们通过多个场景表明，即使在依赖语言或框架特定功能的高级用例中，该平台仍然简单易用。我们还表明，一项在顺序版本中大量使用计算资源的仿真，可以轻松改编为并行版本；即使使用日常运行中的计算资源，也能显著提升性能。

参考文献

亚马逊。主页。https://aws.amazon.com，2022年4月。微软。主页。https://azure.microsoft.com/，2022年4月。谷歌。主页。https://cloud.google.com/，2022年4月a。SungJin Choi、HongSoo Kim、EunJoung Byun、MaengSoon Baik、SungSuk Kim、ChanYeol Park 和 ChongSun Hwang。桌面网格的特征描述与分类。载于第七届 *IEEE* 集群计算与网格国际研讨会（*CCGrid'07*），第743–748页。IEEE，2007年。Barbara Krašovec 和 Andrej Filipˇciˇc。通过云计算增强网格。《网格计算杂志》，17(1)：119–135，2019年。Simon J. E. Taylor、Anastasia Anagnostou、Nura Tijjani Abubakar、Tamas Kiss、James DesLauriers、Gabor Terstyanszky、Peter Kacsuk、Jozsef Kovacs、Shane Kite、Gary Pattison 和 James Petry。仿真创新：基于云的仿真实验经验。载于*2020* 年冬季仿真会议（*WSC*），第3164–3175页，2020年。doi:10.1109/WSC48552.2020.9383971。Google Colab。Google Colab 常见问题。https://research.google.com/colaboratory/faq.html，2022年4月。Kenndy Njenga、Lalit Garg、Amit Kumar Bhardwaj、Vijay Prakash 和 Seema Bawa。肯尼亚高等教育机构采用云计算：阻碍因素及未来发展建议。*Telematics*

andInformatics，38：225–246，2019年。ISSN 0736-5853。 doi:https://doi.org/10.1016/j.tele.2018.10.007。URL https: // www.sciencedirect.com/science/article/pii/S0736585318306397。Dirk Merkel。Docker：用于保持开发和部署一致性的轻量级 Linux 容器。*Linux journal*，2014年（239）：2，2014年。Michel J Litzkow、Miron Livny 和 Matt W Mutka。Condor——空闲工作站的猎手。技术报告，威斯康星大学麦迪逊分校计算机科学系，1987年。David P Anderson。BOINC：一种面向公共资源计算与存储的系统。载于第五届 *IEEE/ACM* 网格计算国际研讨会，第4–10页。IEEE，2004年。Oleg Sukhoroslov 和 Alexander Afanasiev。Everest：面向计算型 Web 服务的云平台。载于第四届云计算与服务科学国际会议论文集 - *CLOSER*，，第411–416页。INSTICC、SciTePress，2014年。ISBN 978-989-758-019-2。 doi:10.5220/0004941404110416。Alfonso Pérez、Sebastián Risco、Diana María Naranjo、Miguel Caballer 和 Germán Moltó。用于事件驱动数据处理应用的本地无服务器计算。载于*2019*年 *IEEE* 第*12*届云计算国际会议（*CLOUD*），第414–421页。IEEE，2019年。Zhuozhao Li、Ryan Chard、Logan Ward、Kyle Chard、Tyler J Skluzacek、Yadu Babuji、Anna Woodard、Steven Tuecke、Ben Blaiszik、Michael J Franklin 等。DLHub：简化科学领域机器学习模型的发布、发现与使用。并行与分布式计算杂志，147：64–76，2021年。Olga Prila、Volodymyr Kazymyr、Volodymyr Bazylevych 和 Oleksandr Sysa。使用 Docker 平台的虚拟环境开发基于 ARC nordugrid 的网格计算组织系统。东欧企业技术杂志，6(2)：114，2021年。Ankita Bhalla。顺序程序并行化的多种方法。国际工程研究与技术杂志，3(1)，2014年。ISSN 2278-0181。URL https://www.ijert.org/ various-ways-of-parallelization-of-sequential-programs。Muhammad Khalid Khan、Tariq Mahmood 和 Syed Irfan Hyder。桌面网格系统中的调度：策略与框架的理论评估。国际高级计算机科学与应用杂志，8(1)，2017年。doi:10.14569/IJACSA.2017.080117。URL http://dx.doi.org/10.14569/IJACSA.2017.080117。Evgeny Ivashko、Ilya Chernov 和 Natalia Nikitina。桌面网格调度综述。*IEEE* 并行与分布式系统汇刊，29(12)：2882–2895，2018年。威斯康星大学 WISCONSIN–MADISON 分校。文档。https://research.cs.wisc.edu/htcondor/，2022年5月。André Luís Barroso Almeida、Leonardo de Souza Cimino、José Estevão Eugênio de Resende、Lucas Henrique Moreira Silva、Samuel Queiroz Souza Rocha、Guilherme Aparecido Gregorio、Gustavo Silva Paiva、Saul Delabrida、Haroldo Gambini Santos、Marco Antonio Moreira de Carvalho 等。一种通用型分布式计算 Java 中间件。并发与计算：实践与经验，31(7)：e4967，2019年。Marco Aldinucci、Valentina Cesare、Iacopo Colonnelli、Alberto Riccardo Martinelli、Gianluca Mittone、Barbara Cantalupo、Carlo Cavazzoni 和 Maurizio Drocco。使用 OpenMP、OpenACC 和 MPI 对科学应用程序进行实用并行化。并行与分布式计算杂志，157：13–29，2021年。Docker。服务的工作方式。https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/，2022年4月。谷歌。什么是 Kubernetes？ https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/，2022年4月b。BOINC。用户手册。https://boinc.berkeley.edu/wiki/User_manual，2022年4月。Hassan B Hassan、Saman A Barakat 和 Qusay I Sarhan。无服务器计算综述。云计算杂志，10(1)：1–29，2021年。Ryan Chard、Yadu Babuji、Zhuozhao Li、Tyler Skluzacek、Anna Woodard、Ben Blaiszik、Ian Foster 和 Kyle Chard。FuncX：面向科学的联邦式函数服务框架。载于第*29*届高性能并行与分布式计算国际研讨会论文集，HPDC '20，第65–76页，美国纽约州纽约市，2020年。计算机协会。ISBN 9781450370523。doi:10.1145/3369583.3392683。URL https: // doi.org/10.1145/3369583.3392683。Mike Street、Andrea Passaglia 和 Paul Halliday。*Vue.js 2* 完全指南：使用 *Vue.js 2* 构建端到端 *Web* 开发解决方案的实用指南。Packt Publishing Ltd，2018年。Django 软件基金会。Django，2021年。URL https://djangoproject.com。

Django 软件基金会。Django REST 框架，2021年。网址：https://www.django-rest-framework.org/。PostgreSQL 全球开发组。PostgreSQL 12.13 文档，2019年。网址：https://www.postgresql.org/files/documentation/pdf/12/postgresql-12-A4.pdf。M. Grinberg。*Flask Web* 开发：使用 *Python* 开发 *Web* 应用。O'Reilly Media, Inc，2018年。Richard D Hipp。SQLite，2020年。网址：https://www.sqlite.org/index.html。Michael Bayer。SQLAlchemy，2021年。网址：https://www.sqlalchemy.org/。Fabian Pedregosa、Gaël Varoquaux、Alexandre Gramfort、Vincent Michel、Bertrand Thirion、Olivier Grisel、Mathieu Blondel、Peter Prettenhofer、Ron Weiss、Vincent Dubourg 等。Scikit-learn：Python 机器学习。《机器学习研究杂志》，12：2825–2830，2011年。Pádraig Cunningham 和 Sarah Jane Delany。K近邻分类器——教程。*ACM* 计算调查，54(6)，2021年7月。ISSN 0360-0300。doi:10.1145/3459665。网址：https://doi.org/10.1145/3459665。PyTorch。文档。https://pytorch.org/docs/stable/rpc.html，2022年5月。Shen Li。分布式 RPC 框架入门。可从以下网址获取：https://pytorch.org/tutorials/ intermediate/rpc_tutorial.html，2002年。Luciano S Souza、Jonathan H A Carvalho 和 Tiago A E Ferreira。超立方体中的懒散量子游走：搜索多个标记顶点。载于巴西智能系统会议，第249–263页。Springer，2021年。William Acheson。费曼与计算：探索计算机的极限。《数学与计算机教育》，35(2)：167，2001年。Richard P Feynman。用计算机模拟物理。《国际理论物理学杂志》，21(6)：467–488，1982年。Jonathan H A Carvalho、Luciano S Souza、Fernando M Paula Neto 和 Tiago A E Ferreira。多个解对懒散量子漫步搜索算法的影响。载于巴西智能系统会议，第122–135页。Springer，2020年。Jonathan HA de Carvalho、Luciano S de Souza、Fernando M de Paula Neto 和 Tiago AE Ferreira。论将懒散量子游走算法用于在网格中搜索多个解。《信息科学》，2022年。