2019 年全国大学生信息安全竞赛

作品报告

作品名称：面向租户基于侧信道的容器云监控系统

电子邮箱：545558183@qq.com

提交日期：2019 年 6 月 5 日

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用A4纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者可以在此基础上增加内容或对文档结构进行微调。

5. 为保证网评的公平、公正，作品报告中应避免出现作者所在学校、院系和指导教师等泄露身份的信息。

目　　录

摘要

当下云计算已成为信息交互与存储的重要模式，虚拟化技术则是云计算中最关键、最核心的技术原动力。传统的云虚拟化技术普遍以虚拟机为基础，但是虚拟机作为一种重量级的虚拟化技术，有其局限性，在运维和性能等方面也有着一系列的问题。而容器是操作系统级别的轻量级虚拟化技术，在为云平台带来极大性能提升的同时，简化了开发和运维环节。因此，容器正在许多场景中逐步取代传统虚拟机。

目前广泛使用的云平台中存在两种典型的租用模式，一种是以独占物理机为前提的租用，如“专用硬件模式”等租用模式；另一种是更为常见的多租户共享同一物理机的租用模式。在独占物理机的租用模式下，由于平台不可信或者利益驱使等原因，有时租户获得的并不是独立的物理机却无法察觉。后果是，租户在利益上收到了侵害；更进一步，如果租户出于保密需求租用单一物理机，实际上却被分配与其他租户共享物理机，则其他租户的不合规操作甚至恶意的攻击可能使租户遭受更大的损失。而从云平台提供的监测手段来说，租户无法感知上述行为且缺乏自主权，云平台可能通过伪造或其他手段使对用户显示的监测结果正常，而实际上则侵害了租户的利益和安全。

针对上述存在的问题，我们基于云平台中宿主机与容器环境间存在的侧信道实现了面向租户的容器云监控系统，该系统通过三个创新点来解决上述问题：第一，系统通过侧信道收集宿主机与容器环境中的共享信息，结合其信息特征实现了针对云平台的监控检测。第二，系统面向租户的特性，使租户能够在容器内部实现云平台监控，主动感知云平台，而不需要依赖云平台提供的监测方法。从而解决了云平台可能伪造监控信息的问题。第三，系统是实时动态监测的，通过运行时动态检测实时地监控云平台，提升了租户预警能力，使租户能够及时采取相应的补救措施。

第一章 作品概述

1.1 研究背景

- **传统的云虚拟化技术普遍以虚拟机为基础**

2006 年，Google 首次提出“云计算”这个名词。经过十多年的发展，云计算已经从发展培育期步入快速成长期，越来越多的企业开始使用云计算服务。在全球范围内，云计算已成为人们进行信息交互与存储的重要模式，其核心技术也在发生着巨大的变化，新一代的技术正在改进甚至取代前一代技术。云计算的核心技术包括虚拟化、分布式数据存储、分布式并发编程模型、大规模数据管理以及分布式云计算平台管理技术等五大类，而这之中，虚拟化是一切建立在云上的服务与应用的基础，是提高服务效率的最佳解决方案。

虚拟化技术通过把多个操作系统整合到一台高性能服务器上，最大化利用硬件平台的所有资源，用更少的投入实现更多的应用，且简化 IT 架构、降低管理资源的难度、避免 IT 架构的非必要扩张。通过虚拟化技术，云计算把计算、存储、应用和服务都变成了可以动态配置和扩展的资源，从而才能够实现在逻辑上以单一整体的服务形式呈现给租户。这样，虚拟化技术可以将 IT 环境改造成为更强大、更具弹性、更富有活力的架构。所以说，虚拟化技术是云计算中最关键、最核心的技术原动力[1]。

传统的云平台上的虚拟化技术普遍以虚拟机为基础。但是虚拟机有其局限性也存在着一系列的问题。通过虚拟化软件创建虚拟机，需要人工指定放在哪台机器上、硬盘放在哪个存储设备上，人工指定网络的 VLAN ID、带宽的具体配置等。所以仅使用虚拟化的运维工程师往往需要一个表格，记录有多少台物理机、每台机器部署了哪些虚拟机。受此限制，一般虚拟化的集群数目不会特别大。

- **轻量标准的容器虚拟技术诞生**

容器（container）作为云基础架构的传统虚拟机的轻型替代应运而生。容器虚拟化技术以其轻便、灵活和快速部署等特性对传统的基于虚拟机的虚拟化技术带来了颠覆性的挑战，正在改变着基础设施即服务(IaaS)平台和平台即服务(PaaS)平台的架构和实现。

容器技术（container technology）提供了一种轻量级虚拟主机系统。所谓容

器就是将软件打包成标准化单元，以用于开发、交付和部署。容器化软件适用于基于 Linux 和 Windows 的应用，在任何环境中都能够始终如一地运行。其中，轻量的、可执行的独立软件包，包含着软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置，就叫做容器镜像。因此，容器赋予了软件独立性，使其免受外在环境差异（例如开发和预演环境的差异）的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突。

事实上，作为虚拟化技术的两种实现形式，容器和虚拟机具有相似的资源隔离和分配优势，但功能有所不同。容器虚拟化的是操作系统而非硬件，容器之间共享同一套操作系统资源，即容器内没有自己的内核，也没有进行硬件虚拟，因此其中的应用进程直接运行于宿主的内核；而传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，再在该系统上运行所需的应用进程。因此容器要比传统虚拟机更为轻便、更容易移植，效率也更高，但容器的隔离级别会稍低一些。图 1-1 表明了容器和虚拟机的架构。

图 1-1 容器和虚拟机的架构

- **Docker 是世界领先的软件容器平台**

2008 年建好的 LXC 是容器管理器的第一个实现。建在 LXC 基础上的 Docker 是近几年最流行的容器管理工具，Docker 可以把应用和它们的依赖（例如源码、运行时间、系统工具、系统库）打包为镜像，这样保证了应用在不同平台间可以移植。

诞生于 2013 年的 Docker 最初是 dotCloud 公司内部一个业余项目，在加入了 Linux 基金会以后，成为了一个遵从 Apache2.0 协议的开源项目。Docker 项目

的目标是实现轻量级的操作系统虚拟化解决方案，其基于 Linux 内核的 cgroup、namespace 以及 AUFS 类的 UnionFS 等技术，使用 Google 公司推出的 Go 语言进行开发，主要功能是通过实现对 LXC 的进一步封装，从而对进程进行封装隔离。

Docker 能够自动执行重复性任务，例如搭建和配置开发环境，从而解放了开发人员以便他们能够更加专注于软件开发。

租户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。

Docker 思想的关键点有三：集装箱、标准化（包括运输方式的标准化、存储方式的标准化和 API 接口的标准化）、隔离。

其主要的两个特点如下

轻量：在一台机器上运行的多个 Docker 容器可以共享这台机器的操作系统内核；它们能够迅速启动，只需占用很少的计算和内存资源。镜像是通过文件系统层进行构造的，并共享一些公共文件。这样就能尽量降低磁盘用量，并能更快地下载镜像。

标准：Docker 容器基于开放式标准，能够在所有主流 Linux 版本、Microsoft Windows 以及包括 VM、裸机服务器和云在内的任何基础设施上运行。

可见，Docker 的流行有其原因：1.提供隔离性。避免公用的服务器上资源易受到其他租户的影响；2.提供一致的运行环境。镜像提供了除内核之外完整的运行时环境，确保了应用运行环境一致性；3.有更快速的启动时间。可以做到秒级甚至毫秒级的启动时间，大大节约了开发测试及部署的时间；4.弹性伸缩和快速扩展。善于处理集中爆发的服务器使用压力；5.迁移方便。能轻易地将在一个平台上运行的应用迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行；6.能够持续交付和部署。以通过定制应用镜像来实现持续集成、持续交付和部署。

- **容器实现机制决定其与宿主机间共享信息**

正如前文所述，容器的出现深刻地改变了多层分布式应用在云端的构造、运输和部署模式，足以说明其是成功有效的。但是容器虚拟化的是操作系统而非硬件，容器之间共享同一套操作系统资源，即容器中的应用进程直接运行于宿主的内核。这种实现机制是容器与虚拟机相比性能得到提升的基础，带来了轻量灵活、

快速部署的优点，但是同时，这也决定了宿主机的信息必然与部署在其上的所有容器共享。目前在容器中已发现的信息共享通道证明，容器不仅仅能够获得其自身的资源分区信息，还能够共享全系统范围的主机信息。容器和宿主机的具体架构如图 1-2 所示。

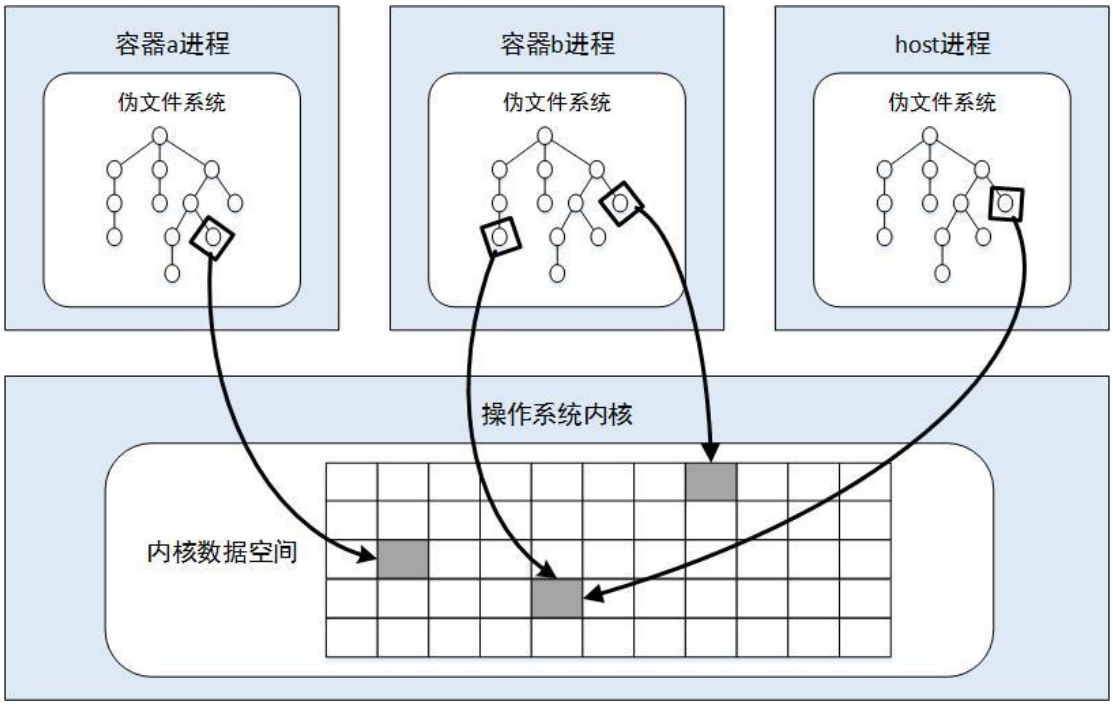

图 1-2 容器与宿主机共享操作系统内核

一方面，Linux 内核中系统资源的隔离机制具有不彻底性，另一方面，由于云计算具有多租户的使能特性，即支持不同租户的多个计算任务或实例运行在同一个物理设备上。当多个属于不同租户的容器共享一个同云服务器的操作系统内核时，其中一个租户通过共享信道也可以在一定程度上获取其他租户的信息。

如果一个租户向云平台提出专用设备的请求，理论上来说该租户应该获得一个独立宿主机的使用权限。但是，由于云平台公信力较差或者利益驱使等原因，有时租户获得的并不是独立的宿主机，一方面，在利益上收到了损害，另一方面，如果还存在其他的租户共享同一宿主机，其他租户的不合规操作甚至恶意的攻击可能使租户遭受更大的损失，如果是有特殊运行需求或者保密要求使用单一宿主机的情况，后果将不堪设想。

我们的工作就是基于这种现状下展开的，我们所设计的系统可以在云平台的环境下，通过共享的信息通道对环境实施监听和分析。一方面，能够发现是否存在其他的租户在与自己共享同一宿主机，及时发现平台的不合规操作来规避风险；

另一方面，如果存在多租户，还可以对其他租户的行为进行分析检测，及时发现存在安全隐患的或者有攻击性的行为，并通过邮件或者短信等方式进行预警，将带来损失危害的可能性降到最低。

1.2 相关工作

武志学等[2]对容器虚拟化技术进行了深入的介绍，并通过分析和比较阐明了容器虚拟化技术和虚拟机虚拟化技术各自的优势、适应场景和亟待解决的问题，并对云计算虚拟化技术的下一步研究方向和发展趋势进行了展望。

Gao Xing 等[3]调研并总结了容器信息共享的通道，深入全面地从源码分析了其根本原因，即类似的信息共享是由于 Linux 内核中容器实现的不完全覆盖造成；并提出一种带来的额外开销微不足道的两阶段的有效防御措施，以在容器云中解决这个问题；系统地探索和识别了可能与主机操作系统和共驻同一主机的容器共享的容器内置泄露信道，并将通过此侧信道获得的信息加以利用。此处可以共享的信息包括主机系统状态信息（功耗、工作数据、全局内核数据、异步内核事件）和容器私有进程的执行信息（进程调度、控制组、进程运行状态）。在特定时间展示出的这些特征信息可以用来唯一识别一台特定的物理机。

procfs 中蕴含侧信道共享信息[4]。已有一系列的研究表明操作系统及其下的 Linux 内核没能有效地控制从 procfs 产生的信息共享。procfs 中的数据可以分为两类：进程数据和全局统计数据。进程数据是与特定进程相关的信息，而全局统计信息报告来自所有进程和整个内核的聚合信息。

张和王等[5]演示了在 Linux 上通过 procfs 所共享的进程的堆栈信息，可获取租户的击键信息。Jana 和 Shitikov[6]发现在现代系统中的许多隔离机制下，通过 Linux 上的 procfs 可以观察到应用程序内存占用空间（例如数据常驻大小）的变化。钱等[7]利用/proc/net/中报告的错误数据包计数器。Zhou[8]演示了利用 Android 的 procfs 通过获得受害者 app 的数据包统计信息，来获知其活动的推断。Chen 等[9]从各种 procfs 的文件中提取受害者 app 的 CPU 使用时间、内存使用率和网络使用率，以检测活动转换，然后识别前台活动。林等[10]利用 procfs 提取应用程序的 CPU 使用率，以检测租户在 Android 上的按键操作。张等[11]通过 IP 摄像头的 Android 应用程序，探索了从 procfs 到指纹租户行为的等一系列类似的侧信道。Diao 等[12]研究了在 procfs 中使用全局中断计数器来推断租户的解锁模式和

前台 App。

云监控系统是云平台的重要组成部分，其主要任务是对物理主机和虚拟设备的资源使用情况和性能状况进行监控，帮助租户和系统管理员准确把握云主机的运行情况[13]。通过将云平台信息的整合，对系统集群的服务器、虚拟机的资源、性能等进行实时可靠的监控，更好的保障系统高效运行，方便租户及时准确的了解系统运行情况，帮助系统管理员及时发现系统故障和异常，从而更好的实现虚拟机的动态迁移、系统的负载均衡和集群的资源管理[14]。云监控目标主要包括物理机和虚拟机，常用的监控方式有两种：集中式监控和分布式监控。集中式监控主要是通过监控中心服务器实现对所有监控对象的直接监控，并对收集到的数据集中处理和展示。其最大的缺点在于当监控对象太多时，监控中心服务器的压力过大，容易造成瓶颈和阻塞，降低监控效率和可维护性，威胁系统安全；分布式监控的结构将监控压力下行，在监控中心服务器以下，设有多台子监控服务器，每个子监控服务器并行监控各自的子监控系统，通过将监控工作分散，降低了中心服务器的压力，减少了网络负载，提高了监控效率[15]。

随着云计算的迅速发展，传统的监控系统不能完全满足云计算平台对于监控的要求，一些具有代表性的商业或开源云监控系统也应运而生。由于云计算具有分布式的特性，因此云资源的监控架构往往借鉴于分布式计算的系统架构并进行实现。商业云计算平台都有专门用于自身云平台的面向开发者和管理员的监控系统，国外有 Amazon 的 Cloud Watch[16]、Google 的 Stackdriver[17]，还有风头正盛的初创企业 Datadog[18]；国内比较令人关注的有阿里云监控[19]、盛大云监控、监控宝和小米开源的 Open-Falcon 监控系统解决方案等。Amazon 利用 Cloud Watch 监控 AWS 云资源和在 AWS 上运行的应用程序，诸如 EC2 实例、Amazon Dynamo DB 表、Amazon RDS 数据库实例等，租户不仅可以自动提供 AWS 资源的指标信息，也可以提供自己的日志，同时自定义的指标也会被纳入 Cloud Watch 的监控范围之内。Google 在 2014 年通过收购云服务提供商 Stackdriver，后续推出了新版云监控服务。其服务通过 Google App Engine、Google Compute Engine、Cloud SQL 等获取基础设施的基本信息，也可实现对这些服务之上的应用进行监控。更重要的是开发者也能获得其云监控的使用权限，利用其开放的接口对自己的应用进行性能追踪和分析。Datadog 采用 Stats D[20]技术作为系统监

控的核心，其特点是轻量级，获取应用程序的监控信息非常便捷，开发者可以关注于业务逻辑，而无需在配置上花费过多精力。而在开源监控软件方面，Nagios[21]和 Zabbix[22]是其中的翘楚[23]。

1.3 特色描述

1.3.1租户自主检测

在当前云环境下，云平台对租户来说是透明的，其上的具体操作对租户来说也不可感知。当租户要求申请单一宿主机即租户需要单租户的环境，可能存在这样的情况：云平台承诺租户独享一台物理机，却将租户申请的容器和其他租户的容器在同一宿主机上部署，并向租户隐瞒此行为。租户作为消费者，若使用云平台提供的监测工具和数据，可能无法察觉到这种异样，也就无法得知自己的需求是否真正得到了满足、合法权益是否收到了侵害。以租户为中心的本检测系统创新性地提出租户自治的概念。不再需要像传统方法依赖云平台提供的检测工具和数据，而是给租户提供主动感知其不信任的云平台的各种行为，以便及时做出应对。

1.3.2运行时动态监测

根据调研，目前云服务商提供的监控技术在设计上只要求满足开发者和运维管理员的需要；而他们提供的日志架构是为了满足企业安全审计人员的需要，其目标主要为向云平台上 PaaS 内部的原生应用提供日志服务，不包括硬件、操作系统、数据库以及与云服务不相干的日志管理，而且只能应用于隐私泄露之后的审计，不具有即时性。而我们基于从侧信道获取的信息，实现了在透明的容器云环境下对所在整个宿主机实时地、动态地监测。

1.3.3过程可视化

现有的类似项目大多仅支持在 Linux 终端下的命令行操作，并且环境配置较为复杂，对向命令行的输入格式也有着苛刻的要求。因此为了能向租户全面动态地展示通过侧信道获取的整个云平台下的情况，我们提供了友好的租户界面，我们对采样模块、分析模块、传输模块和解析模块进行了一定程度封装并提供良好可视化扩展。

在具体实现上，Qt 和 Python 结合，容器端获取数据样本进行检测并传输给

客户端。客户端是个 TCP 服务器，监听某端口，等待容器端发起连接请求，收到请求后建立链接，接收容器端的传输数据，待容器端断开 TCP 连接后，检查、处理数据，并将数据转换成折线图展示。

1.4 应用前景

当下流行的云服务器稂莠不齐，从依托于互联网巨头企业赫赫有名的阿里云、腾讯云、百度云，到初创公司的后起之秀小鸟、UCloud、QingCloud[24]，同时还有某些公司自己搭建的云服务器，以供其客户或合作伙伴使用。现实中各种各样的云服务器很多，租户因某些原因不能使用大型云平台，而是选择了小型的、公信力不能满足所需安全程度的云服务平台。不知道平台中是否有操作会对自己产生安全威胁，所以需要此工具对整个平台下的操作进行监控，若发现有恶意的操作。除此之外，此产品可以用于任何云平台上来通过侧信道监测宿主机或其它任何容器的操作，从而确保容器的安全。

一方面，此产品可以应用于一些较小的、商业模式可靠程度较低的、信誉不是很高的云平台，对使用的环境和平台的行为进行监测分析，及时预警平台的不规范操作，从而确保自身的利益不受侵害。

另一方面，此产品还可以应用于多租户共同使用同一物理机的情况，对来自其他租户的异常行为和攻击举措进行及时的检测和反馈，将可能带来的损失降到最低。

第二章　作品设计与实现

2.1　系统方案

2.1.1系统目标

1. 针对当前云平台环境中缺乏租户自主监测的问题，我们系统旨在通过程序实现云平台租户自治监测，主动保证环境安全；

2. 针对云平台透明性所造成的异常行为难以及时检测的问题，我们系统通过运行时动态监测来提升及时性。

3. 针对云平台后台行为检测往往存在用户交互性差的问题，我们系统通过 UI 界面来实现用户交互友好。

2.1.2系统概述

本作品系统由六个模块构成，包括采样模块、预处理模块、训练模块、分析模块、通信模块和解析模块。训练模块只在系统设计的时候用于训练模型，在程序最终运行过程中并不参与。分析模块在执行分析操作的时候使用训练模块已经训练好的模型。

采样模块进行系统侧信道共享内容的采样；预处理模块通过做差、SAX 操作、BOP 固定来进行原始采样数据的处理。

训练模块通过收集经过采样模块和预处理模块处理得到的大量数据，送入 SVM 模型进行训练，经过 One vs. One 和 One vs. Rest 两种训练方式的训练比较最优解，生成模型，其具体实现方式如图 2-1 所示。

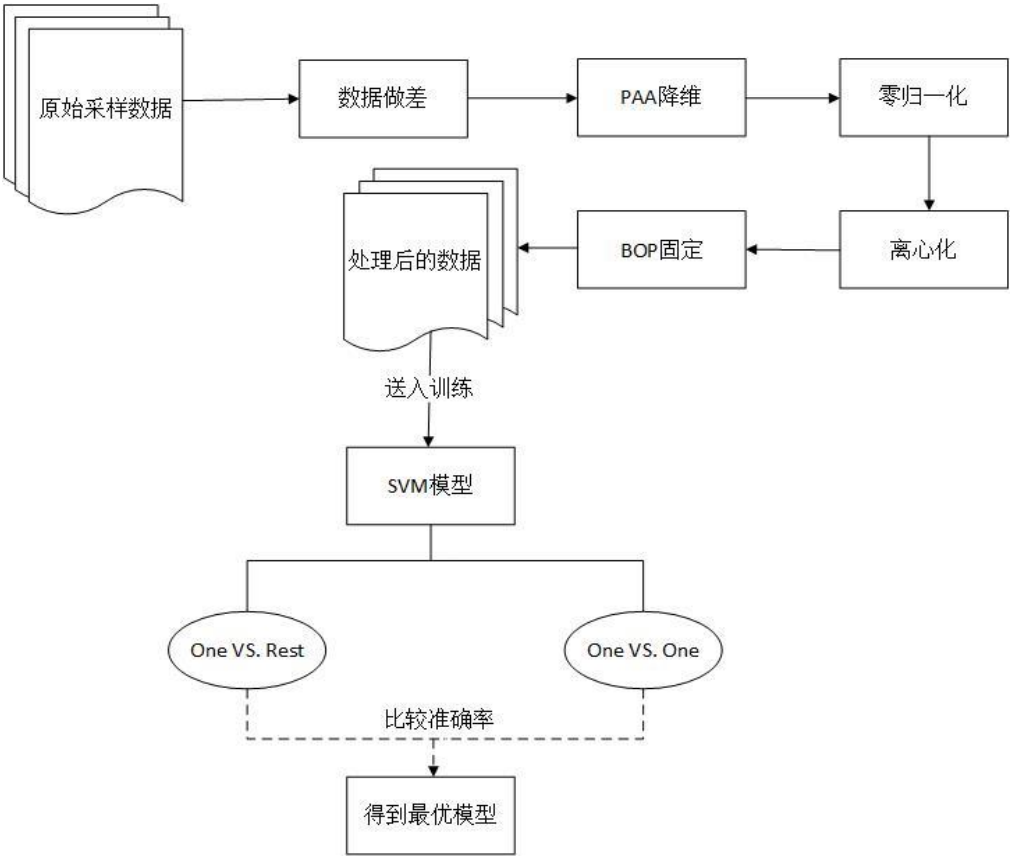

图 2-1 生成模型示意图

而分析模块在执行分析操作时，直接将经过预处理模块的数据输入已得到的模型中，得到分析结果。

通信模块通过将采样数据和结果打包封装，使用 TCP 协议传输给解析模块。

解析模块通过解析收到的报文，进行绘图展示等显示功能，完整的程序运行过程如图 2-2 所示。

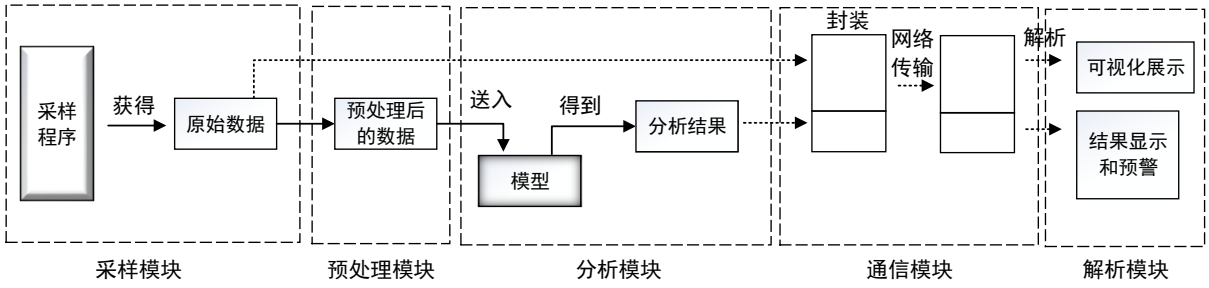

图 2-2 组织架构图

2.2 准备工作

proc 文件系统及侧信道

procfs 是一个由 UNIX 操作系统向租户提供的伪文件系统，它可以用于将虚拟和物理内存、cpu 使用情况等系统内部参数导出到租户空间。它是进程文件系统和内核文件系统组成的复合体，其包含许多文件，可用于监视、调试和更改运行中内核的参数。

procfs 还是一种将内核数据对象化，以文件形式进行存取的内存文件系统。这通过文件系统接口实现，用于输出系统运行状态，以文件系统的形式，为操作系统和应用进程间的通信提供了界面。因而使应用程序能够安全、方便地获得系统当前的运行状态和内核的数据信息，进而可以修改某些系统的配置信息[25]。

当下商业容器云服务中存在的信息共享情况如图 2-4 所示。

共享信道	共享信息	潜在的漏洞			容器云服务				
		Co-re	DoS	Info	CC1	CC2	CC3	CC4	CC5
/proc/locks	内核锁定的文件	●	○	●	●	●	●	●	◐
/proc/zoneinfo	RAM的物理信息	●	○	●	●	●	●	●	◐
/proc/modules	加载内核模块信息	○	○	●	●	●	●	●	●
/proc/timer_list	时钟和定时器的配置信息	●	○	●	●	●	●	○	●
/proc/sched_debug	任务调度程序行为	●	○	●	○	○	●	○	●
/proc/softirqs	调用软件中断处理的程序数	●	●	●	●	●	●	●	●
/proc/uptime	上升和空闲的时间	●	○	●	●	●	●	●	◐
/proc/version	内核、gcc的发行版本	○	○	●	●	●	●	●	●
/proc/stat	内核活动信息	●	●	●	●	●	●	●	◐
/proc/meminfo	内存信息	●	●	●	●	●	●	●	○
/proc/loadavg	CPU和IO的利用率	●	○	●	●	●	●	●	◐
/proc/interrupts	每次中断请求的中断数	●	○	●	●	●	●	●	●
/proc/cpuinfo	CPU信息	●	○	●	●	●	●	●	○
/proc/schedstat	安排统计信息	●	○	●	●	●	●	●	◐
/proc/sys/fs/*	文件系统信息	●	○	●	●	●	○	●	●
/proc/sys/kernel/random/*	随机数生成信息	●	○	●	●	●	●	●	●
/proc/sys/kernel/sched_domain/*	安排域信息	●	○	●	●	●	●	●	●
/proc/fs/ext4/*	Ext4文件系统信息	●	○	●	●	●	●	●	●
/sys/fs/cgroup/net_prio/*	分配运输的优先事项	○	○	●	●	●	○	○	○
/sys/devices/*	系统设备信息	●	●	●	●	●	●	○	○
/sys/class/*	系统设备信息	○	●	●	●	●	●	○	○

图 2-4 商业容器云中存在的侧信道情况

下文就 proc 文件系统下几个关键文件的参数做了具体的说明。

2.2.1 Cpuinfo

通过 proc 文件下的 cpuinfo 可以得到系统中 CPU 的提供商和相关配置信息。基于不同指令集的 CPU 产生的/proc/cpuinfo 文件不一样，基于 X86 指令集 CPU 的/proc/cpuinfo 文件包含的参数解释如下所示：

processor：系统中逻辑处理核的编号。对于单核处理器，

则可认为是其 CPU 编号，对于多核处理器则可以是物理核、或者使用超线程技术虚拟的逻辑核

vendor_id：CPU 制造商

cpu family：CPU 产品系列代号

model：CPU 属于其系列中的哪一代的代号

model name：CPU 属于的名字及其编号、标称主频

stepping：CPU 属于制作更新版本

cpu MHz：CPU 的实际使用主频

cache size：CPU 二级缓存大小

physical id：单个 CPU 的标号

siblings：单个 CPU 逻辑物理核数

core id：当前物理核在其所处 CPU 中的编号，这个编号不一定连续

cpu cores：该逻辑核所处 CPU 的物理核数

apicid: 用来区分不同逻辑核的编号，系统中每个逻辑核的此编号必然不同，此编号不一定连续

fpu：是否具有浮点运算单元（Floating Point Unit）

fpu_exception：是否支持浮点计算异常

cpuid level：执行 cpuid 指令前，eax 寄存器中的值，根据不同的值 cpuid 指令会返回不同的内容

wp：表明当前 CPU 是否在内核态支持对租户空间的写保护（Write Protection）

flags：当前 CPU 支持的功能

bogomips：在系统内核启动时粗略测算的 CPU 速度（Million Instructions Per Second）

clflush size：每次刷新缓存的大小单位

cache_alignment：缓存地址对齐单位

address sizes：可访问地址空间位数

power management：对能源管理的支持，还可选支持温度传感器、频率 id 控制等功能

2.2.2 Stat

proc 文件下的 stat 中包含系统启动以来的很多系统和内核的统计信息，包括 CPU 运行情况，中断情况，启动时间，上线文切换次数，运行中的进程等信息都在其中，其相应参数的即使如下所示：

user:租户态的 CPU 时间

nice：低优先级程序所占用的租户态的 cpu 时间

system：系统态的 CPU 时间

idle：CPU 空闲的时间（不包含 IO 等待）

iowait：等待 IO 响应的时间

irq：处理硬件中断的时间

softirq：处理软中断的时间

steal:其他系统所花的时间（个人理解是针对虚拟机）

guest：运行时间为客户操作系统下的虚拟 CPU 控制（个人理解是访客控制 CPU 的时间）

guest_nice：低优先级程序所占用的租户态的 cpu 时间。（访客的）

intr 这行展示系统中断的信息，第一个为自系统启动依赖，发生的所有中断的次数；然后每个数对应一个特定的中断自系统启动以来所发生的次数。

ctxt 这行展示自系统启动以来 CPU 发生的上下文交互的次数

btime 这行展示从系统启动到现在为止的时间（以 UTC 时间开始计算，单位为秒）

processes 这行展示自系统启动以来所创建的任务的个数

procs_runnig 这行显示当前运行队列的任务数目

procs_blocked 这行显示当前被阻塞的任务数目

spftirq 这行显示软中断的情况

2.2.3 Meminfo

proc 文件下的 Meminfo 是了解 Linux 系统内存使用状况的主要接口，其相应参数的即使如下所示：

MemTotal: 所有可用 RAM 大小 （即物理内存减去一些预留位和内核的二进制代码大小）

MemFree: LowFree 与 HighFree 的总和，被系统留着未使用的内存

Buffers: 用来给文件做缓冲大小

Cached: 被高速缓冲存储器（cache memory）用的内存的大小（等于 diskcache minus SwapCache ）.

SwapCached:被高速缓冲存储器（cache memory）用的交换空间的大小已经被交换出来的内存，但仍然被存放在 swapfile 中。用来在需要的时候很快的被替换而不需要再次打开 I/O 端口。

Active: 在活跃使用中的缓冲或高速缓冲存储器页面文件的大小，除非非常必要否则不会被移作他用.

Inactive: 在不经常使用中的缓冲或高速缓冲存储器页面文件的大小，可能被用于其他途径.

SwapTotal: 交换空间的总大小

SwapFree: 未被使用交换空间的大小

Dirty: 等待被写回到磁盘的内存大小。

Writeback: 正在被写回到磁盘的内存大小。

AnonPages：未映射页的内存大小

Mapped: 设备和文件等映射的大小。

Slab: 内核数据结构缓存的大小，可以减少申请和释放内存带来的消耗。

SReclaimable:可收回 Slab 的大小

SUnreclaim：不可收回 Slab 的大小（SUnreclaim+SReclaimable＝Slab）

PageTables：管理内存分页页面的索引表的大小。

NFS_Unstable:不稳定页表的大小

VmallocTotal: 可以 vmalloc 虚拟内存大小

VmallocUsed: 已经被使用的虚拟内存大小。

2.3 模块设计

2.3.1采样模块

信息共享通道本质上是记录系统使用情况的日志，所以采样模块的关键点主要是通过侧信道获得系统的使用情况，在 Linux 系统中已经有一些命令或者应用可以直接读取这些文件，针对已发现的存在侧信道信息共享的特征，我们读取/proc 目录下的信息。

实现采样模块的另一关键点是关于采样时间的控制。时间控制需要达到两个要求：一是要达到千次每秒的高频取样率，实验表明取样频率为千次每秒能够取

得较优的效果，二是要确保每次采样的时间间隔相同。为了达到较高的准确度，本系统使用 select()函数实现精准定时器。

2.3.2预处理模块

经过采样模块的工作，我们得到了原始数据，接下来进行数据的预处理，预处理的目的是使得数据更容易被分析提取特征，分为数据的做差，SAX 操作和 BOP 操作三个部分，流程图如图 2-5 所示。

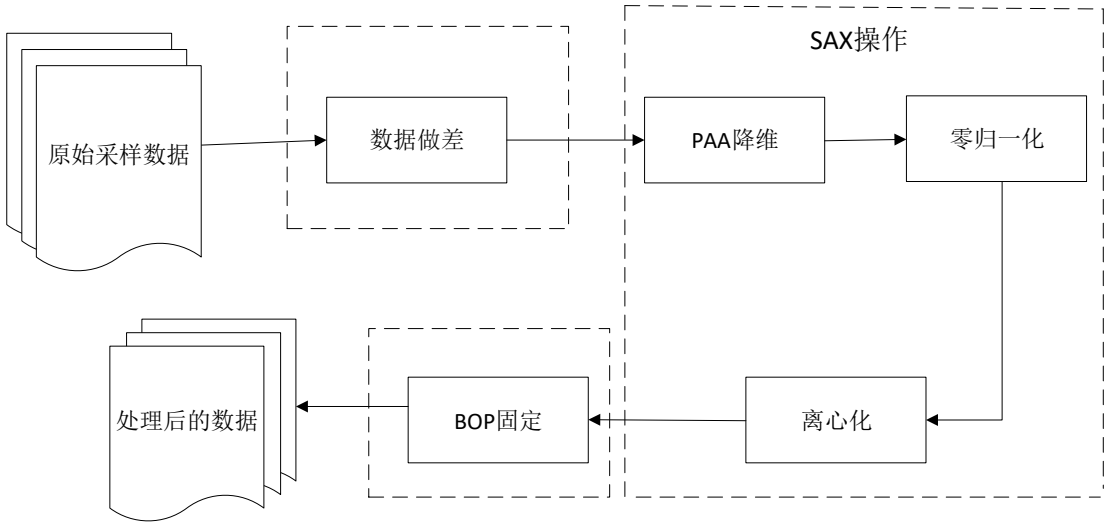

图 2-5 预处理模块流程图

获得了原始采样数据之后，由于我们只关心数据随时间序列的变化而不是对数据本身的绝对值感兴趣，所以对于每个时间序列$\mathrm{seq_k(k=1,2,\cdots,l)}$,我们计算每两个数据点之间的差异，即

$$\mathrm{diff_k[i]} = seq_{\mathrm{k}}[\mathrm{i}] - seq_{\mathrm{k}}[\mathrm{i}-1]$$

经过做差操作我们得到了某个侧信道观测值随时间的变化序列，我们称为差异时间序列。

SAX(Symbolic Aggregate approXimation)是由 Keogh 和 Lin 于 2002 年发明的方法。它使我们能够以有效的方式对时间序列进行编码，而不会丢失重要信息。它需要一个时间序列作为输入，它将输出一个表示此时间序列的字符串。

SAX 操作主要由三个部分组成：PAA 操作、零归一化操作和离心化。

PAA(分段聚合近似)操作实际上是一个降维操作，将 n 维的差异时间序列转换为 m 维的窗口(windows)，降维的目的是保障机器学习的效率，因为如果矩阵纬度过大会影响机器学习的结果。经过这样的操作每个窗口实际包含 p=n/m 个

数据点，作为 SAX 操作的第一步预处理，经过实验我们发现将窗口大小 windows 设为 5，能够取得比较好的效果。

零归一化操作，即 Z-normalization 操作，该过程确保输入向量的所有元素都被转换为平均值为 0 的输出向量，而标准偏差在接近 1 的范围内。若时间序列X_{ti}的平均值和标准差分别为$\hat{\mu}$ 和$\hat{\sigma}$,则转换的公式为：

$$X'_{ti} = \frac{X_{ti} - \hat{\mu}}{\hat{\sigma}}$$

转换完成后能够将序列变成遵循正态高斯分布的序列。

离心化的目的是为了将数字转换成字符串，经过零归一化操作后序列的值满足正态高斯分布，所以我们很容易确定切分断点将不同的数值和不同的字符对应起来，切分的总段数也会影响最终模型的训练结果，经过实验我们发现，切分成 5 段能够取得比较好的效果。

通过上述三步操作，已经将差异时间序列处理为一串字符串。

BOP(Bag-of-Patterns)操作由 Lin 和 Li 在 2009 年提出，其目的是为了让不定长的 SAX 字符串变成一个固定的数组，因为 SVM 需要一个规范长度的输入。其具体的操作为：创建所有关于 SAX 字符串的字典，然后给定字典，BOP 计算原始 SAX 字符串中不同单词出现的频率，为了避免过度复杂的匹配计算，对于原始 SAX 字符串中的相邻相同单词，我们只计算该单词的第一次出现。

例如，对于 α = 2 且 w = 2，字典的大小将是 α = 4，字典内容包括 aa，ab，ba，bb。BOP 计算原始 SAX 字符串中不同单词的频率，对于 SAX 字符串 aabaabbbbb，最终结果将是：aa：2，ab：2，ba：1，bb：1。（bb 属于字符串中相邻的相同单词，所以我们只计算一次）。

处理完成后得到的将会是定长的字符串，称之为特征。至此，预处理模块工作完成。

2.3.3分析模块

分析模块包含两个阶段，分别是训练阶段和系统运行时的检测阶段，训练阶段是输入大量的训练材料，生成训练模型；检测阶段是面向租户使用的部分，租户将采样并且经过预处理的数据送入训练模型，得到反馈的结果。

scikit-learn 是基于 Python 语言的机器学习工具。其特点包括：

1.简单高效的数据挖掘和数据分析工具；

2.可供大家在各种环境中重复使用；

3.建立在 NumPy，SciPy 和 matplotlib 上；

4.开源，可商业使用-BSD 许可证。

SVM(Support Vector Machines)是一组支持分类、回归、异常值检测的监督学习方法。优点如下：

1.在高维空间较好；

2.在特征空间远远大于样本空间时，依然较好；

3.决策函数中只使用了训练数据集的子集（称为支持向量）；

4.多功能：可以为决策功能指定不同的内核函数。提供了公共的内核，但是也可以指定定制的内核。

scikit-learn 中的 svm 支持稠密矩阵和稀疏矩阵样本向量作为输入。和其他的分类器一样，SVC/NuSVC/LinearSVC 将两个数组作为输入，分别是：二维向量(矩阵)X[n_samples, n_features]和一维向量(矩阵)Y[n_samples]。

X 矩阵大小为： 样本个数 * 单个样本的特征个数；

Y 矩阵大小为： 样本个数(每个样本都会有个分类/标签)。

SVMs 决策函数依赖于训练数据集中部分子集——支持向量(support vector), 一些支持向量对应着 SVMs 中的属性成员变量。

SVM 算法最初是为二值分类问题设计的，当处理多类问题时，就需要构造合适的多类分类器。目前，构造 SVM 多类分类器的方法主要有两类：一类是直接法，直接在目标函数上进行修改，将多个分类面的参数求解合并到一个最优化问题中，通过求解该最优化问题“一次性”实现多类分类。这种方法看似简单，但其计算复杂度比较高，实现起来比较困难，只适合用于小型问题中；另一类是间接法，主要是通过组合多个二分类器来实现多分类器的构造，常见的方法有 one-against-one 和 one-against-all 两种。

a.一对多法（one-versus-rest,简称 OVR SVMs）。训练时依次把某个类别的样本归为一类,其他剩余的样本归为另一类，这样 k 个类别的样本就构造出了 k 个 SVM。分类时将未知样本分类归为具有最大分类函数值的类。

假如有四类要划分（即 4 个 Label），他们是 A、B、C、D。于是在抽取训练

集的时候，分别抽取 A 所对应的向量作为正集，B,C,D 所对应的向量作为负集；B 所对应的向量作为正集，A,C,D 所对应的向量作为负集；C 所对应的向量作为正集，A,B,D 所对应的向量作为负集；D 所对应的向量作为正集，A,B,C 所对应的向量作为负集，将这四个训练集分别进行训练，得到四个训练结果文件，在测试的时候，把对应的测试向量分别利用这四个训练结果文件进行测试，最后每个测试都有一个结果 f1(x),f2(x),f3(x),f4(x)，于是最终的结果便是这四个值中最大的一个。

b.一对一法（one-versus-one,简称 OVO SVMs 或者 pairwise）。其做法是在任意两类样本之间设计一个 SVM，因此 k 个类别的样本就需要设计 k(k-1)/2 个 SVM。当对一个未知样本进行分类时，最后得票最多的类别即为该未知样本的类别。Libsvm 中的多类分类就是根据此方法实现的。

依然假设有四类要划分（即 4 个 Label），他们是 A、B、C、D。 在训练的时候选择 A,B; A,C; A,D; B,C; B,D;C,D 所对应的向量作为训练集，最终得到六个训练结果，在测试的时候，把对应的向量分别对六个结果进行测试，然后采取投票形式，最后得到一组结果。

在训练阶段，我们使用 scikit-learn 中的 svm 作为模型，将经过上述几步的处理得到的特征和对应的类别转换为二维向量矩阵，送入 SVM 进行训练，由于两种 SVM 训练方法各有优缺点，所以我们采用同时进行一对多法和一对一法进行训练学习，得到最终的结果选择准确度较高的作为训练后得到的模型，过程如图 2-6 所示。

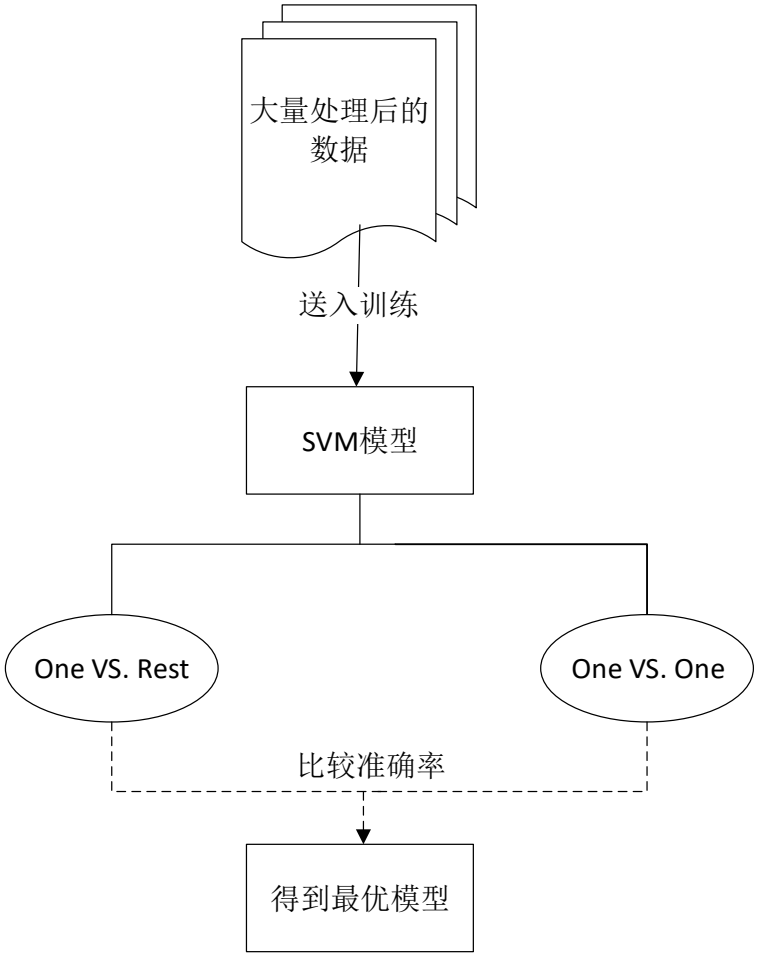

图 2-6 选取模型

2.3.4通信模块

通信模块的设立，主要是为了将运行在容器端的检测程序所采样得到的数据以及检测的结果传回桌面端，因此，基于实现目标，我们设计的通信模块需要满足：

1、模块由两个部分组成，即容器端和桌面端。其中，容器端部分负责数据的传输，桌面端负责数据的接收；

2、设计的容器端部分不能长时间处于工作状态，因为持续性的工作会对采样工作产生噪声，影响检测的正确性；

3、模块需要保证传输的可靠性，从而保证显示结果的正确性；

4、为保证传输的实时性，传输的数据量不可以过大。

综合考虑以上因素，在设计实现模块过程中，在传输数据采样处理后，选择了便于使用人员阅读的数据；采用了 TCP 连接传输数据来确保传输数据的可靠

性；容器端部分不作为服务器，只在需要传输数据时工作从而减小噪声；将桌面端程序作为服务器，等待连接请求，接收数据并解析展示；检测部分选择在容器端完成，与处理过的数据一起送回桌面端，从而在传输数据以及绘图所需要的时间的多方面考虑下，尽可能保证结果的实时性，。

容器端工作流程如下：

1、建立一个 Socket 对象，绑定桌面端 IP 和端口号，建立连接。

2、打开需要传输数据的文件，读取一行并传输，直至文件结束。

3、等待传输结束，断开连接。

容器端工作的过程如图 2-7 所示。

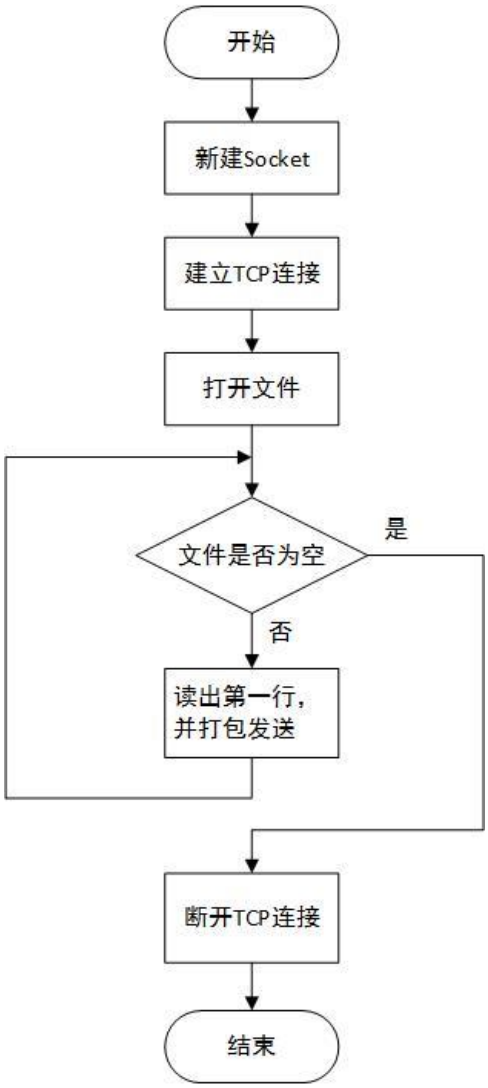

图 2-7 容器端通信实现

其中，TCP 连接是为每次传输数据建立。经过实验发现，若在最开始就建立 TCP 连接，直至客户程序结束运行才断开连接，在容器端会产生噪音，对其从宿

主机的采样造成影响。因此，实现方式为在每次需要传输数据前才发出请求并建立 TCP 连接，当次采样的数据发送完毕后断开。

桌面端工作流程如下：

1.建立 TCPServer 对象，绑定端口号，启动服务器。

2.接收到 TCP 连接请求时，新建一个 Socket 对象。

3.使用 Socket 对象接收数据，并存储。

4.接收到断开连接请求后，重新开始监听，等待连接请求。

2.3.5解析模块

在最终与租户进行交互的界面上，如果仅仅展示一个结果，必然是十分不直观，也十分没有说服力的。因为无论是检测程序的工作状态，还是检测结果的正确性租户都无从得知，而且直接展示的数据对于租户而言，是不友好而且效率低下的。因此，为了直观地展示数据，系统实现了将接收到的数据转化为折线图展示。对于本系统而言，一次检测的结果并不能成为对当前云平台做出评价的依据，因此仅仅展示检测结果是不够的，还需要对各种现象进行统计分析，为租户提供评价平台的依据。

考虑到接收到的数据量大且绘图所需时间长，为了提高实时性，本模块的实现采用了双缓冲机制，设置了前、后两个缓冲区。前缓冲区存储的数据用来绘图，后缓冲区用来接收数据。传输完成后，自动将后缓冲区数据拷贝至前缓冲区中，前缓冲区用数据开始绘图工作，后缓冲区继续接收数据。通过这种机制，模块实现了接收数据和绘图的并行工作，极大地提高了系统的实时性。

解析模块的工作流程如下：

1、通过字符串匹配算法，确定传输的数据指标的归属，并将之存入对应的后缓冲区数组中。

2、接收完成后，将前缓冲区清空，并将后缓冲区中的数据拷贝至前缓冲区对应的数组中。

3、使用前缓冲区数据绘图，清空后缓冲区，等待传输模块传输数据。

后缓冲区相关机制的工作如图 2-8 所示，前缓冲区相关机制的流程如图 2-9 所示。

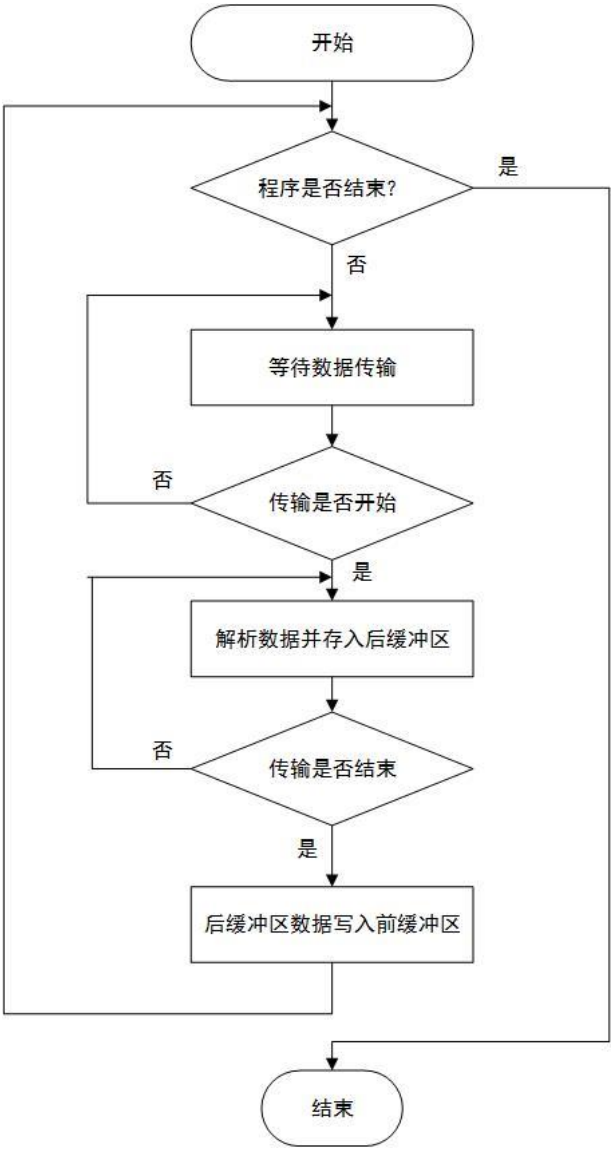
开始
程序是否结束?
是
否
等待数据传输
传输是否开始
否
是
解析数据并存入后缓冲区
传输是否结束
否
是
后缓冲区数据写入前缓冲区
结束

图 2-8 后缓冲区

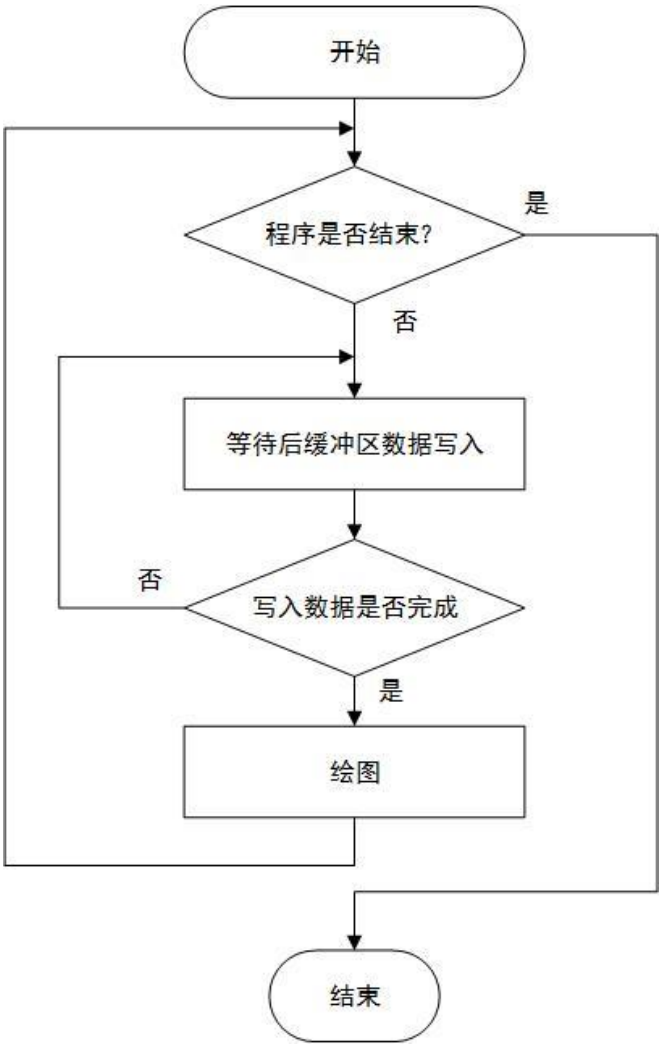

图 2-9 前缓冲区

第三章　作品测试与分析

3.1 开发环境

操作系统	CentOS-7-x86_64
基本开发环境	Gedit、Vim、Python、Shell
图形租户界面应用程序框架	QT Creator 5.9.6
机器学习模块	Scikit-learn

3.2 环境搭建及使用说明

3.2.1安装 VMware Workstation

在官方网站上下载 VMware Workstation 软件，在本地进行安装

3.2.2安装 Centos7 操作系统

在官方网站上下载 Centos7 操作系统镜像文件，在 VMware Workstatioon 中进行安装

3.2.3安装配置 Docker 环境

我们所有的程序是在 Linux 上运行，建议使用发行版 Centos 7。首先需要安装配置 Docker 环境，整个配置过程请按照顺序依次执行下列步骤。

表 3-1 Docker 安装配置步骤

1、Docker 要求 CentOS 系统的内核版本高于 3.10，通过 uname -r 命令查看当前的内核版本保证能够安装 Docker，查看验证命令如下所示：

```
$ uname -r
```

2、使用 root 权限登录 Centos，确保 yum 包更新到最新，更新命令如下所示：

```
$ sudo yum update
```

3、如果在先已经安装过 Docker 的旧版本，需要将旧的版本进行卸载，卸载

的命令如下所示：

```
$ sudo yum remove Docker Docker-common Docker-selinux Docker-engine
```

4、安装所需要的软件包。安装 yum-util，device-mapper-persistent-data 和 lvm2，安装 yum-util 是因为其能够提供 yum-config-manager 功能，而另外两个则是 device-mapper 驱动所依赖的，具体命令如下所示：

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

5、使用以下的命令来设置 yum 源，从而设定一个稳定的仓库：

```
$ sudo yum-config-manager --add-repo\
https://download.Docker.com/linux/centos/Docker-ce.repo
```

6、使用命令查看所有仓库中所有 Docker 版本，可以选择特定版本进行安装（默认选择最新的版本）：

```
$ yum list Docker-ce --showduplicates | sort -r
```

7、安装最新版本的 Docker，命令如下：

```
$ sudo yum install Docker-ce
```

8、启动运行 Docker 并且将其加入开机启动中：

```
$ sudo systemctl start Docker
$ sudo systemctl enable Docker
```

9、通过查看 Docker 版本验证安装是否成功，如果有 client 和 service 两部分表示 Docker 安装启动都成功了。

```
$ Docker version
```

3.3 系统测试

3.3.1测试数据集

在服务器上，启动 Docker，启动一个容器，设置好 IP 以及端口号，启动本系统的容器端程序。之后，在宿主机上每五秒进行一次操作，依次为无操作、pause、remove、stop、create 操作。

3.3.2可视化展示

打开桌面端程序。如图 3-1 所示，窗口左上方的输入栏用来供租户输入 IP、端口号、设置绘图时的坐标轴范围（用以改善数据的观看体验）。在本次测试中端口号等信息均为测试所准备的默认值，无需调整。直接点击开始，等待第一次的检测结果传输到桌面端后点击停止，如图 3-2 所示。

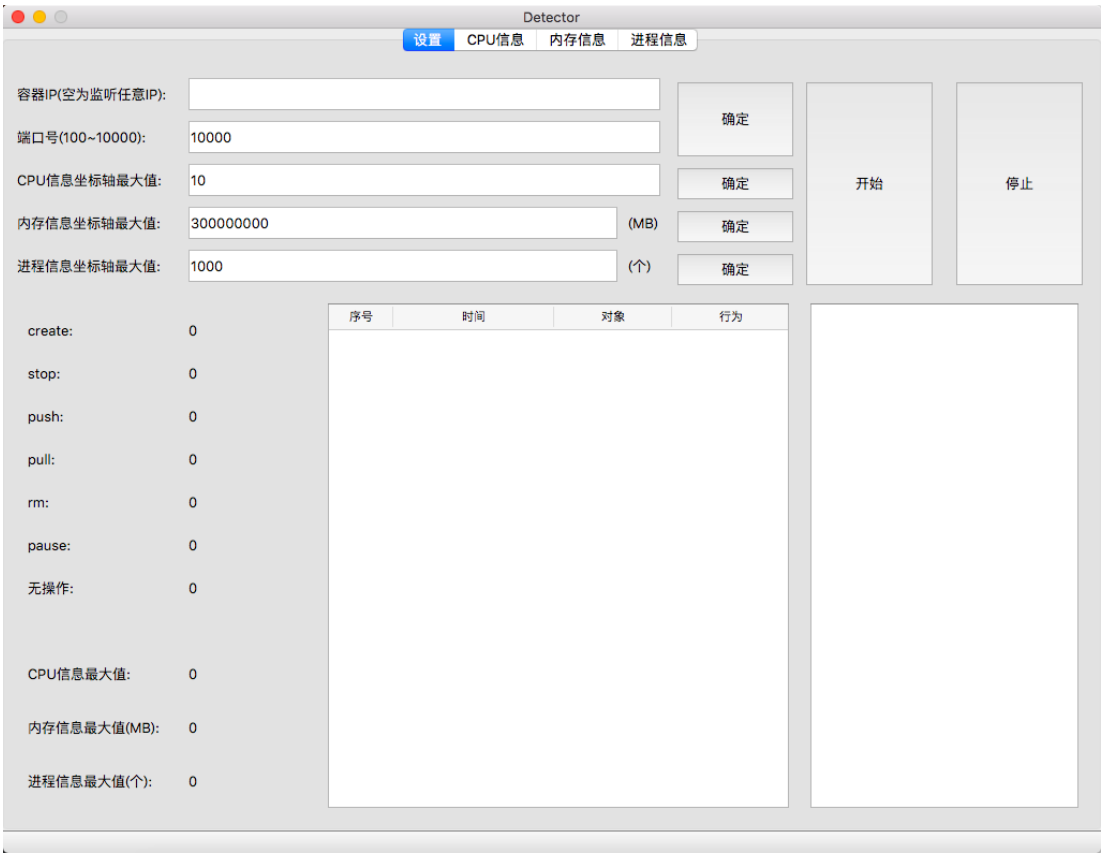

图 3-1 开始界面

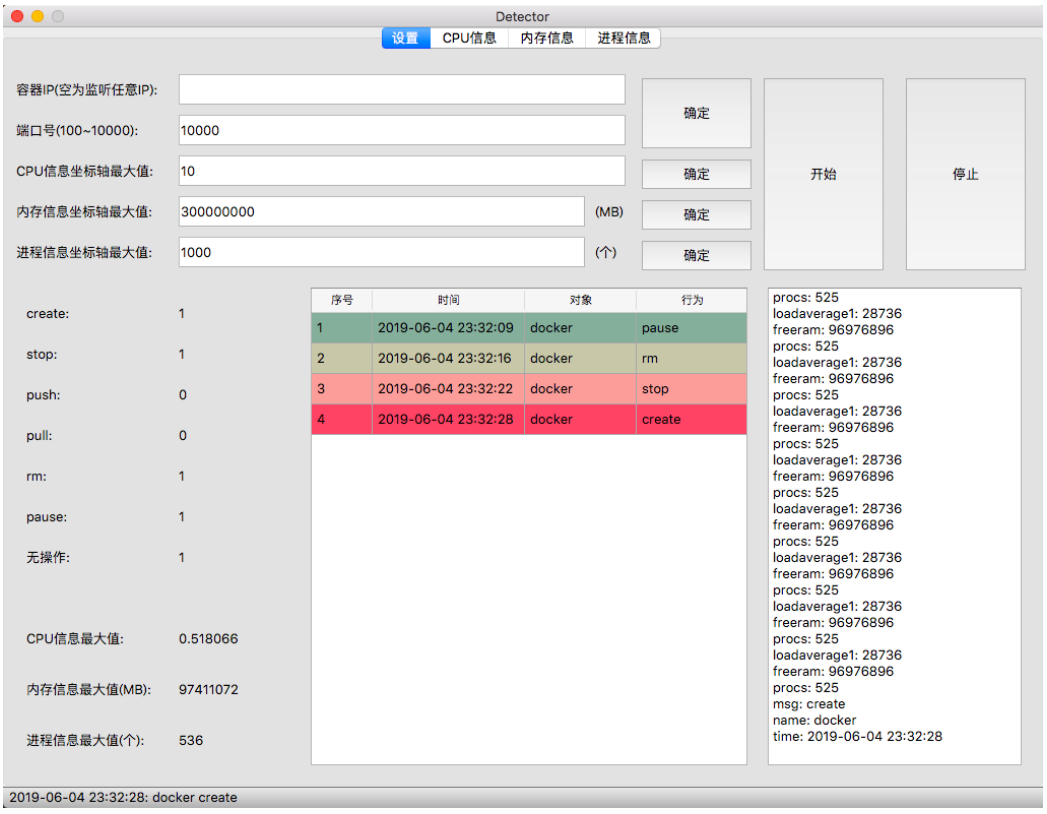

图 3-2 四次数据传入客户端

窗口的左下方，从上到下依次统计了自程序运行以来所有操作的出现次数，以及当前检测时各项指标的最大值，此数值不仅可以反映容器的一部分状态，同时也可以当做调整坐标轴时的一个参考，改善折线图的显示效果；窗口的中部是一个表格，用以统计自程序运行以来，所有异常操作的出现时刻，租户可以通过查看该表格，了解容器检测的异常操作的详细日志；窗口的右下部展示了上一次传输的数据的全部内容，主要用于判断程序是否正常工作，作为排错用途；最下方的状态栏展示了本次检测的最终的结果，即推断出的被监测容器中的操作。

点击 CPU 信息标签，由于数据的不可预测性，程序预设的坐标轴很难适用于所有数据，类似于这样的展示效果对用户很不友好，因此需要调整坐标轴，租户可以切换到设置页面修改坐标轴。但是我们并不推荐这么做，因为有更加便捷的做法：直接单击右上角的 format 按钮，程序会根据此时曲线的数值特点，自动调整坐标轴以适应屏幕，用户也可以通过直接在图上选择放大的相应的区域，调整到最适合阅读的状态，如图 3-3 所示。按照的同样的方法调整内存信息和进程信息，最终显示结果如图 3-4，3-5 所示。

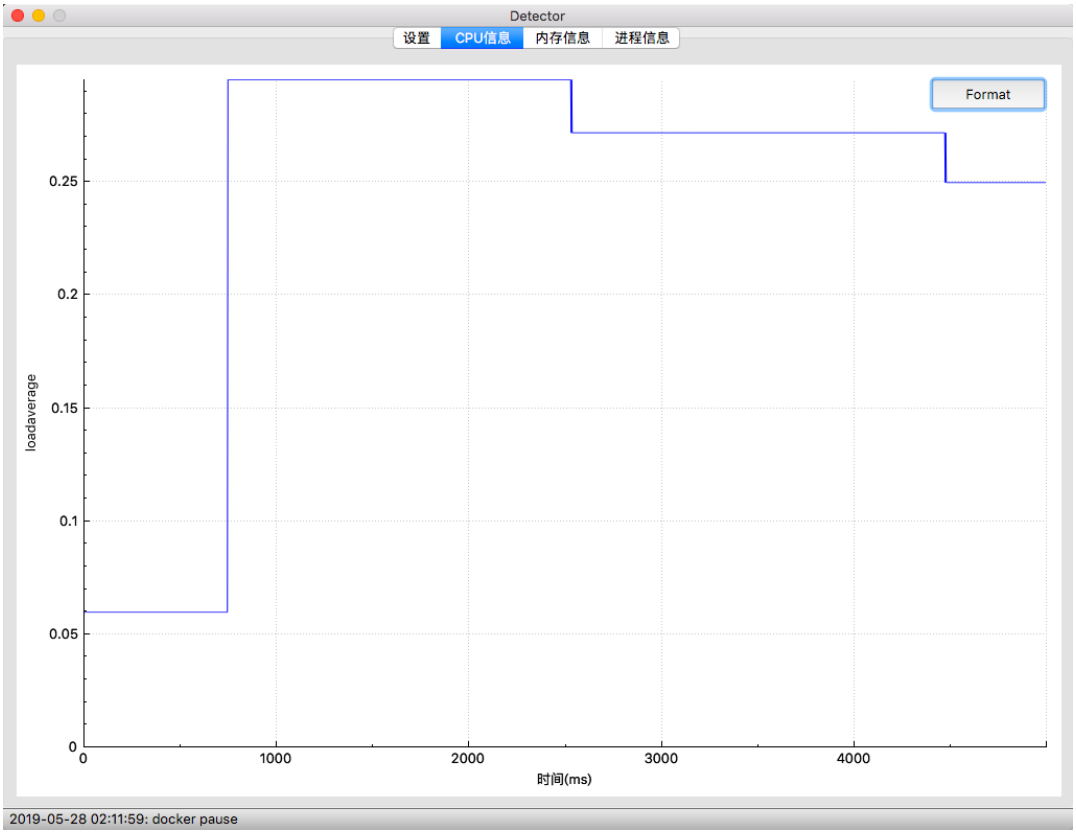

图 3-3 CPU 信息

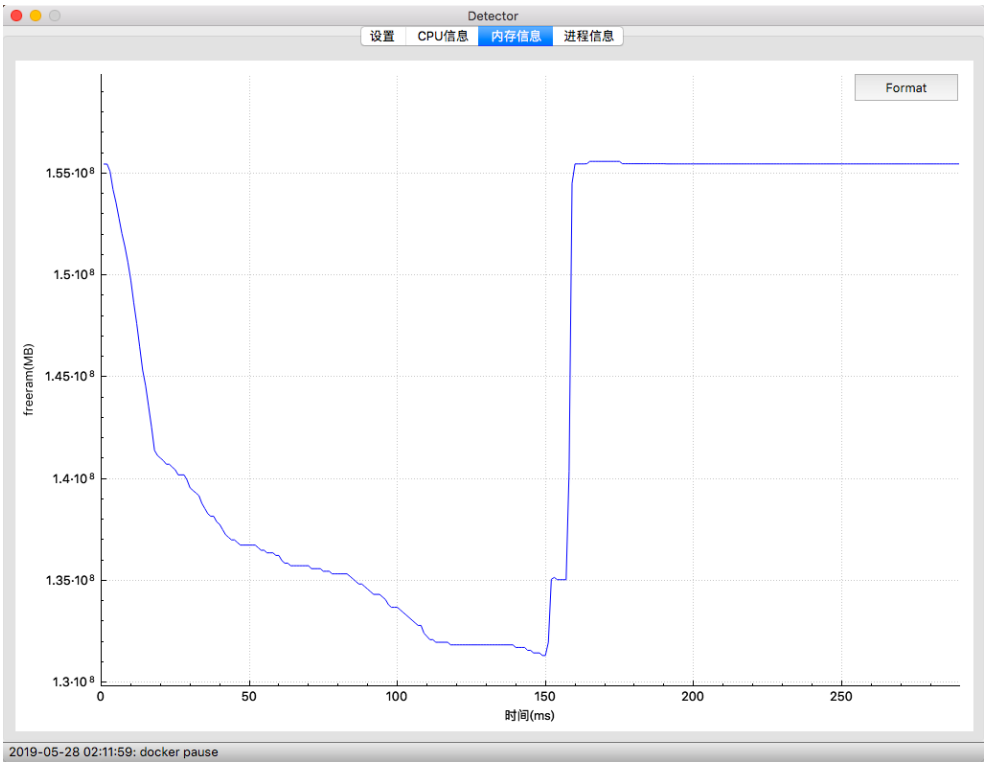

图 3-4 内存信息

图 3-5 进程信息

第四章　创新性说明

我们的产品存在诸多创新点。针对平台对租户有较强的透明性，我们创新性地提出租户自主监控的概念，使租户能够主动感知云平台行为；我们通过提取不同操作的特征来训练判断模型，因此该产品能够基本覆盖各种操作的检测，从而达到全面监测的能力；同时针对其他监测产品通过日志来提供服务，我们通过运行时动态检测，提高了作为监测软件的即时性和时效性，能够更为及时的通知租户减少损失。接下来的创新主要是集中在可视化等方面。

4.1 租户自主感知全宿主机的操作

租户在云平台部署容器等设施后，云平台对租户而言是透明的，即租户无法得知自己的需求是否得到了满足、在云平台上的行为合法权益是否受到了侵害，比如本作品所应用的领域，当租户要求申请一个相对独立的容器部署，即申请单一宿主机的情况。可能存在的情况是：云平台承诺租户该请求，却将租户申请的容器和其它不属于同一租户的容器在同一宿主机上部署，租户作为消费者若使用云平台提供的监测手段，可能无法察觉到这种异样。但本产品创新地提出租户自治能力。化被动为主动，使得租户能够主动感知云平台上的行为，而不需要依赖云平台提供的监测方法，掌握主动权和监测权，保护租户的合法利益不受公信力较低的云平台或其上别有用心的其他租户的恶意侵犯。

4.2 在容器运行时动态即时监测

传统虚拟环境监测工具，主要通过日志来进行监测环境的反馈，而不能对全宿主机上的操作行为进行实时动态的监控，即时性不强，这主要是因为租户的操作行为具有不可知性和透明性。相比之下，我们基于从侧信道获取共享信息，实现了在透明的容器云环境下对整个云平台实时地、动态地监测，从而能够取得较好的及时性，提高了预警和告知租户的能力，能够使租户更加及时的发现异常操作，并采取相应的补救措施。

4.3 系统可视化

由于现有的类似项目大多仅支持在 Linux 终端下的命令行操作，并且环境配置较为复杂，对向命令行的输入格式也有着苛刻的要求。因此为了能向租户全面动态地展示通过侧信道获取的整个云平台下的情况，我们提供了友好的用户界面，

我们对采样模块、分析模块、传输模块和解析模块进行了一定程度封装并提供良好可视化扩展。为租户提供了选中矩形框并全屏查看细节的功能、回看任意时间的容器状态的功能等。

第五章 总结

5.1 已完成的工作

我们开发出了一个具有显著特色和亮点的产品，解决了当前云平台容器所在宿主机独立性安全环境监测的两个主要问题：一个是失去公信力的云平台提供不可信的监测信息，作为租户无法获得真实反馈；再者便是租户无法知道非法行为的内容和发生时间。对此，租户自治能力贯穿了我们作品实现的整个过程，在确定了大体的框架后，我们所实现的作品分为以下几个主要的部分：

对于采样模块，我们首先确定了存在侧信道信息共享的具体内容，即/proc/下的特定相关信息，我们确定了采样频率以能够获得足够的特征数据，同时我们选择了合适的采样函数和处理采样时间间隔函数，从而得到原始采样数据。

对于预处理模块，在经过一定的调研之后，我们选择了做差、SAX 转型和 BOP 变换为流程进行原始采样数据的处理，并通过实验确定了进行上述操作的最佳参数。经过预处理模块能够在保留原始采样特征的前提下，规范化原始数据，更符合主流分类学习工具的输入。

对于分析模块，我们借助了当下的主流分类学习库，即 Python 中 scikit learn 库中的 SVM 模型，通过实验我们确定了最为合适的参数和取样区间，使得模型能够有较高的准确率，当模型训练完成后。使用者只需要将当前环境下的采样数据经过预处理，便可以送入模型中，并得到最终的监测分析结果。

对于通信模块，需要保证信息能够准确、快速地从容器端传输到桌面端，为了降低错误率，我们使用了 TCP 协议，并通过对传输数据的简化处理降低传输时间；我们还使用设置桌面端为服务器，等待容器端建立连接，和容器端程序在需要发送数据时启动，传输完成后即结束等方法减少噪声。

对于解析模块，由于传输模块每次会传输超过 15000 个数据，这些数据需要在下一次数据到来之前，被绘制出来，而 QCustomPlot 绘图程序较慢，为了解决这个问题，首先我们将对采样数据的检测流程放在了容器端，简化桌面端的流程。第二，我们采用了双缓冲区处理机制，后缓冲区负责接受数据，前缓冲区负责绘图，这样子可以实现接收数据和绘图的并行执行。事实上，容器操作的执行时间非常短，采样的大量数据只有一小部分是租户能看出变化的，为了方便租户观察

曲线的局部信息，我们提供坐标轴的缩放和移动功能，以及设定坐标轴最大值的功能，同时在统计信息的时候，给出了各个数据量的最大值，给租户设定坐标轴最大值的参考。最后，我们统计了各个行为出现的时间和次数，以供租户对当前云平台进行评价。

5.2 存在的问题与不足

1.目前采样模块对系统的性能存在一定的影响

在系统使用的过程中，我们需要不间歇地运行采样模块，从而达到实时监控的目的，然而采样程序的运行必然会对系统的性能造成一定的影响，消耗掉部分系统资源，我们可能需要进一步改进来减少对系统性能的消耗。

2.信息传输的条件设置较为基础

当前的传输模式是监测分析部分和展示解析部分直接通信，这需要展示解析部分有一个公网 IP，或者两个部分在同一个子网下，否则无法正常工作。针对这一问题，我们可能需要通过优化通信模块来进行分析操作。

3.数据集的不完善性导致一定程度的误报

在实践的过程中，我们使用基于收集到的数据集所训练生成的模型来进行结果的分析。这些数据集在进行采集的时候，可能存在采样过程中的不准确，导致对于某些采样数据模型并不能够给出很好的预测结果，这便会导致一定程度的误报。我们可能需要通过优化采样过程来进行改进。

5.3 未来的相关工作

1.增加解析模块功能

我们当前的界面能够基本满足要求，仍有可优化之处，后续可以增加一些柱状图或者饼图来展示统计结果，最终可以增加一个评价功能，即通过监测容器较长一段时间运行的数据，得到其一段时间内的操作集，之后根据一定的标准对容器进行评分，可以作为评价容器的依据。例如：若检测到某容器频繁地出现异常操作，则对此容器的评价分数会降低。这样，通过分数的形式直观地展示给租户当前容器的运行情况，给租户更直接更优秀的体验。也可增加租户的控制功能。租户可以选择停止监控或开始监控；当对检测结果不满意时，可以选择重新训练模型。

时间安排：2019 年 8 月

2.优化通信传输模式

我们可以优化网络传输，降低噪音对采样的影响。例如可以对数据进行压缩编码处理，减少数据的传输量，降低网络延迟。更进一步的，可以改变当前两个部分直接通信的传输模式。可以增加一个服务器，接收容器端的数据；客户端通过 http/https 请求来获取数据，甚至不需要客户端，直接登录 web 页面即可获得容器端发送给租户的数据。

时间安排：2019 年 8 月。

3.增加模型准确率和操作拓展

在后续的工作中，通过广泛调研异常行为，我们能够收集形成更加全面完备的采样数据库，从而降低模型的误报率，并且丰富程序所能识别区分的异常操作。

时间安排：2019 年 10 月

参考文献

[1]武志学. 云计算导论:概念架构与应用[M]. 北京:人民邮电出版社，2016:43－52. (WU Z X. INTRODUCTION TO CLOUD COMPUTING: CONCEPTS FRAMEWORKS AND APPLICATIONS [M] . BEIJING: POSTS AND TELECOM PRESS, 2016:43－52.)

[2]武志学.云计算虚拟化技术的发展与趋势[J].计算机应用,2017,37(04):915-923

[3]Gao,Xing & Gu, Zhongshu & Kayaalp, Mehmet & Pendarakis, Dimitrios & Wang, Haining. (2017). ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds. 237-248. 10.1109/DSN.2017.49.

[4]Zhang, Xiaokuan & Wang, Xueqiang & Bai, Xiaolong & Zhang, Yinqian & Wang, Xiaofeng. (2018). OS-level Side Channels without Procfs: Exploring Cross-App Information Leakage on iOS. 10.14722/ndss.2018.23272.

[5]K. Zhang and X. Wang, “Peeping Tom in the neighborhood: Keystroke eavesdropping on multi-user systems,” in 18th USENIX Security Symposium,2009

[6]S. Jana and V. Shmatikov, “Memento: Learning secrets from process footprints,” in 2012 IEEE Symposium on Security and Privacy, 2012

[7]Z. Qian, Z. M. Mao, and Y. Xie, “Collaborative TCP sequence number inference attack: How to crack sequence number under a second,” in 19th ACM Conference on Computer and Communications Security, 2012

[8]X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, “Identity, location, disease and more: Inferring your secrets from Android public resources,” in 20th ACM Conference on Computer and Communications Security, 2013

[9]Q. A. Chen, Z. Qian, and Z. M. Mao, “Peeking into your app without actually seeing it: UI state inference and novel Android attacks,” in 23th USENIX Security Symposium, 2014

[10]C.-C. Lin, H. Li, X. Zhou, and X. Wang, “Screenmilker: How to milk your Android screen for secrets,” in 21st ISOC Network and Distributed System Security Symposium, 2014

[11]N. Zhang, K. Yuan, M. Naveed, X. Zhou, and X. Wang, “Leave me alone: App-level protection against runtime information gathering on android” in 36th IEEE Symposium on Security and Privacy, 2015

[12]W. Diao, X. Liu, Z. Li, and K. Zhang, “No pardon for the interruption: New inference attacks on android through interrupt timing analysis” in 37th IEEE Symposium on Security and Privacy, 2016

[13]Chaves D, Aparecida S,Uriarte R B,et al.Toward an architecture for monitoring private clouds[J].Communications Magazine, IEEE: 2011.49(12): 130-137.

[14]Rimal,B.P. Eunmi Choi,Lumb, I. A Taxonomy and Survey of Cloud Computing Systems[C]. INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference:2009.(8):44-51.
[15]石宇婷. 基于 ZABBIX 平台的私有云监控系统设计与实现[D].东华大学,2017
[16]Liu D, Pei D, Zhao Y. Application-aware latency monitoring for cloud tenants via Cloud Watch+[C]. Network and Service Management (CNSM), 2014 10th International Conference on. IEEE, 2014: 73-81.
[17] Stackdriver Monitoring, https://cloud.google.com/monitoring/docs/
[18]Guide to Monitors, http://docs.datadoghq.com/guides/monitors/
[19]Cloud Monitor, https://cn.aliyun.com/product/jiankong
[20]Collecting Metrics Using Stats D, a Standard for Real-Time Monitoring , https://thenewstack.io/collecting-metrics-using-statsd-a-standard-for-real-time-monitoring/
[21]Barth W. Nagios: System and network monitoring[M]. No Starch Press, 2008.
[22]Olups R. Zabbix 1.8 network monitoring[M]. Packt Publishing Ltd, 2010.
[23]朱文标. 云平台多粒度监控系统的设计与实现[D].中山大学,2017.
[24]左雾.云计算服务企业 100 强[J].互联网周刊,2018(12):92-93
[25]李东亮,王海花.基于/PROC 文件系统及对内核信息的获取[J].河北工程大学学报(自然科学版),2007(02):73-77.