

基于 Online Boutique 微服务系统

的故障检测与复现实验

一、实验目的

本实验旨在通过部署开源微服务系统 Online Boutique，结合 ChaosMesh、Prometheus+Grafana、Selenium 与 JMeter 工具链，构建一个完整的微服务实验平台。通过故障注入与性能测试采集系统数据，复现异常检测相关论文中的算法，评估其在真实系统中的应用效果。

二、实验环境

```
matplotlib==3.10.3
numpy==2.2.6
pandas==2.3.0
PyYAML==6.0.2
PyYAML==6.0.2
Requests==2.32.3
scikit_learn==1.7.0
seaborn==0.13.2
selenium==4.33.0
torch==2.7.1+cu128
```

三、微服务系统介绍

本实验部署的微服务系统为 Online Boutique（原地址：<https://github.com/JoinFyc/Online-Boutique>），它是 Google 提供的基准演示项目，包含约 12 个基于容器化的微服务组件，典型用于研究服务通信、负载测试与可观测性技术。

系统结构图示意：

frontend：前端 UI，处理用户请求

productcatalogservice：商品目录管理

cartservice：购物车管理

checkoutservice：结算流程逻辑

currencyservice：货币转换

emailservice：下单通知服务

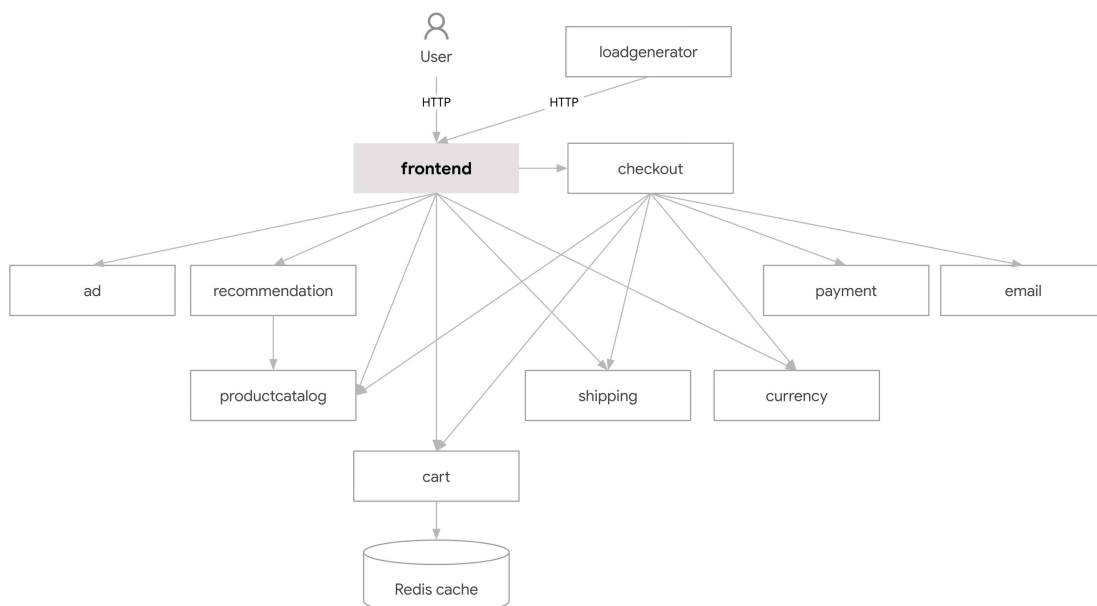
recommendationservice：推荐引擎

adservice：广告展示服务

其他服务：paymentservice, shippingservice, loadgenerator（负载生成器）

组件交互说明：

当用户进入系统浏览商品 → 添加至购物车 → 结算 → 支付成功 → 收到邮件通知，该流程将串联起以上多个服务。通过 **Kubernetes** 统一编排部署，并使用 **Docker** 实现微服务的容器化。



四、项目测试展示

1. Selenium 自动化功能测试

运行测试脚本 `selenium_test.py`，功能正常。

核心测试内容

模拟手机屏幕：测试网站在移动设备上的布局 and 交互是否正常，确认元素在小屏幕上能正确显示和操作

模拟网络延迟：通过 CDP 设置 3000ms 延迟和 50kbps 吞吐量，测试网站在弱网环境下的加载性能和用户体验。

核心功能测试：测试商品浏览-购物车管理-结账流程是否出现异常

异常处理：记录测试过程中的异常

测试结果

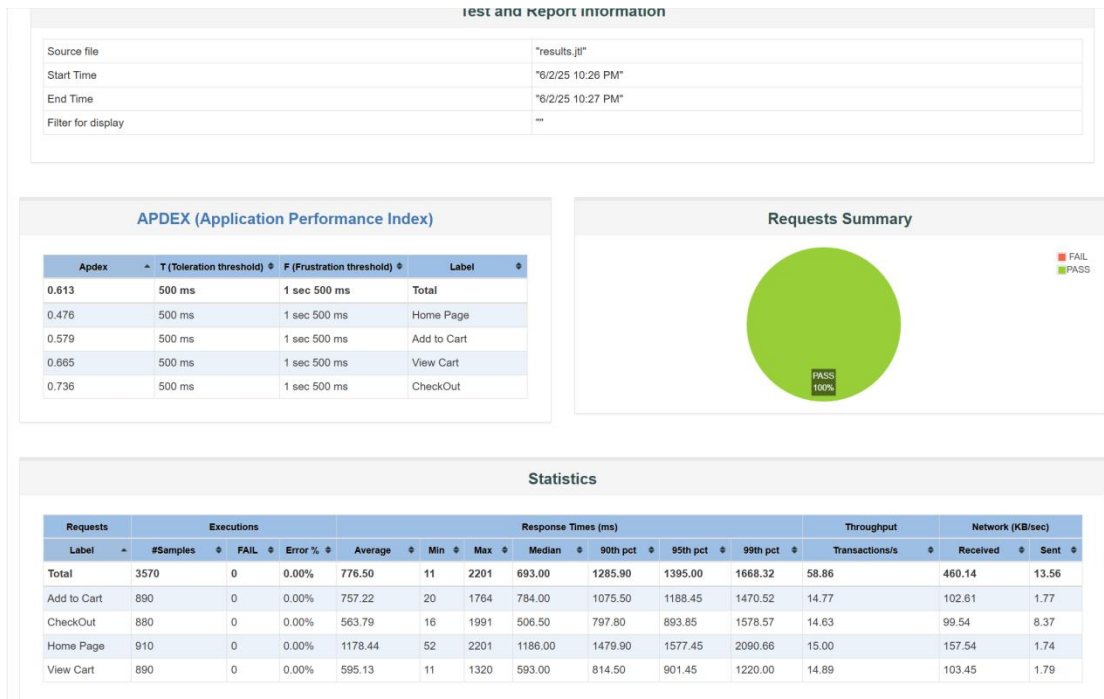
购物的核心路径能完整执行，页面跳转、按钮交互、购物车数据管理功能是有效的。

网络延迟模拟开启成功且流程未中断，说明系统在弱网环境下，基础功能仍可执行，具备一定弱网容错性

测试详情见视频：Selenium_test.mp4

2. JMeter 性能测试

测试报告：



性能表现

Requests Summary 里所有请求 100% 通过，说明系统在当前测试场景下，功能层面能稳定响应，没有报错或超时终止的情况

吞吐量达到 58.86/s，各业务也有稳定吞吐量输出，说明系统能承载一定并发高并发下，系统处理延迟不稳定，可能存在资源竞争

响应时间

所有业务响应时间随测试推进而上升，说明系统在持续运行下处理延迟在增加

高百分位（90+）曲线斜率大，说明长尾请求延迟明显。同时线程数上升后，响应时间波动大、峰值高，说明高并发下资源竞争加剧

稳定性

无请求失败，短期运行时系统没出现崩溃、服务中断

整体来看，测试验证了核心流程功能稳定性，但性能体验有优化空间。

五、故障注入与监控

1. ChaosMesh 故障注入实验

docker 部署 Chaos-Mesh:

运行 `kubectrl port-forward -n chaos-testing svc/chaos-dashboard 2333:2333`

按要求编写并运行 `rbac.yaml` 并生成 `token`

绑定 token



```
{'apiVersion': 'chaos-mesh.org/v1alpha1', 'kind': 'PodChaos', 'metadata': {'name': 'pod-failure-20250608-171324', 'namespace': 'chaos-testing'}, 'spec': {'action': 'pod-kill', 'mode': 'one', 'selector': {'namespaces': ['default']}, 'duration': '10s'}}
第566次执行结果: podchaos.chaos-mesh.org/pod-failure-20250608-171324 created

{'apiVersion': 'chaos-mesh.org/v1alpha1', 'kind': 'PodChaos', 'metadata': {'name': 'pod-failure-20250608-171325', 'namespace': 'chaos-testing'}, 'spec': {'action': 'pod-kill', 'mode': 'one', 'selector': {'namespaces': ['default']}, 'duration': '10s'}}
第567次执行结果: podchaos.chaos-mesh.org/pod-failure-20250608-171325 created

{'apiVersion': 'chaos-mesh.org/v1alpha1', 'kind': 'PodChaos', 'metadata': {'name': 'pod-failure-20250608-171327', 'namespace': 'chaos-testing'}, 'spec': {'action': 'pod-kill', 'mode': 'one', 'selector': {'namespaces': ['default']}, 'duration': '10s'}}
第568次执行结果: podchaos.chaos-mesh.org/pod-failure-20250608-171327 created
```

2. Prometheus + Grafana 实时监控

docker 部署 Prometheus + grafana:

```
(base) PS C:\Users\chen> minikube service prometheus -n monitoring --url http://127.0.0.1:64931
! 因为你正在使用 windows 上的 Docker 驱动程序，所以需要打开终端才能运行它。
```

```
(base) PS C:\Users\chen> minikube service grafana -n monitoring --url http://127.0.0.1:64973
! 因为你正在使用 windows 上的 Docker 驱动程序，所以需要打开终端才能运行它。
```

运行 `kubectl port-forward pod/node-exporter-4z6zr 9100:9100 -n monitoring` 暴露 node-exporter 的 9100 端口到集群外部

```
(base) PS C:\Users\chen> kubectl port-forward pod/node-exporter-4z6zr 9100:9100 -n monitoring
Forwarding from 127.0.0.1:9100 -> 9100
Forwarding from [::1]:9100 -> 9100
Handling connection for 9100
Handling connection for 9100
```

3. 监控图示结果分析



故障注入分析

资源波动: CPU 短时冲高、磁盘写入突发、I/O 耗时增加, 属于 Pod 中断+重启类故障的资源特征

临界瓶颈: 内存、网络带宽等关键指标均在安全阈值内, 说明系统具备基础韧性, 未因单次故障引发级联崩溃

优化方向: 可针对 I/O 和 CPU 短时高峰分别做 I/O 隔离和并行任务限制

六、异常数据采集与分析

选择论文 KDD20-UASD

1. 数据导出与预处理

正常状态数据生成:

多线程运行 Jmetet 测试 用作模拟用户访问 Online-Boutique 微服务
 编写并运行 集群指标抓取脚本 生成包含集群各项指标的 .txt 文件
 编写并运行 集群指标预处理脚本 将.txt 内的数据处理为 .csv 文件
 编写并运行 集群状态标记脚本 标记当前数据为 Normal
 编写并运行 数据清洗脚本 清洗无关数据,运算并保留以下数据
`other_cols = ['window_start', 'window_end', 'Normal/Attack']`

`load_cols = [col for col in df.columns if 'node_load' in col]`

`keep_cols = other_cols + load_cols + ['cpu_util', 'mem_util', 'disk_read_mbps',
 'disk_write_mbps', 'net_rx_mbps', 'net_tx_mbps', 'fs_util']`

获得 normal0.csv

异常状态数据生成:

编写 故障脚本 使得随机向 final(Online-Boutique 运行的位置) 命名空间的 pod 注入故障

编写并运行 故障注入脚本 每 1~2s 进行故障注入

多线程运行 Jmetet 测试 用作模拟用户访问 Online-Boutique 微服务

运行 集群指标抓取脚本 生成包含集群各项指标的 .txt 文件

运行 集群指标预处理脚本 将.txt 内的数据处理为 .csv 文件

运行 集群状态标记脚本 标记当前数据为 Attack

运行 数据清洗脚本 清洗无关数据,运算并保留以下数据

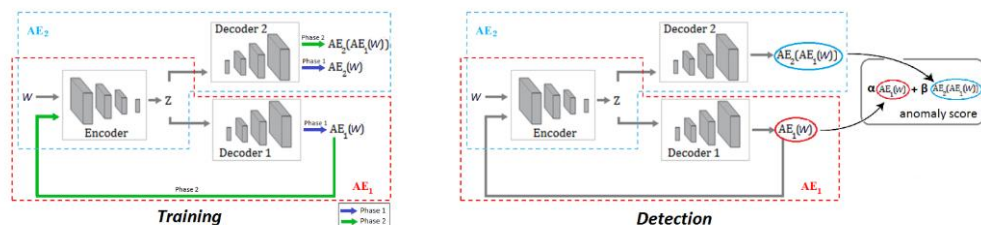
`other_cols = ['window_start', 'window_end', 'Normal/Attack']`
`load_cols = [col for col in df.columns if 'node_load' in col]`
`keep_cols = other_cols + load_cols + ['cpu_util', 'mem_util', 'disk_read_mbps', 'disk_write_mbps', 'net_rx_mbps', 'net_tx_mbps', 'fs_util']`

添加少量 Normal 数据到异常状态数据中

获得 attack0.csv

2. 论文算法复现

核心架构与原理



USAD 由一个编码器 E 和两个解码器 D_1 、 D_2 组成,

形成两个共享编码器的自动编码器 AE_1 和 AE_2

自动编码器训练: AE_1 和 AE_2 分别学习重建正常输入窗口, 最小化重建误差

对抗训练:

AE1 生成一种“伪正常”的数据，使得 AE2 无法区分它和真实的正常数据
AE2 提高对数据的识别能力，尽可能区分真实的正常数据和 AE1 的重构数据

异常评分机制

AE1 和 AE2 的损失函数：

$$\mathcal{L}_{AE_1} = \frac{1}{n} \|W - AE_1(W)\|_2 + \left(1 - \frac{1}{n}\right) \|W - AE_2(AE_1(W))\|_2 \quad (7)$$

$$\mathcal{L}_{AE_2} = \frac{1}{n} \|W - AE_2(W)\|_2 - \left(1 - \frac{1}{n}\right) \|W - AE_2(AE_1(W))\|_2 \quad (8)$$

异常分数定义：

$$\mathcal{A}(\hat{W}) = \alpha \|\hat{W} - AE_1(\hat{W})\|_2 + \beta \|\hat{W} - AE_2(AE_1(\hat{W}))\|_2$$

通过调整 α 和 β 的比例，可动态平衡检测的“灵敏度”与“精确性”，
最终通过与阈值的比较，来判断是否为异常部分

本地配置 USAD 的运行环境

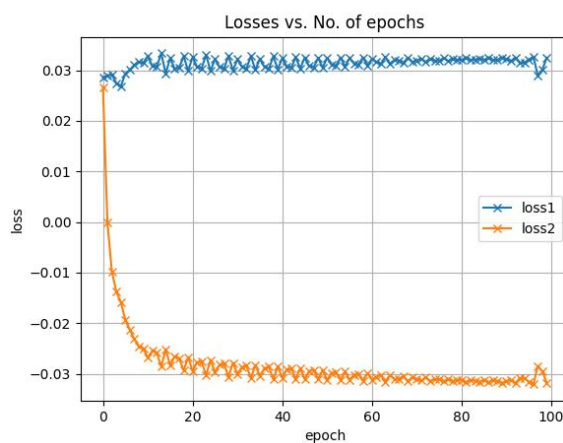
运行 `git clone https://github.com/manigalati/usad.git`

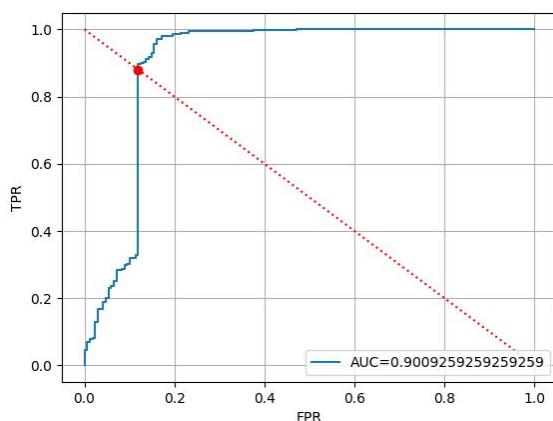
运行 `pip install -r requirements.txt` 下载依赖

修改 USAD.ipynb Attack 数据为 `attack0.csv` , Normal 数据为 `normal0.csv`

3. 实验结果

jupyter notebook 下运行 USAD.ipynb 获得以下结果，分别为损失率变化图像
和准确率变化图像：





从损失率变化图像可见，loss1（对应模型中某一自动编码器或判别环节损失）在训练初期有小幅波动后，迅速趋于稳定（维持在 0.03 附近），说明模型对正常数据的重构能力快速收敛，能稳定学习数据特征；loss2（另一环节损失，如对抗训练中判别器或重构约束损失）则呈现明显下降趋势，从初始 0.03 左右逐步降低至 -0.03 区间并保持平稳。这一曲线特征契合 USAD “双解码器对抗训练”设计：loss1 保障基础重构稳定性，loss2 通过对抗博弈优化模型对异常的辨别力，最终两损失协同收敛，证明模型训练流程完整、对抗机制有效。

准确率变化图像（以 TPR，即真正例率为核心指标）中，蓝色曲线快速攀升并趋近于 1.0，说明随训练推进，模型对异常数据的识别能力显著提升；红色虚线（理想随机猜测线）与蓝色曲线的交点（红点）可作为模型 “有效收敛节点” 参考——交点前模型快速学习，交点后进入稳定高识别阶段。结合曲线形态，USAD 在实验数据集上展现出优秀的异常检测精度，能够高效区分正常与异常时序模式，验证了算法在无监督异常检测场景的实用性。

七、总结与展望

1. 总结

本实验成功构建了基于 Online Boutique 微服务系统的故障检测与复现实验平台，借助 ChaosMesh、Prometheus+Grafana、Selenium 与 JMeter 等工具链，完成了从系统部署、功能与性能测试、故障注入到异常数据采集分析及算法复现的全流程实验。

在系统部署与测试方面，Online Boutique 微服务系统作为 Google 提供的基准演示项目，其 12 个容器化微服务组件在 Kubernetes 编排下，实现了用户浏览商品、添加购物车、结算、支付及接收通知的完整业务流程。Selenium 自动化功能测试验证了系统在移动设备布局、弱网环境下的基础功能可用性与一定的容错性；JMeter 性能测试则表明系统功能稳定，但在高并发下存在响应时间上升、波动大等性能优化空间。

故障注入与监控实验中，ChaosMesh 成功实现了对随机 Pod 的故障注入，Prometheus+Grafana 实时监控显示，模拟 Pod 意外终止的场景，故障注入引发了 CPU 短时冲高、磁盘写入突发、I/O 耗时增加等资源波动，但内存、网络带宽等指标处于安全阈值内，说明系统具备基础韧性，不过 I/O 和 CPU 短时高峰问题仍需针对性优化。

异常数据采集与分析环节，通过复现 KDD20 - UASD 论文中的 USAD 算法，

对正常与异常状态数据进行处理与训练。实验结果显示，模型的 `loss1` 和 `loss2` 协同收敛，`TPR` 指标快速提升并趋近于 `1.0`，证明 `USAD` 算法在无监督异常检测场景下具有良好的实用性，能够有效区分正常与异常时序模式。

2. 展望

故障注入扩展：本次实验主要进行了 `Pod Kill` 故障注入，未来可拓展故障类型，如模拟网络延迟、带宽限制、内存泄漏、服务依赖故障等，以更全面地评估系统在多种异常场景下的容错能力和稳定性，进一步挖掘系统的潜在缺陷。


算法优化与融合：`USAD` 算法在本次实验中取得了较好的检测效果，但仍可尝试结合其他异常检测算法，如孤立森林、变分自编码器等，通过集成学习或特征融合的方式提高检测的准确性和鲁棒性。同时，进一步优化算法参数，动态调整 α 和 β 的比例，平衡检测的灵敏度与精确性，以适应不同场景下的异常检测需求。

实验平台应用拓展：将当前构建的实验平台应用于更多实际微服务场景，与不同行业的业务系统相结合，开展更广泛的测试和验证，积累更多样化的数据，验证实验平台和检测算法在复杂实际环境中的通用性和有效性，为微服务系统的故障检测与管理提供更具普适性的解决方案。

八、附录

- 实验分工：石彬辰，高文康：微服务的部署，测试和数据采集
- 蔚佳明：实验文档撰写
- 张婧怡：论文算法复现
- 陈雨佳： PPT 制作

附 github 源码地址：
<https://github.com/chen4546/KDD20-USAD-SoftwareTest/blob/master>

 chen4546	update readme	0c9c44e · 1 hour ago	🕒 15 Commits
📁 Online-Boutique	update online-Boutique		3 days ago
📁 Online-Boutique_test	update README		2 days ago
📁 chaosMesh	update files about minikube		3 days ago
📁 data	update files about minikube		3 days ago
📁 docs	add ppt and video		1 hour ago
📁 manifests-monitoring	update files about minikube		3 days ago
📁 usad	update dataset		2 days ago
📄 .gitignore	update dataset		2 days ago
📄 README.md	update readme		1 hour ago
📄 requirements.txt	update files about minikube		3 days ago