哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

中期项目总结

——proj120:实现智能的操作系统异常检测

项目成员

姓名	年级	联系方式（QQ）
于伯淳（队长）	大二	1978689494
满洋	大二	2579504636
李怡凯	大三	1351855206

指导教师：夏文、李诗逸

完成进度

本项目的主要目标如下：

1. 实现进程级系统调用和占用资源（CPU、内存、网络）的采集；

2. 对采集到的数据进行处理，从中检测到操作系统可能发生的异常（CPU 冲高、内存泄漏、网络流量异常等）；

3. 集成不同数据处理算法，使项目能够有较强的异常检测能力；

目前，我们的赛题完成度如下：

目标编号	完成情况	说明
1	基本完成（约 90%）	能够以 csv 格式，按时序记录目标进程的四种关键数据。
2	大致完成（约 70%）	对内存异常分配和 CPU 冲高已经有了较好的检测能力。
3	大致完成（约 70%）	目前采用了聚类算法和 lstm 神经网络算法。
总计	大致完成（约 70%）	算法部分还有较大优化空间。

目录

1 概述 3

1.1. 项目背景及意义 3

1.2. 环境搭建和工具选择 3

1.3. 项目模块简述 4

1.3.1 数据收集模块 4

1.3.2 算法模块 6

基于 PyOD 的检测算法 7

基于 lstm 的检测算法 8

方案的特点 12

1.4. 项目工作流程简述 12

2 项目进展 12

2.1 与导师的沟通 12

2.2 开发状态与需要得到的帮助 13

2.3 总结与展望 13

1 概述

1.1. 项目背景及意义

随着云技术的飞速发展，云系统的复杂性和规模不断增加，云系统的稳定性受到了极大挑战。在日常的业务流程中，操作系统可能会出现各种异常现象，如 CPU 冲高、内存泄漏、网络流量异常等。

为了使运维人员能够及时发现操作系统异常，定位到相关进程，并获取异常数据，构建智能化、自动化的操作系统异常检测平台已经成为重要的工作。

本项目旨在从系统指标（CPU 占用率、内存分配、网络流量）的时间序列中分析并检测出可能存在的异常情况，并将检测结果通过日志、图片等形式呈现，为运维人员掌控系统异常情况、分析解决异常提供可靠参考。

1.2. 环境搭建和工具选择

本项目目前基于以下环境进行开发与测试：

- 操作系统：Ubuntu 20.04
- 内核版本：5.4.0
- CPU 架构：x64

本项目目前使用的工具：

- Python 3.8.10
- bcc release 0.24.0
- pyod 1.0.0
- pytorch 1.11.0
- onnx 1.11.0
- onnxruntime 1.11.1

1.3. 项目模块简述

当前本项目主要分为下面两个模块：

1. 3. 1 数据收集模块

此模块主要基于 eBPF 技术的 python 框架 bcc。eBPF 是一项革命性技术，起源于 Linux 内核，可以在特权上下文（如操作系统内核）中运行沙盒程序。它用于安全有效地扩展内核的功能，而无需更改内核源代码或加载内核。eBPF 原理图如图 1 所示。

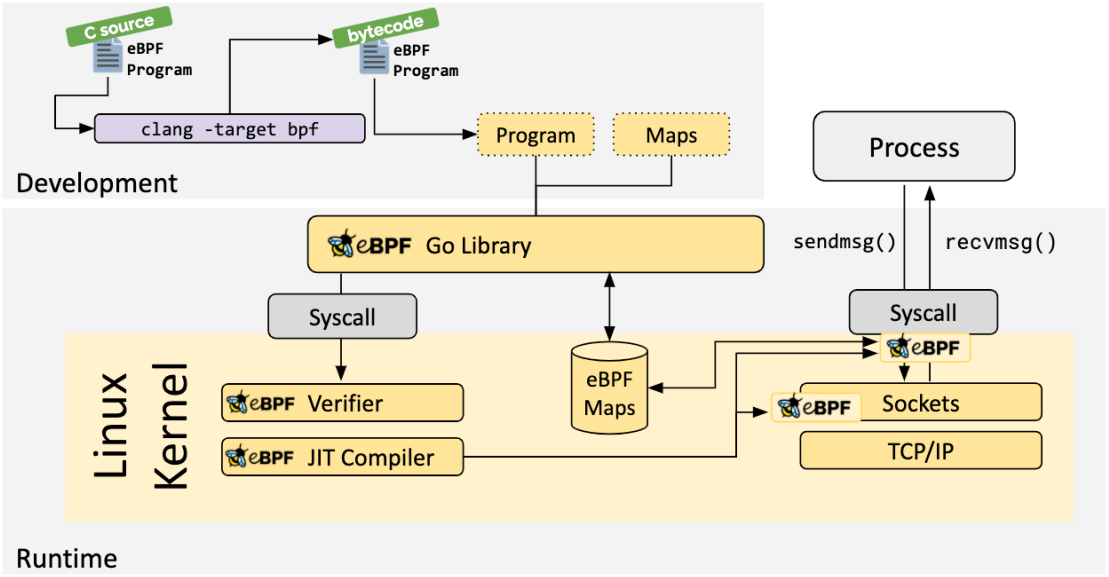

图 1 eBPF 原理图

bcc 是一个用于创建高效内核跟踪和操作程序的工具包，其使用 C 中的内核工具（包括围绕 LLVM 的 C 包装器），以及 python 和 lua 中的前端，使得 eBPF 程序更加容易编写，现已被应用于许多任务，包括性能分析与网络流量控制。

目前本模块由一个顶层模块和四个子模块组成，模块结构如图 2 所示：

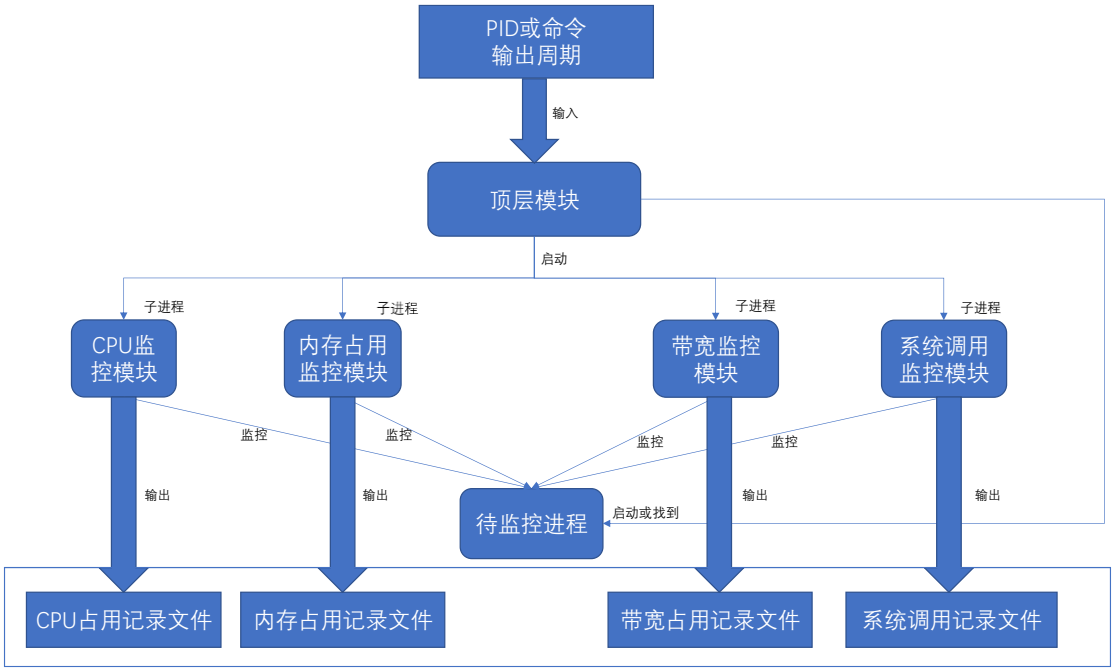

图 2 数据收集模块结构示意

其顶层模块负责处理命令参数，并启动子模块进程进行对目标进程的资源占用、系统调用的监控。四个子模块分别实现 CPU 占用、内存占用、带宽占用、系统调用的监控功能，并通过 csv 文件记录各资源指标的时间序列。

输入参数处理：模块支持参数如图 3 所示。

```
usage: top_snoop.py [-h] [-p PID] [-c COMMAND] [-i INTERVAL]

Attach to process and snoop its resource usage

optional arguments:
  -h, --help            show this help message and exit
  -p PID, --pid PID     id of the process to trace (optional)
  -c COMMAND, --command COMMAND
                        execute and trace the specified command (optional)
  -i INTERVAL, --interval INTERVAL
                        The interval of snoop (unit:s)

EXAMPLES:
    ./top_snoop -c './snoop_program' # Run the program snoop_program and snoop its resource usage
    ./top_snoop -p 12345  # Snoop the process with pid 12345
    ./top_snoop -p 12345 -i 1  # Snoop the process with pid 12345 and output every 1 second
```

图 3 监控模块支持参数

顶层模块通过使用 Python 程序库 argparse 可以很容易从输入命令中获取各个参数字段。

启动子进程进行监控：在完成对输入参数的处理后，顶层模块会启动四个子进程，分别对应四个监控子模块，用于对指定进程进行监控记录。

1.3.2 算法模块

此模块负责对采集到的 csv 文件进行处理，并通过选取的异常检测算法进行异常的检出。

操作系统的参数实际上是具有连续性的时间序列参数，因此时序数据分析的很多方法都可以在异常检测中运用。

常见的异常类型有如下几种：

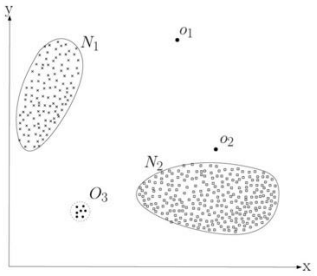

Fig. 1. A simple example of anomalies in a two-dimensional data set.

图 4 点异常

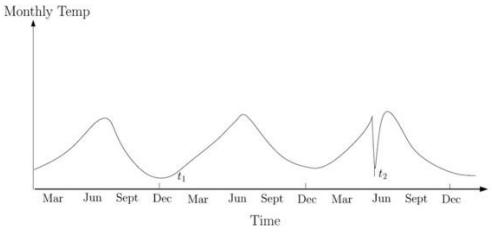

Fig. 3. Contextual anomaly t_2 in a temperature time-series. Note that the temperature at time t_1 is same as that at time t_2 but occurs in a different context and hence is not considered as an anomaly.

图 5 上下文异常

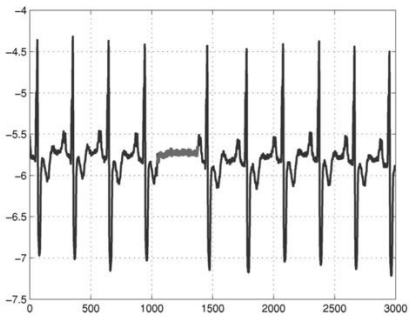

Fig. 4. Collective anomaly corresponding to an *Atrial Premature Contraction* in an human electrocardiogram output.

图 6 集合异常

基于 PyOD 的检测算法

PyOD 是 python 的一个异常检测工具库，集成了 kNN/LOF/LOCI/ABOD 等多种异常检测算法，支持多种 python 版本和操作系统，可以对采集到的数据进行初步的异常检测。

Method	Category	JIT Enabled	Multi-core
LOF (Breunig et al., 2000)	Proximity	No	Yes
kNN (Ramaswamy et al., 2000)	Proximity	No	Yes
AvgkNN (Angiulli and Pizzuti, 2002)	Proximity	No	Yes
CBLOF (He et al., 2003)	Proximity	Yes	No
OCSVM (Ma and Perkins, 2003)	Linear Model	No	No
LOCI (Papadimitriou et al., 2003)	Proximity	Yes	No
PCA (Shyu et al., 2003)	Linear Model	No	No
MCD (Hardin and Rocke, 2004)	Linear Model	No	No
Feature Bagging (Lazarevic and Kumar, 2005)	Ensembling	No	Yes
ABOD (Kriegel et al., 2008)	Proximity	Yes	No
Isolation Forest (Liu et al., 2008)	Ensembling	No	Yes
HBOS (Goldstein and Dengel, 2012)	Proximity	Yes	No
SOS (Janssens et al., 2012)	Proximity	Yes	No
AutoEncoder (Sakurada and Yairi, 2014)	Neural Net	Yes	No
AOM (Aggarwal and Sathe, 2015)	Ensembling	No	No
MOA (Aggarwal and Sathe, 2015)	Ensembling	No	No
SO-GAAL (Liu et al., 2018)	Neural Net	No	No
MO-GAAL (Liu et al., 2018)	Neural Net	No	No
XGBOD (Zhao and Hryniewicki, 2018b)	Ensembling	No	Yes
LSCP (Zhao et al., 2019)	Ensembling	No	No

图 7 PyOD 集成的异常检测算法

目前本项目主要采用了 PyOD 中基于近邻的几种算法，包括 kNN、LOF、CBLOF 等，可以通过 pandas 工具对读入的 csv 文件进行处理，利用 PyOD 中已集成的算法对处理后的数据进行训练，并对原始数据进行打分。

```
clf = KNN(method='mean')

x = npy.array(df['ticks']).reshape(-1, 1)
y = npy.array(df[' size(B)']).reshape(-1, 1)
train_data = npy.concatenate((x, y),axis=1)

clf.fit(train_data)

train_pred = clf.labels_
train_scores = clf.decision_scores_
```

图 8 PyOD 进行内存监控数据训练的部分代码

通过设定分数阈值，可以在原始数据中标记出可能存在异常的数据点，并通

过 matplotlib 将训练结果可视化呈现：

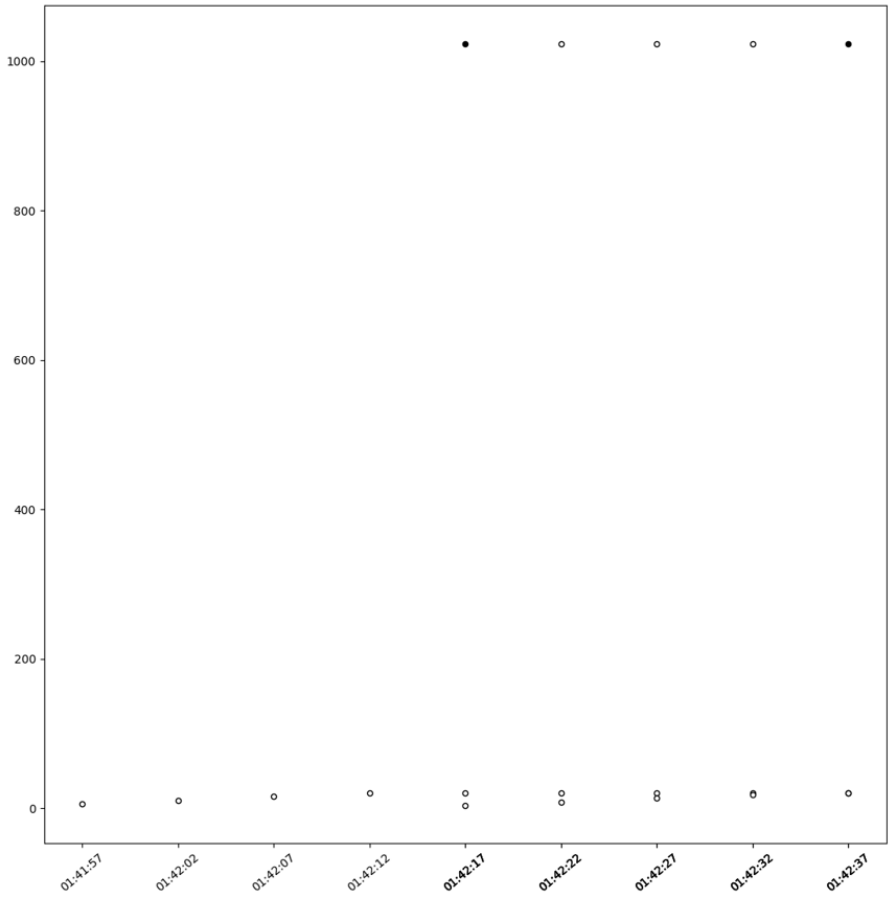

图 9 PyOD 对内存数据进行检测的结果

上图中，黑色数据点被认为是异常数据。

基于 lstm 的检测算法

根据业界前沿研究动态，深度学习的方法在大量数据的工业场景下具有准确性高，可迁移性好的优点，因此我们还提出了一种基于深度学习的操作系统异常检测流程。

根据深度学习的常用流程，我们也将流程分为两部分：训练和推理。其中训练部分可以离线完成，可以根据实际情况和实际数据对模型结构或训练参数进行调整，有利于提高模型的表现。而推理部分是设计在真实工业场景中运行，参数不会频繁改变，因此采用不同的模型部署方法，达到减少依赖，提高性能的目的。

训练阶段

训练阶段可以采用任意的数据，甚至是非操作系统中收集的数据，只要能够代表无异常的数据，体现合理数据的特征，就可以用于训练。训练模型也可以使用新兴的预训练-微调模式，也可以利用迁移学习等深度学习技术提高模型的通用性。

在我们的项目中，模型结构使用了简单的两层 LSTM+线性分类器，能够作为深度学习的异常检测方法的代表。

我们在训练阶段使用业界常用的 PyTorch 框架进行模型搭建和训练。

训练原理

工业场景中，常常没有可供训练的有标注数据，大多数情况下能够获得的是大量无标注的正常数据，所以无监督训练的实际应用场景更广泛。

在我们的系统中，使用无监督的训练模式，使用时序数据的一段窗口内的数据作为输入，训练模型预测之后一段时间的数据。模型结构：

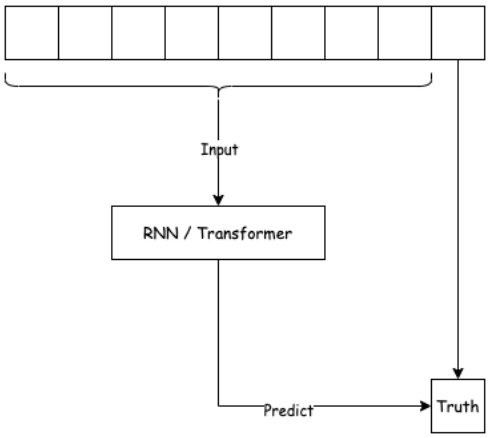

图 10 训练模型结构（1）

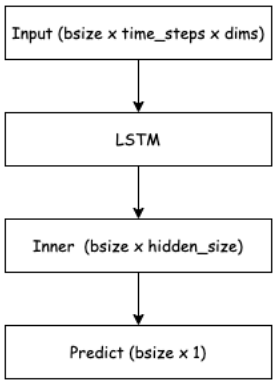

图 10 训练模型结构（2）

上图中 Batch size 是可变的，在训练时使用较大的 batchsize 可以提高训练速度，在预测时通常使用的 Batch size 为 1。而 time_steps 是选取的窗口大小，这个参数需要选取合适。Dims 则是检测变量的维度，比如同时对 CPU、内存和网络进行监控，那么 dims 就设置为 3，如果有更多的数据那么也可以灵活地设置成更大的值。这种方法可以容易的扩展到多维度的数据，也可以扩展预测的范围以获得更灵敏的检测。

推理阶段

推理阶段需要完成的任务仅仅是对已有的参数进行矩阵运算，因此并不需要使用 PyTorch 这样重型的依赖。我们采用 OnnxRuntime 对模型的进行部署，OnnxRuntime 是一个轻量化的推理框架，可以支持多种平台，在 arm 架构或嵌入式设备的 NPU 中也可以运行。

下图是 OnnxRuntime 的支持矩阵

Optimize Inferencing	Optimize Training					
Platform	Windows	Linux	Mac	Android	iOS	Web Browser (Preview)
API	Python	C++	C#	C	Java	JS	Obj-C	WinRT
Architecture	X64	X86	ARM64	ARM32	IBM Power	
Hardware Acceleration	Default CPU	CoreML	CUDA	DirectML	oneDNN	
	OpenVINO	TensorRT	NNAPI	ACL (Preview)	ArmNN (Preview)	
	MIGraphX (Preview)	TVM (Preview)	Rockchip NPU (Preview)	Vitis AI (Preview)		

图 11 OnnxRuntime 的支持矩阵

异常的判定

同样从预测样本的数据中抽取一个窗口期的数据，根据模型的预测数据和一个可调的阈值，判定下一段时间内的数据是否异常。

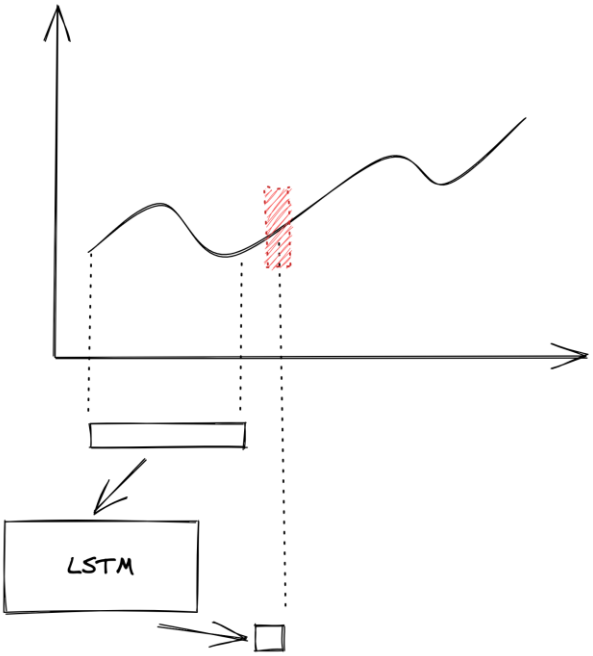

图 12 异常判定窗口示意

目前实现了基于 LSTM 的模型结构，然后根据 80 个样本的窗口预测下一个样本的情况，如果偏差超过阈值 100，那么认为这个样本是异常点。

方案的特点

我们设计的流程具有高度的灵活性，无论是模型的结构还是各种超参数都可以灵活地调整，可以针对不同系统所需的准确度和性能进行合适的定制。例如在嵌入式系统上运行的异常检测可能作为数据的初步筛选，并不需要很高的准确度，但是嵌入式系统的算力有限，因此可以使用简单的 LSTM 和低阈值进行筛选，将可能的异常信息上报后再由中心化的服务器使用 Transformer 类的大型模型对这些信息进行检测，最终确认异常的信息上报给管理人员。

1.4. 项目工作流程简述

目前本项目通过顶层 main.py 模块进行各模块的集成调度。各模块的独立性很高，数据收集部分和算法部分的组合可以根据实际场景进行很大程度的定制。例如，可以自由设定数据收集模块的监控周期，并定期调用算法模块进行一个时段的数据处理，将数据备份并进入下一阶段的监控，运维人员可以根据每个周期的数据进行该时段的异常分析工作。

2 项目进展

2.1 与导师的沟通

本赛题由华为方面的高若姝和陈东辉导师负责，目前已经通过邮件与两位导师取得了联系，并建立了微信群进行沟通。导师目前在比赛的相关具体指标和所用工具方面进行了指导。

2.2 开发状态与需要得到的帮助

目前本项目的数据采集模块已经基本开发完毕，基于 PyOD 的算法模块目前结构功能还比较单一，只能进行初步的异常监测，对大量数据的情况处理能力较差；基于 lstm 的算法部分可以实现大数据量的异常处理，目前还需要进一步封装和优化。

目前为止，需要得到的帮助主要体现在异常监测的真实案例方面，希望能够提供一些异常注入的工具和真实场景，以便对已经开发实现的功能进行针对性的优化。

2.3 总结与展望

下一阶段的工作主要是以下方面：

1. 进一步提高项目的封装程度，以便提供完整的异常监控方案；
2. 通过对真实案例的测试监控，对某些特定种类的异常进行算法部分的针对性优化；
3. 考虑整个项目的可移植性，考察项目在不同操作系统或内核版本下的兼容性。

我们希望能够在这些方面进一步优化、完善本项目，使其具有真实场景下的使用价值。