\Orchestrating a brighter world NEC

April 5th, 2022 - ACM EuroMLSys 2022

syslrn: Learning What to Monitor for Efficient Anomaly Detection

Davide Sanvito, Giuseppe Siracusano, Sharan Santhanam, Roberto Gonzalez, Roberto Bifulco

NEC Laboratories Europe

System monitoring based on logs

- Monitoring software behaviour is a critical task in any operational system deployment
- Logs track application state and

```
[req-✶] [instance: ✶] Attempting claim on node ✶: memory ✶ MB, disk ✶ GB, vcpus ✶ CPU
```

```
TIMESTAMP               PID   VERB COMPONENT            LOG MESSAGE
2021-11-25 19:49:51.479 22191 INFO nova.compute.claims [req-dfdc8879-1710-44db-9acb-00927348ce05 ...] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] Attempting claim on node c431: memory 512 MB, disk 1 GB, vcpus 1 CPU
2021-11-25 19:49:51.479 22191 INFO nova.compute.claims [req-dfdc8879-1710-44db-9acb-00927348ce05 ...] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] Total memory: 32010 MB, used: 512.00 MB
[...]
2021-11-25 19:49:51.481 22191 INFO nova.compute.claims [req-dfdc8879-1710-44db-9acb-00927348ce05 ...] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] Claim successful on node c431
2021-11-25 19:49:55.799 22191 INFO nova.compute.manager [-] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] VM Started (Lifecycle Event)
2021-11-25 19:49:55.827 22191 INFO nova.compute.manager [req-acdc7c48-a118-43a7-90e4-cfbc870b8c2f - - - - -] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] VM Paused (Lifecycle Event)
2021-11-25 19:49:57.265 22191 INFO nova.compute.manager [req-acdc7c48-a118-43a7-90e4-cfbc870b8c2f - - - - -] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] VM Resumed (Lifecycle Event)
2021-11-25 19:57:15.231 22191 INFO nova.compute.manager [req-2684c5a4-30a7-4a5a-93d2-82929bb0a3e8 ...] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] Attaching volume f194c1ef-fca0-4a36-8962-9d9ea8b06fbe to /dev/vdb
[...]
2021-11-25 20:04:47.705 22191 INFO nova.compute.manager [-] [instance: f6f318e3-3922-46d1-96df-3013a32acb77] VM Stopped (Lifecycle Event)
```

```
[req-✶] [instance: ✶] Attaching volume ✶ to ✶
```

- Tipical steps for a log-based Anomay Detection system
  - Correlation
  - Parsing
  - Anomaly Detection (AD)
- Observations
  - Need app-specific knowledge, not re-usable
  - Limited by *when* and *what* an application logs

Provenance Graphs

- Graph capturing the relationships across OS-level entities
- Based on monitoring of OS events (e.g. syscalls)

- Observations
  - Types and number of processes and their relationships disclose relevant information on the application
  - Mostly used for security-critical services and for offline analysis
  - Failure detection might require the monitoring of a smaller set of events

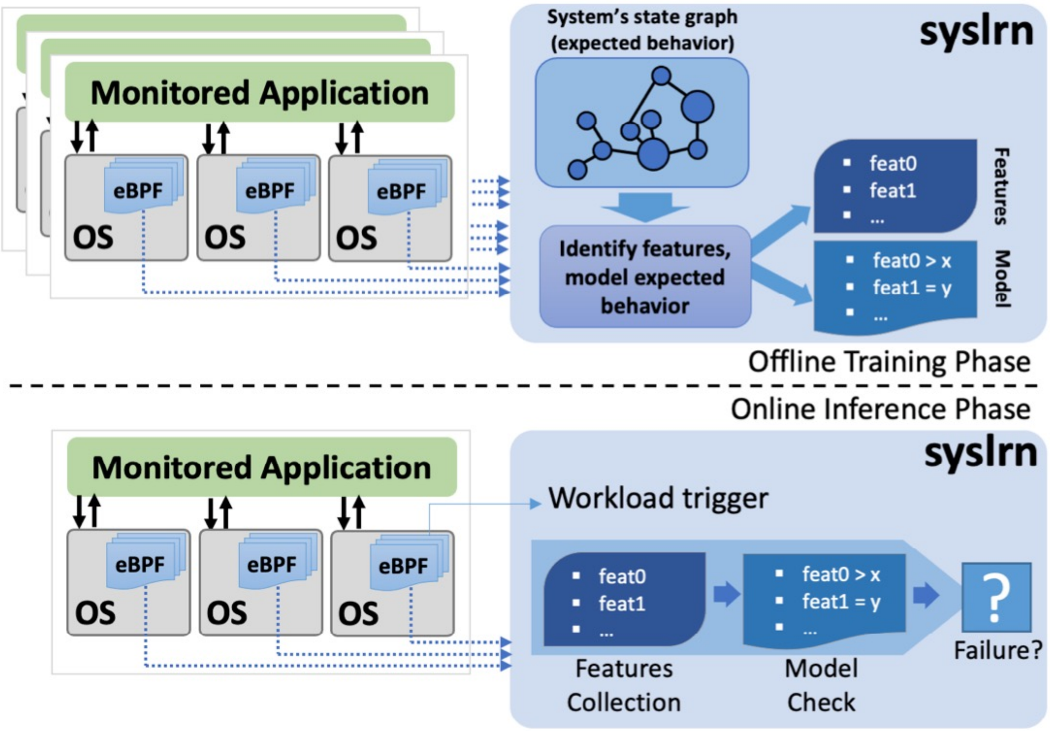
syslrn

- Complement these approaches with an alternative
  - Little domain knowledge
  - Independent from software developer practices
  - Lightweight enough to be deployed in high performance scenarios

- High-level design
  - **Offline phase**: *detailed* monitoring to identify key indicators of normal behaviour
  - **Online phase**: *lightweight* monitoring to verify the correct behaviour

Offline vs Online phases

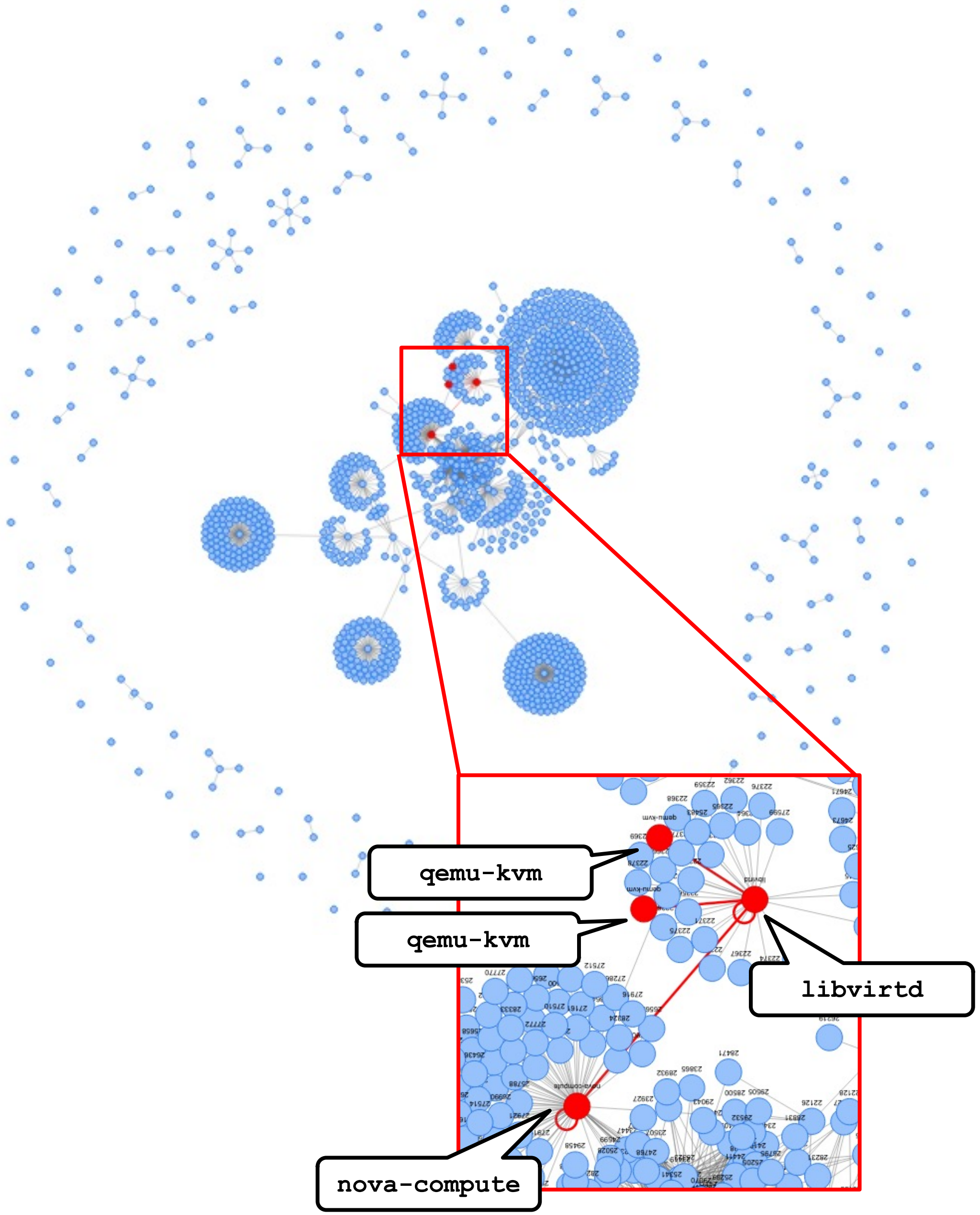
- **Offline phase**
  - Build complete system behaviour graph
  - Run graph analysis methods to identify relevant features
  - Derive a model for normal behaviour
- **Online phase**
  - Monitoring based on Linux eBPF
  - Collect only relevant features
  - Driven by monitoring application's external interfaces
  - Perform Anomaly Detection



- **In this paper:**
  - Initial prototype of syslrn
  - Graph analysis method: heuristic based on bag-of-components graph embedding and linear regression
  - Tested with an use case based on OpenStack

Case study: OpenStack

- Complex cloud management system
  - Several modules (e.g. compute, networking, storage, etc)
  - Interactions across modules and with third-party software
- Instrumented testbed
  - Common OpenStack operations
  - Injection of realistic failures based on [1], extended to support multiple workloads
- Application graph example
  - System background processes
  - Application background processes
  - ➡ Processes related to the handling of the service requests



[1] D. Cotroneo, L. De Simone, P. Liguori, R. Natella, N. Bidokhti - *How Bad Can a Bug Get? An Empirical Analysis of Software Failures in the OpenStack Cloud Computing Platform* [ACM ESEC/FSE 2019]

Case study: OpenStack (2)

Offline phase

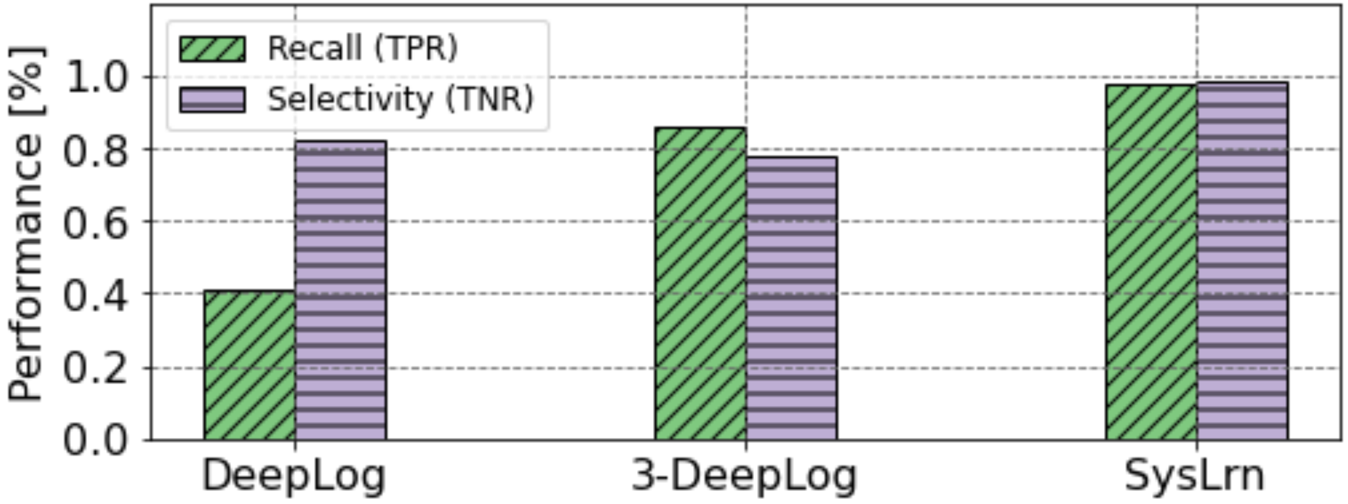
- Bag-of-nodes graph embedding: two types of features, instance counter and relationships counter
- Normal behaviour model: analyze the relationship between the features of the graph embeddings and the number of service requests received using an heuristic
  - Fit a Linear Regression (LR) model for each feature of the embedding
  - Selects a subset of features based on a goodness-of-fit measure
- Features backtracking: map selected features to OS primitives required to monitor them

Online phase

- Collect selected features using eBPF programs
- Anomaly Detection periodically triggered to check them against the model of normal behaviour
  - Based on an ensemble of LR models

Evaluation

- **Baseline**
  - DeepLog
  - 3-DeepLog
- **Dataset**
  - 900+ experiments: failure free (FF) or single failure point in one OpenStack component
  - One or more homogeneous workloads per experiment
  - Collection of both application logs and OS-level events
- **Models**
  - Training on FF data only
  - Testing on FF and failures
- **Metrics**
  - Recall (TPR)
  - Selectivity (TNR)



Monitoring overhead

- We investigated the overhead of running OS-level feature extraction with eBPF
- Benchmark based on Redis, a high performance key-value store that heavily relies on communication
  - OpenStack VM generation workload is unsuitable to perform a stress test
- `redis-benchmark` tool
  - 50 concurrent clients
  - No connection keep-alive
- When logs are not required, for some performance critical deployment syslrn may provide a more efficient monitoring alternative

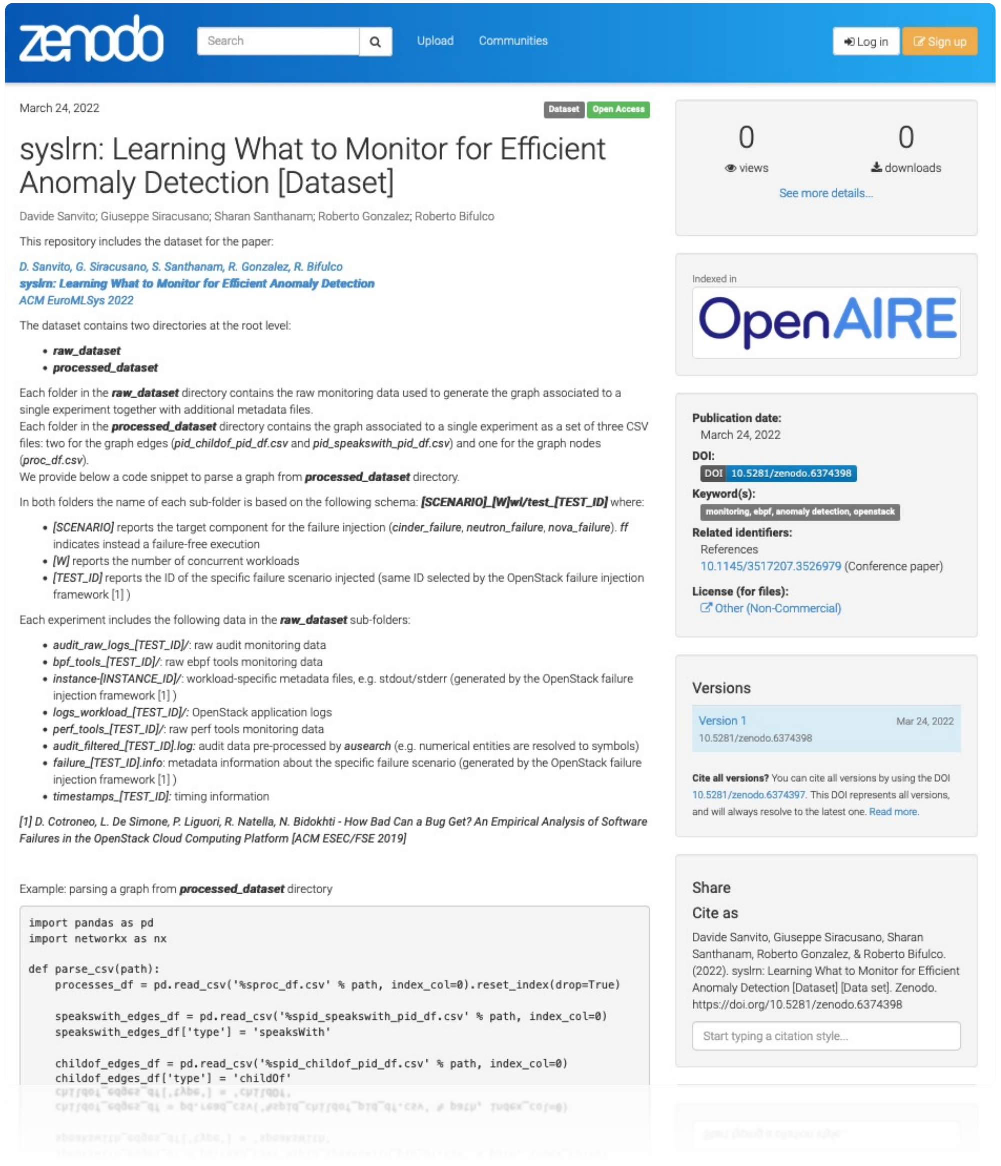
Operation	Baseline (no mon)	Log-based monitoring	eBPF program (w/ user code)
SET	48.8k	17.2k (-64.73%)	47.4k (-2.78%)
GET	48.3k	17.8k (-63.43%)	47.1k (-2.61%)
LPUSH	48.6k	17.2k (-64.51%)	47.4k (-2.36%)
LPOP	49.7k	17.1k (-65.63%)	48.6k (-2.19%)

redis-server throughput in req/s

Conclusion

- Initial prototype of syslrn
  - Minimal set of functionalities
  - Preliminary evaluation and deployment models
    - Single use case with simplified subset of workloads
    - Simplifying assumption (e.g. timing of features collection and anomaly detection)
- Next steps
  - Evaluation on larger set of applications to investigate benefits and limitations
  - Extend syslrn with multiple graph representation and normal behaviour modeling methods

Dataset available


zenodo | Search | Upload | Communities | Log in | Sign up

March 24, 2022 | Dataset | Open Access

syslrn: Learning What to Monitor for Efficient Anomaly Detection [Dataset]

Davide Sanvito; Giuseppe Siracusano; Sharan Santhanam; Roberto Gonzalez; Roberto Bifulco

This repository includes the dataset for the paper:

D. Sanvito, G. Siracusano, S. Santhanam, R. Gonzalez, R. Bifulco
syslrn: Learning What to Monitor for Efficient Anomaly Detection
ACM EuroMLSys 2022

The dataset contains two directories at the root level:

- ***raw_dataset***
- ***processed_dataset***

Each folder in the ***raw_dataset*** directory contains the raw monitoring data used to generate the graph associated to a single experiment together with additional metadata files.
Each folder in the ***processed_dataset*** directory contains the graph associated to a single experiment as a set of three CSV files: two for the graph edges (*pid_childof_pid_df.csv* and *pid_speakswith_pid_df.csv*) and one for the graph nodes (*proc_df.csv*).
We provide below a code snippet to parse a graph from ***processed_dataset*** directory.

In both folders the name of each sub-folder is based on the following schema: ***[SCENARIO]_[W]wl/test_[TEST_ID]*** where:

- *[SCENARIO]* reports the target component for the failure injection (*cinder_failure, neutron_failure, nova_failure*). *ff* indicates instead a failure-free execution
- *[W]* reports the number of concurrent workloads
- *[TEST_ID]* reports the ID of the specific failure scenario injected (same ID selected by the OpenStack failure injection framework [1])

Each experiment includes the following data in the ***raw_dataset*** sub-folders:

- *audit_raw_logs_[TEST_ID]/*: raw audit monitoring data
- *bpf_tools_[TEST_ID]/*: raw ebpf tools monitoring data
- *instance-[INSTANCE_ID]/*: workload-specific metadata files, e.g. stdout/stderr (generated by the OpenStack failure injection framework [1])
- *logs_workload_[TEST_ID]/*: OpenStack application logs
- *perf_tools_[TEST_ID]/*: raw perf tools monitoring data
- *audit_filtered_[TEST_ID].log*: audit data pre-processed by ***ausearch*** (e.g. numerical entities are resolved to symbols)
- *failure_[TEST_ID].info*: metadata information about the specific failure scenario (generated by the OpenStack failure injection framework [1])
- *timestamps_[TEST_ID]*: timing information

[1] D. Cotroneo, L. De Simone, P. Liguori, R. Natella, N. Bidokhti - How Bad Can a Bug Get? An Empirical Analysis of Software Failures in the OpenStack Cloud Computing Platform [ACM ESEC/FSE 2019]

Example: parsing a graph from ***processed_dataset*** directory

```
import pandas as pd
import networkx as nx

def parse_csv(path):
    processes_df = pd.read_csv('%sproc_df.csv' % path, index_col=0).reset_index(drop=True)

    speakswith_edges_df = pd.read_csv('%spid_speakswith_pid_df.csv' % path, index_col=0)
    speakswith_edges_df['type'] = 'speaksWith'

    childof_edges_df = pd.read_csv('%spid_childof_pid_df.csv' % path, index_col=0)
    childof_edges_df['type'] = 'childOf'
```

0 views | 0 downloads | See more details...

Indexed in OpenAIRE

Publication date: March 24, 2022
DOI: 10.5281/zenodo.6374398
Keyword(s): monitoring, ebpf, anomaly detection, openstack
Related identifiers: References 10.1145/3517207.3526979 (Conference paper)
License (for files): Other (Non-Commercial)

Versions
Version 1 — Mar 24, 2022 — 10.5281/zenodo.6374398

Cite all versions? You can cite all versions by using the DOI 10.5281/zenodo.6374397. This DOI represents all versions, and will always resolve to the latest one. Read more.

Share
Cite as
Davide Sanvito, Giuseppe Siracusano, Sharan Santhanam, Roberto Gonzalez, & Roberto Bifulco. (2022). syslrn: Learning What to Monitor for Efficient Anomaly Detection [Dataset] [Data set]. Zenodo. https://doi.org/10.5281/zenodo.6374398

Start typing a citation style...

- Pre-processed graph data
- Raw monitoring data
  - eBPF monitoring data
  - Linux Audit monitoring data
  - OpenStack application logs

GitHub https://github.com/nec-research/syslrn-EuroMLSys22

zenodo https://zenodo.org/record/6374398

SCAN ME

Thank you!

Davide.Sanvito@neclab.eu

\Orchestrating a brighter world

NEC