Multi-source Distributed System Data for AI-Powered Analytics

Sasho Nedelkoski, Jasmin Bogatinovski, Ajay Kumar Mandapati, Soeren Becker, Jorge Cardoso, Odej Kao

Multi-Source Distributed System Data for AI-powered Analytics

Sasho Nedelkoski[1], Jasmin Bogatinovski[1], Ajay Kumar Mandapati[1], Soeren Becker[1], Jorge Cardoso[2,3], and Odej Kao[1]

[1] Distributed Systems, Technische Universität Berlin, Berlin, Germany
nedelkoski, firstname.lastname@tu-berlin.de
[2] Huawei Munich Research Center, Munich, Germany
[3] Department of Informatics Engineering/CISUC, University of Coimbra, Portugal
jorge.cardoso@huawei.com

Abstract. The emerging field of Artificial Intelligence for IT Operations (AIOps) utilizes monitoring data, big data platforms and machine learning to automate operations and maintenance (O&M) tasks in complex IT systems. The available research data usually contain only a single source of information, often logs or metrics. The inability of the single-source data to describe the precise state of the distributed systems leads to methods that fail to make effective use of the joint information, thus, producing a large number of false predictions. Therefore, current data limits the possibilities for greater advances in AIOps research. To overcome these constraints, we created a complex distributed system testbed, which generates multi-source data composed of distributed traces, application logs, and metrics. This paper provides detailed descriptions of the infrastructure, testbed, and the experiments for the generated data. Furthermore, it identifies how this data can be utilized as a stepping stone for the development of novel methods for O&M tasks such as anomaly detection, root cause analysis, and remediation.
The data from the testbed and its code is available at https://zenodo.org/record/3549604.

Keywords: aiops · distributed system · dataset · tracing · metrics · logs · anomaly detection · root-cause analysis

1 Introduction

AIOps refers to multi-layered technology platforms that automate and enhance IT operations by using analytics and machine learning [5]. AIOps was introduced to reduce the cost and increase the effectiveness of O&M tasks on ever-increasing complex public, private, edge, mobile, and hybrid cloud environments. The transition from mainframes, to virtual machines, to containers, and serverless computing made existing approaches and tools which rely on simple statistical methods obsolete due to the increasing complexity and communication patterns between services. Notable examples include Zabbix, Cacti, and Nagios [13, 21].

Monitoring data is a key element of new AIOps tools and one of the cornerstones of research. The data generated by distributed IT systems can be classified into three main categories: metrics, application logs, and distributed traces [19]. *Metrics* are numeric values measured over some time. They describe the utilization and status of the infrastructure, typically regarding CPU, memory, disk, network throughput, and service call latency. *Application logs* enable developers to record what actions were executed at runtime by software. Service, microservices, and other systems generate logs which are composed of timestamped records with a structure and free-form text. *Distributed traces* record the workflows of services executed in response to requests, e.g., HTTP or RPC requests. The records contain information about the execution graph and performance at a (micro)service level.

Recently, various approaches – focusing on a wide range of datasets, O&M tasks, and IT systems – have been proposed. This includes a variety of tasks, which extract knowledge from a specific type of data. For example, anomaly detection has been applied to metrics (numeric) [17], logs (unstructured numeric and text data) [6, 12], and also to distributed system traces (unstructured numeric and text data) [14, 15].

The existing research has mainly explored publicly available data, which usually captures only a single data source category. This limits both the development of new methods that could extract knowledge from multi-source data and their proper evaluation. The absence of data repositories capturing the three data categories from modern distributed systems prevents the development of methods for multi-source mining, knowledge extraction, semantic information learning from the naturally linked data sources. Furthermore, enables fault detection, root-cause analysis, and remediation. These contributions can provide further advances in the field as existing approaches typically produce a large number of false positives.

We address these issues by producing the following contributions:

- A new data of metrics, logs, and traces generated by a distributed system based on microservice architecture.
- Description of the approach developed to generate the multi-source system data.
- Analysis of existing datasets utilized for the evaluation of AIOps algorithms, highlighting their benefits and their limitations.
- Applications of the multi-source data to develop new algorithms to support additional O&M tasks.

Specifically, during the development and data generation process, we adhere to the following requirements to ensure quality of the data: **(R1) originality**, **(R2) reusability**, **(R3) quality**, and **(R4) extendability**.

2 Related Work

Metrics, logs, and traces are important data sources that are fundamental to the operation of complex distributed systems. In following we study the related work for these data accordingly.

The metrics data are obtained from monitoring of the resources such as CPU, memory, disk, network throughput and latency. A plethora of available collections of datasets containing metric data can be found in Stonybrook [20]. As a collection, it has multiple datasets for different tasks related to anomaly detection. Numenta [1] predominantly stores datasets from streaming and real-time applications, while Harvard [8], ELKI [7], LMU [11] store network intrusion data.

The main challenge AIOps systems built on log data are facing, is the unstructured nature of the logs. This problem usually requires prior and proper preprocessing and/or inclusion of domain knowledge. Often, approaches extract log key identifiers for the logs and are modeling their sequences. There exist two resources of log data for cluster systems available. The CFDR resource [3] stores link to 19 log datasets grouped in 11 data collections. The datasets cover both hardware and software logs. The second resource is the loghub data resource [22]. It consists of 16 datasets describing systems spanning across distributed systems, supercomputers, operating systems, mobile systems, server applications and standalone software. The datasets cover a different period from a few days until a few months. From the perspective of the system description, these data have weakness in providing just a single aspect of the system.

In microservice architectures, traces are graph-like structures composed of events or spans [16]. The traces represent the system execution workflow, hence detailed information for individual services and the causal relationship to other related services can be inferred. Nedelkoski et al. [14,15] introduce novel anomaly detection methods for distributed tracing data. They proposed a multimodal neural network with long short-term memory (LSTM) to enable the learning from the sequential nature in the tracing data. They describe how the data is obtained, but the datasets are not publicly available. Azure Public dataset composes of two datasets representing two traces of the virtual machine of Microsoft Azure [4]. It is mostly utilized to improve resource management in large cloud platforms. Alibaba's cluster data is a collection of two datasets from real-world production [2]. Google's collection of two tracing datasets originates from parts of Google cluster management software and systems [9].

Our collection of data, describing the same system from the three perspectives of logs, metrics, and traces, to the best of our knowledge, is the first of its kind. This enables building models with diverse complementary information, hence making AIOps systems to perform better [15].

3 Dataset generator

In this section, we describe the infrastructure, experiments, workload, and the injected faults as part of the testbed for data generation. The testbed and the

generated data follow the requirements stated above, as every parameter stated in the following can be easily changed, satisfying part of **R2**, and **R4**.

3.1 Infrastructure

An OpenStack [18] testbed based on a microservice architecture that is running in a dockerized environment called Kolla-Ansible was first deployed. OpenStack is a cloud operating system that controls large pools of computing, storage, and networking resources throughout a data-centre. They are all managed and provisioned through APIs with common authentication mechanisms.

The experimental testbed setup is shown in Figure 1 and for the purpose of the generation of the data it consists of one control node named `wally-113` and four compute nodes: `wally-127`, `wally-122`, `wally-123`, and `wally-124`. It was deployed on bare-metal nodes of a cluster where each node has RAM 16GB, 3x 1TB of disks, and 2x 1Gbit Ethernet NIC. Three hard disks were combined to a software RAID 5 for data redundancy.

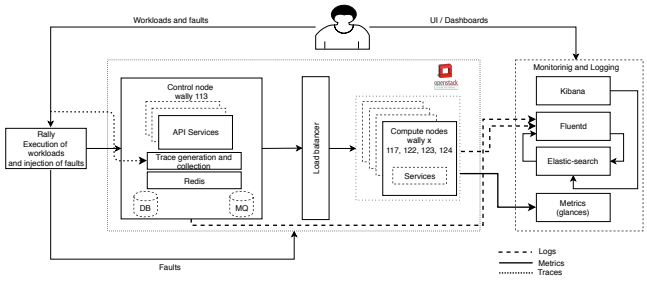

Fig. 1. Illustration of the infrastructure from where the data was generated.

3.2 Workloads and faults injected

To generate workloads and inject faults into the infrastructure we used Rally.

Rally docker image was used to create the load and inject os-faults appropriately. The listed workloads and faults in the following cover user request that is served by the main OpenStack projects:

- *Create and delete server.* We injected a compute fault which is restarting the API container that runs on the compute nodes.
- *Create and delete image.* We inject the fault in the `glance-api` running on the controller node.
- *Create and delete network.* There are various components that we focus on while injecting faults such as disrupting the below-mentioned services running in docker containers: `neutron metadata agent`, `neutron l3 agent`, `neutron dhcp agent`, `neutron openvswitch agent` and `neutron server`.

We performed two different experiments. In the first experiment, the user actions as a workload were executed in a sequential way, when one finishes then the next is started. This experiment was performed for 750, 1000, and 3000 iterations (*create and delete server*, *create and delete image*, *create and delete network*), where faults were injected every 250 iterations respectively. The fault was injected in only one iteration, however, we noticed that some of the faults take time and propagate the errors to other iterations as well. In the second experiment, the Rally workloads were concurrently executed. This experiment was performed for 2000, 3000, and 6000 iterations for *create and delete server*, *create and delete image* and *create and delete network*, respectively. The faults were injected at different rates, 250 for *create and delete server* and *create and delete image* and 500 iterations for *create and delete network*. The number of iterations for each action was chosen so that all workloads approximately finish at the same time. The data from the second experiment is slightly more suited for multi-source methods utilizing distributed log data, as it was generated with that as a goal. Also, HTML reports were collected which correlates all the events of creations, failures and fault injections. These reports serve as pseudo ground truth for the normal and anomalous state of the system.

4 Dataset Description

The workloads and faults described in the previous section were executed on the testbed. As explained, the execution generated three main categories of observability data: distributed traces, metrics, and application logs. These data were recorded in concurrently to provide the state of the system from multiple points of view, which satisfies the **R1** for originality as no such dataset exists in previous work.

4.1 Metrics

The metrics data category contains data for the 5 physical nodes in the infrastructure. The 5 files are named metrics_wally_N, where N is either the controller node or one of the compute nodes. Each of these files has 7 features: `now` (the timestamp of the recording), `cpu.user` (percentage time spent in userspace), `mem.used` (the RAM usage of the physical host), `load.cpucore` (the number of cores of the physical host), `load.min1, min5, min15` (Linux load averages are system load averages that show the running tasks demand on the system as an average number of running plus waiting threads).

4.2 Logs

The log files are distributed over the infrastructure and they are grouped in directories by the OpenStack projects (e.g., nova, neutron, glance, etc.) at the wally nodes. At each of the physical nodes, different projects are running. The

control node has more services running and thus has more log files for the OpenStack projects. We provide the raw log directories in this dataset along with the aggregated log file. Using the Elastic Search and Kibana stack we can aggregate all the logs into a central database which can serve as a starting point for the analysis.

The log entries have in total of 23 features. Not all the features are always present for all the log entries. The features appearing are: `_id, _index, _score` (added metadata from Kibana), `_type` (collector which is responsible for sending all the metrics and logs to Kibana), `hostname` (name of the physical host (e.g., wally113)) `user id, project domain, tenant id, request id, user domain, domain id` (features describing the user request to Openstack) `timestamp, @timestamp` (time when the record was created), `log level` (describes the level of the log entry (info, error, warrning etc.)), `pid` (Process ID), `Payload` (has the most important information of the log i.e., the body of the log entry), `programname` (the OpenStack project that generated the log entry), `python module` (module responsible for generation of the log entry), `logger` (tells which project logs the event) `http related fields` (only present if there is an HTTP call describing the endpoint, status core, version, and the method).

4.3 Traces

The traces in the dataset are contained in 3 directories: *boot_delete, create_delete_image,* and *network_create_delete*. Each of the directories contains the scripts for running the workload and the fault injections along with the actual tracing data. These directories contain `.JSON` files of the traces. This structure is preserved among all types of workloads (Rally actions).

Every trace has its features in the `JSON` entries or events. These features depend on multiple factors such as the user request, infrastructure, load balancers, and caching. An event is a vector of key-value pairs (k_i, v_i) describing the state, performance, and further characteristics of service at a given time t_i. In following we describe the main features of the events in a trace:`host`. Name of the physical host, `name`, event name (e.g., compute_apistop), `service`. service name (e.g., osapi_compute), `project`, Openstack project (e.g., nova), `timestamp`, the time when the event is recorded, `trace_id`, ID of the span (contains two events, e.g., compute_api-stop and compute_api-start), `parent_id`, the *parent_id* gives the ID of the parent event. This attribute can be used to represent the trace in a graph, `base_id`. ID of the trace, different events and spans with same base_id belong to one trace.

Two events, start and stop (e.g., `compute_apistart` and `compute_apistop`) with the same *trace_id* form a span. The subtraction between the stop timestamp and the start timestamp gives the duration of the span. The above features together with the duration are the most important in describing the structure, preserving the parent-child causal relationship, and the duration which represents the response time of the service invoked.

The events also contain other attributes that can be found for specific types. For example, `path, scheme, method` for HTTP calls, where the `path` and

`scheme` represents the HTTP endpoint and HTTP scheme and `method` can be GET or POST. Further, the `db statement` in DB (data base) calls gives information about the SQL query, while the `function, name, args, kwargs` in RPC calls tell which function was invoked with the its corresponding arguments.

5 Applications of Multi-source AIOps

While previous work has been generally done on single-source data, we believe that to develop robust, holistic approaches for anomaly detection, root-cause analysis, self-healing, resource optimization, and performance analysis a multi-source data is highly desirable. In this section, we shortly describe anomaly detection and root-cause analysis can exploit the benefits of processing multi-source observability data.

Multi-source Anomaly Detection. The distributed logs over projects and physical hosts enable multimodal end-to-end learning and more robust log anomaly detection. Of course, this adds complexity for data integration and fusion, as the distributed logs are produced with different timestamps. Together, the distributed logs and metrics can again be combined into more complex model or network of models. Lastly, the graph-like structures of the tracing data can be incorporated to complete the robust anomaly detection where all available observability data is considered.

Root-cause Analysis. The integration of multi-source observability data can be exploited by using some kind of Fishbone diagrams [10] to find the root-cause of problems. A method can start with simple metric-only anomaly detection, which typically provides little information about the root-causes of problems, and drill down to more complex data structures which are richer in explaining anomalies. For example, one can start by analyzing the latency of microservices endpoints. If anomalies are detected after processing metrics, one can use the timeframe when the anomaly occurred to select and analyze structural changes in traces. Traces can provide information about which servers are possibly faulty. Afterwards, application logs can be accessed to find the root-cause of problems.

6 Conclusion

AIOps systems rely on suitable observability data. We released a multi-source data containing distributed metrics, logs, and tracing data obtained from a complex distributed system based on microservice architecture. We describe in details the infrastructure, experiments performed, and the fault injection. Furthermore, we provided descriptive statistical properties of the data.

Furthermore, we motivated possible applications of this data for improvements in anomaly detection, root-cause analysis, remediation, and feature extension. We hope that this dataset will foster advances in the research of AIOps, which has been limited mainly to explored data capturing only a single data source category.

References

1. Ahmad, S., Lavin, A., Purdy, S., Agha, Z.: Unsupervised real-time anomaly detection for streaming data. Neurocomputing **262**, 134–147 (2017)
2. Alibaba trace data (2019), https://github.com/alibaba/clusterdata
3. Cfdr (2019), https://www.usenix.org/cfdr-data
4. Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., Bianchini, R.: Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the International Symposium on Operating Systems Principles (SOSP) (2017)
5. Dang, Y., Lin, Q., Huang, P.: Aiops: real-world challenges and research innovations. In: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings. pp. 4–5. IEEE Press (2019)
6. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1285–1298. ACM, New York, NY, USA (2017)
7. Elki (2019), https://elki-project.github.io/datasets/outlier
8. Goldstein, M.: Unsupervised Anomaly Detection Benchmark (2015), https://doi.org/10.7910/DVN/OPQMVF
9. Google trace data (2019), https://github.com/google/cluster-data
10. Ishikawa, K.: Guide to Quality Control. JUSE, Tokyo (2012)
11. Lmu (2019), https://www.dbs.ifi.lmu.de/research/outlier-evaluation/
12. Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., Zhou, R.: Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 4739–4745 (2019)
13. Nagios enterprises, https://github.com/NagiosEnterprises
14. Nedelkoski, S., Cardoso, J., Kao, O.: Anomaly detection and classification using distributed tracing and deep learning. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 241–250. IEEE, New Jersey (2019)
15. Nedelkoski, S., Cardoso, J., Kao, O.: Anomaly detection from system tracing data using multimodal deep learning. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). pp. 179–186. IEEE, New Jersey (2019)
16. OpenZipkin: openzipkin/zipkin (2018), https://github.com/openzipkin/zipkin
17. Schmidt, F., Gulenko, A., Wallschläger, M., Acker, A., Hennig, V., Liu, F., Kao, O.: Iftm - unsupervised anomaly detection for virtualized network function services. In: 2018 IEEE International Conference on Web Services (ICWS). pp. 187–194. IEEE, New Jersey (2018)
18. Shrivastwa, A., Sarat, S., Jackson, K., Bunch, C., Sigler, E., Campbell, T.: OpenStack: Building a Cloud Environment. Packt Publishing (2016)
19. Sridharan, C.: Distributed Systems Observability: A Guide to Building Robust Systems. O'Reilly Media (2018)
20. Oregon (2019), http://odds.cs.stonybrook.edu/
21. Zabbix, https://github.com/zabbix
22. Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and benchmarks for automated log parsing. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice. pp. 121–130. IEEE Press, Piscataway, NJ, USA (2019)