Multi-Source Distributed System Data for AI-powered Analytics

Sasho Nedelkoski*, Jasmin Bogatinovski*, Ajay Kumar Mandapati*, Soeren Becker*, Jorge Cardoso†, Odej Kao*
*Complex and Distributed IT-Systems Group, TU Berlin, Berlin, Germany
Email: {nedelkoski, firstname.lastname}@tu-berlin.de
†Huawei Munich Research Center, Munich, Germany
Department of Informatics Engineering/CISUC, University of Coimbra, Portugal
Email: jorge.cardoso@huawei.com

***Abstract*—In recent years there has been an increased interest in Artificial Intelligence for IT Operations (AIOps). This field utilizes monitoring data from IT systems, big data platforms, and machine learning to automate various operations and maintenance (O&M) tasks for distributed systems. The major contributions have been materialized in the form of novel algorithms. Typically, researchers took the challenge of exploring one specific type of observability data sources, such as application logs, metrics, and distributed traces, to create new algorithms. Nonetheless, due to the low signal-to-noise ratio of monitoring data, there is a consensus that only the analysis of multi-source monitoring data will enable the development of useful algorithms that have better performance. Unfortunately, existing datasets usually contain only a single source of data, often logs or metrics. This limits the possibilities for greater advances in AIOps research. Thus, we generated high-quality multi-source data composed of distributed traces, application logs, and metrics from a complex distributed system. This paper provides detailed descriptions of the experiments, statistics of the data, and identifies how such data can be analyzed to support O&M tasks such as anomaly detection, root cause analysis, and remediation. The data is available at https://doi.org/10.5281/zenodo.3484800.**

***Index Terms*—AIOps, dataset, anomaly detection, root-cause analysis, observability, application logs, metrics, distributed tracing.**

I. Introduction

The term AIOps refers to multi-layered technology platforms that automate and enhance IT operations by using analytics and machine learning [1].

AIOps was introduced to make O&M tasks of ever-increasing complex public, private, edge, mobile, and hybrid cloud environments effective and cost-efficient. 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 [2], [3].

Monitoring data is a key element of new AIOps tools, but it is also one of the cornerstones of research. The *observability* data generated by distributed IT systems can be classified into three main categories: metrics, application logs, and distributed traces [4]. *Metrics* are numeric values measured over a period of 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 tasks, such as anomaly detection and root-cause analysis, which process a specific type of observability data. For example, anomaly detection has been applied to metrics [5]–[7], logs [8]–[12], and also to distributed system traces [13], [14].

The existing research has mainly explored data capturing only a single data source category. This limits both the development of new multi-source (or multimodal) methods and their proper evaluation. The absence of observability data capturing the three data categories prevents the development of methods for fault detection, root-cause analysis, and remediation that could give advances in the field as existing approaches typically produce a large number of false positives. The benefits of using several data source categories are related to the concept of triangulation [15] from the field of social sciences, which refers that the application and combination of several observers, methods, and empirical materials can overcome the weaknesses and limitations that come from single observers and single methods.

In following we summarize the contributions of this work:

- 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 observability data and its statistics.
- 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.

II. Related Work

Traces, logs, and metrics are important data sources that are fundamental to the operation of complex distributed systems. The metric data is a common way to extract useful information for describing the state of the system. However, it is not sufficient for a holistic approach aiming to model the whole system. The metrics data are obtained from monitoring of the resources such as CPU, memory, disk and network throughput and latency. A plethora of available collections of datasets containing metric data can be found in Stonybrook [16], where multiple datasets for different tasks related to anomaly detection can be found. Numenta [17] predominantly contains datasets from streaming and real-time applications, while Harvard [18], ELKI [19], LMU [20] store network intrusion data. Recently, there are multiple studies which utilize these datasets for anomaly detection, root-cause analysis, and remediation. In Subutai et al. [17], a novel anomaly detection method based on hierarchical temporal memory (HTM) is introduced. It enables anomaly detection in the streaming setting to tackle the problems of concept drift and the problem of multiple streaming sources utilizing metrics data. In Schmidt et al. [5], an unsupervised anomaly detection framework is developed and applied to real-time monitoring data in a distributed environment.

The main challenge that AIOps systems analyzing 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 [21] stores links or 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 time 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 Meng et al. [23] the LogAnomlay system for detection of anomalies from logs is introduced. It utilizes a novel template2vec technique to encode the logs. Further, it extracts quantitative patterns from the logs. It uses LSTMs to detect the sequential and quantitative anomalies in the logs. In Zheng et al. [10] the DeepLog system is introduced. It tries to model the logs as natural language sequences. It allows to update the model by the operator and provides an automatic reconstruction of the workflows to enable root cause analysis.

In microservice architectures, traces are graph-like structures composed of events or spans [24]. 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. [13], [14] 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 representative traces of the virtual machine of Microsoft Azure [25]. 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 [26]–[28]. In Zhen et al. [26] a novel system which automatically diagnoses stragglers for jobs is introduced. It utilizes unsupervised clustering methods to group the tasks based on their execution time. Then a supervised learning method is used to diagnose rules for diagnosing the stragglers and their adequate resource assignment data. Li et al. [27] propose a deep reinforcement learning approach towards the job scheduling task. It can automatically obtain a fitness calculation method that optimizes the throughput of a set of jobs from experience. Google's collection of two tracing datasets originates from parts of Google cluster management software and systems [29]. In Elsayed et al. [30] effective predictors of job and task terminations are identified. The knowledge of job config parameters known at launch time is discovered to be sufficient to predict whether a job is going to be killed. Furthermore, various methods for mitigating failures in clusters are proposed, including limiting the number of retries, increasing frequency of checkpoints, scheduling redundant tasks, adjusting scheduling priority or allocated resources and turning on additional monitoring.

Limitation for all the above-mentioned datasets is the absence of multi-view data describing a single system. The lack of data from all observability components from one system does not allow the development of holistic systems for fault detection, root-cause analysis and remediation that consider multiple sources of data simultaneously. Our collection of data, describing the same system from the 3 perspectives of logs, metricise 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 [13].

III. Dataset generator

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

A. Infrastructure

For the generation of the data, we deployed an OpenStack [31] testbed based on a microservice architecture that is running in a dockerized environment called Kolla-Ansible [32]. OpenStack is a cloud operating system that controls large pools of computing, storage, and networking resources throughout a data-centre, all managed and provisioned through APIs with common authentication mechanisms.

The testbed setup is shown in Figure 1 and consists of one control node named `wally-113` and four compute nodes: `wally-122`, `wally-123`, `wally-124`, and `wally-117`.

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.

B. Workloads and faults injected

To generate workloads and inject faults into the infrastructure we used Rally [33]. It automates and unifies multi-node OpenStack deployment, cloud verification, testing and profiling. Rally does it generically, making it possible to check whether OpenStack is going to work well on various configurations and installations under high workloads.

Rally docker image was used to create the load and inject os-faults [34] appropriately. We selected a list of workloads and faults that are close representatives to real production faults. The listed workloads and faults in following cover user request that is served by the main Openstack projects.

- *Create and delete server*. Creates and deletes a server using the Rally task (NovaServers.boot_and_delete_server). Nova project is mostly affected and present in the data. We injected a compute fault which is restarting the api container that run on the compute nodes.
- *Create and delete image*. Rally task (GlanceImages.create_and_delete_image) for images accepts the image-location locally/ over the internet , format of the output image once created. It creates and deletes an image. The glance project of Openstack provides a service where users can upload and discover data assets that are meant to be used with other services. Here we inject the fault in the `glance-api` running on the controller node.
- *Create and delete network*. Rally provides task (NeutronNetworks.create_and_delete_networks) the format from creating and deletion of networks for various configurations such as multiple users and tenants. Neutron is an OpenStack project to provide networking as a service between interface devices (e.g., vNICs) managed by other Openstack services (e.g., nova, heat etc). 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 1000 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 the iterations for each action was chosen so that all workloads approximately finish in 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 which injections were made. This report serves as ground truth for the normal and anomalous state of the system.

C. Data collection

In following we describe the technologies and the methods used to collect the generated data.

1) Metrics: For the metrics collection across the physical nodes in the infrastructure, we utilize Glances [35], a cross-platform monitoring tool which aims to present a maximum of information into a minimal space through curses or Web-based interface. Glances is written in Python and uses the `psutil` library to get information from a system. It can adapt dynamically the displayed information depending on the terminal size. It can also work in client/server mode, also remote monitoring could be done via terminal, Web interface or API (XMLRPC and RESTful). Glances was used to gather information such as CPU, MEM and load of the machine (either controller or the compute nodes) with a frequency of 0.1 seconds. These metrics were saved into a CSV file via the `glances-cli`.

2) Logs: OpenStack services use standard logging levels. For aggregating logs from all services running across the physical nodes, was used ELK (Elasticsearch, Logstash, and Kibana). Elasticsearch is a search and analytics engine which resolves the search requests. Logstash is a serverside data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to Elasticsearch. For this Fluentd, which is an open-source data collector for the unified logging layer, was utilized. It allows unifying data collection and consumption for better use and understanding of data. Kibana is a dashboard that gives the ability to the users to visualize data with charts and graphs using data that is collected by Elasticsearch. Finally, for exporting data from Elasticsearch into CSV a CLI tool, es2csv [36] was utilized. The benefit we obtain from this tool is that it can query bulk docs in multiple indices and get only selected fields, this reduces query execution time and enhances the speed of aggregating these logs that are existing on various physical nodes. We provide both, the aggregated logs as well as the raw logs to cover possible development of methods that process raw logs, such as log parsing.

3) Traces: OpenStack consists of multiple projects, where each project is composed of multiple services. To process user requests, e.g., creating a virtual machine, OpenStack uses multiple services from different projects. To support troubleshooting, OpenStack introduces a small but powerful

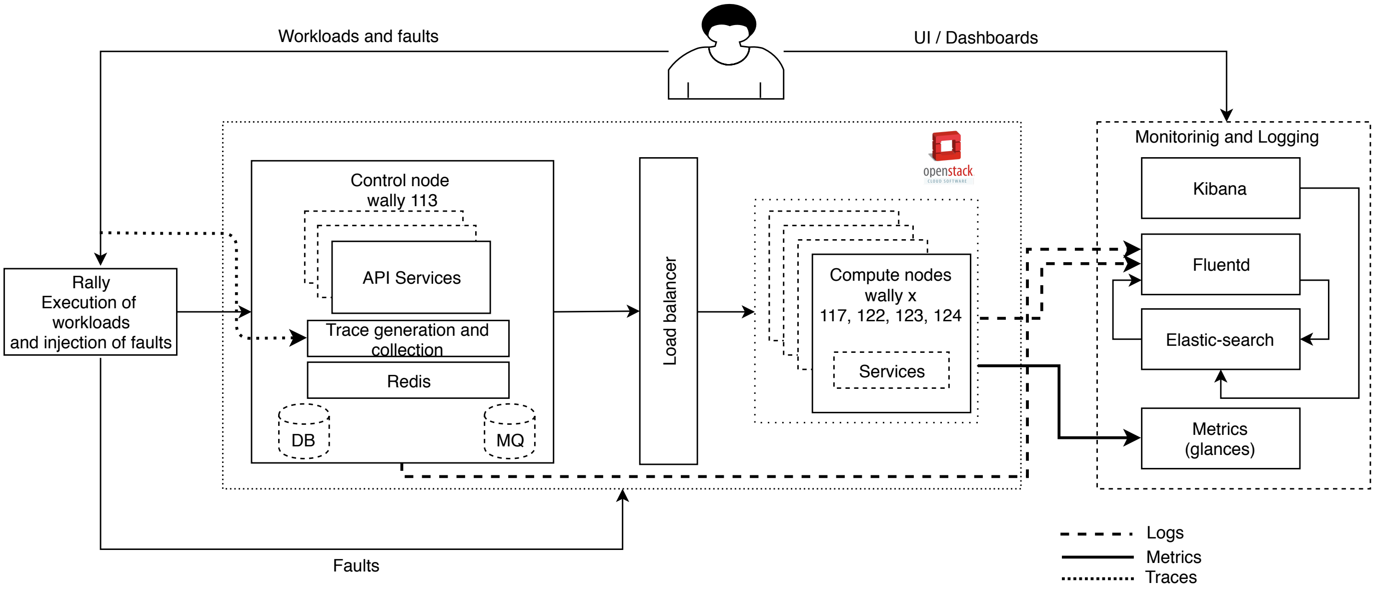

Fig. 1. Illustration of the infrastructure from where the data was generated.

library called `osprofiler` that is used by all OpenStack projects and their Python clients [37] to generate traces. It generates one trace per request, that goes through all involved services, and builds a tree of calls which captures a workflow of service invocations. To identify workflows, we monitor the following call types:

- HTTP. Captures HTTP requests, the latency of service, and projects involved.
- RPC. Represent the duration of parts of request related to different services in one project.
- DB API. The time that the request spent in the DB layer.
- Driver. In the case of nova, cinder and others we have vendor drivers.

The `osprofiler` library collects these records in a trace per request and stores them in a database (e.g., Redis). From Redis, we can query and analyze traces.

IV. 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 in order to provide the state of the system from multiple points of view. In the following two sections, we describe the main attributes, properties, and statistics of each data category of the first experiment. Due to page limitations, we refer the reader to the above link in the abstract for the code for extracting the data statistics from the second experiment. All other properties hold for both experiments.

A. Metrics

The metrics data category contains data for the 5 physical nodes in the infrastructure. The frequency of the recordings is 0.1 seconds for the compute nodes and for the controller node. 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`. Percent time spent in userspace. The user CPU time is the time spent on the processor running your programs code (or code in libraries).
- `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. This measures demand, which can be greater than what the system is currently processing.

A small sample of the metrics data for the `wally113` is shown in Table I where we can see part of the metrics data.

B. 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, there are different project running. The control node has more services running and thus has more log files for the OpenStack projects. Each project on the physical hosts has its log directory where the logs are stored. Inside each of the log directories for the projects, there are several log files. Important to note here is that even the log files are highly distributed over projects and physical nodes, they all represent the state of the system. We provide the raw log directories in this dataset along with the aggregated log file. Using the elastic

TABLE I
METRICS FROM THE CONTROLLER NODE (WALLY 113)

timestamp	cpu.user	mem.used (B)	load.cpucore	load.min1	load.min5	load.min15
2019-11-19 16:56:32	11.5	10221035520	8	0.8	1.02	1.18
2019-11-19 16:56:32	10.4	10221117440	8	0.8	1.02	1.18
2019-11-19 16:56:33	11.1	10222948352	8	0.8	1.02	1.18
2019-11-19 16:56:33	14.3	10223144960	8	0.8	1.02	1.18
2019-11-19 16:56:34	10.7	10222866432	8	0.8	1.02	1.18
2019-11-19 16:56:34	10.7	10223480832	8	0.8	1.02	1.18

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: *_id, _index, _score* are added metadata from Kibana. The *_type* is fluent, the collector which is responsible for sending all the metrics and logs to Kibana. In the following, we describe the main features present in the log data.

- hostname. Name of the physical host (e.g., wally113)
- user_id, project_domain, tenant_id, request_id, user_domain, domain_id. Are features describing the user request to Openstack.
- timestamp, @timestamp. The time when the record was created.
- log_level. Describes the level of the log entry. It can be info, error, warning, etc.
- pid. Process ID.
- Payload. Gives 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 The module responsible for generation of the log entry, and the
- logger Tells which project logs the event.
- http_* related fields. Are only present if there is an HTTP call describing the endpoint, status core, version, and the method.

For the parsing of the logs, template matching, and analysis we suggest using the aggregated file described instead of the directories with raw log files, as all of the information is preserved and more structured for direct analysis. For multi-source log anomaly detection, if the aggregated file is utilized, we suggest splitting by ”logger” in order to obtain entries which are grouped by their corresponding service.

C. 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 start and stop events (e.g., compute_apistart and compute_apistop) with the same *trace_id*. 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 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.

D. Ground Truth Labels

The workloads described along with the faults injected are both recorded in Rally HTML and JSON reports which are located at each of the directories containing trace data. These reports provide pseudo ground truth labels for the traces, metrics, and logs. They contain information for the times when the faults were injected and the resulting high level error messages. Taking the period when the anomaly was injected and merging it with the timestamps of the data files can give us true labels for the evaluation. We suggest to use the ground truth labels to evaluate algorithms and methods which are based on unsupervised learning, as in production systems injection of anomalies and access to labeled data is restricted.

V. Dataset Statistics

This section provides a descriptive statistic of the datasets generated. It quantitatively describes the properties of the trace, metrics, and log datasets. In following we discuss the statistics for the first experiment only, due to page limitations. The code for extracting the statistics for the second experiment is provided in the data repository.

A. Metrics

The number of recordings of the utilization of the resources, more specifically the CPU, memory and the load, per node varies in the range of (108900, 298251). The average number of recordings is 239127. The total number of the metric recordings is 1195637. All of the nodes have 8 CPU cores. It is important to note that the metrics data cover a time span larger than the period of execution of the experiments.

In general, the wally113 experience the greatest CPU and memory load as observed by the distribution of these two features. Furthermore, the correlation analysis of the load.min1, load.min5 and load.min15 show that they exhibit high correlation given their relatedness through time. The correlation analysis also shows quite distinct behaviour for the load.min5, load.min10, load.min15 correlations between the control node and the remaining nodes. Regarding the dependence between the cpu.user, memory.used and load.min features, no significant correlation can be identified. Roughly 3 groups of features emerge - the load.CPU, mem.used and the load.min group.

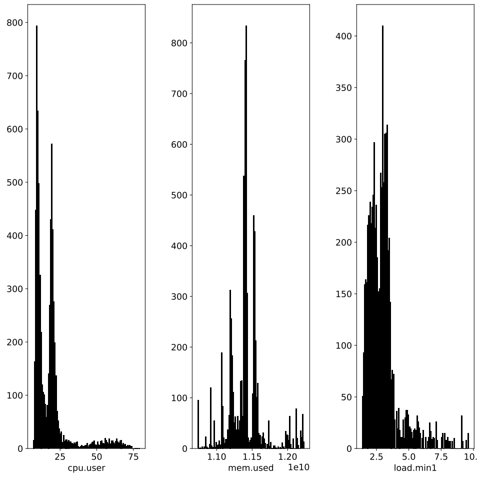

Fig. 2. Distribution of the values of the metrics feature for the node wally113.

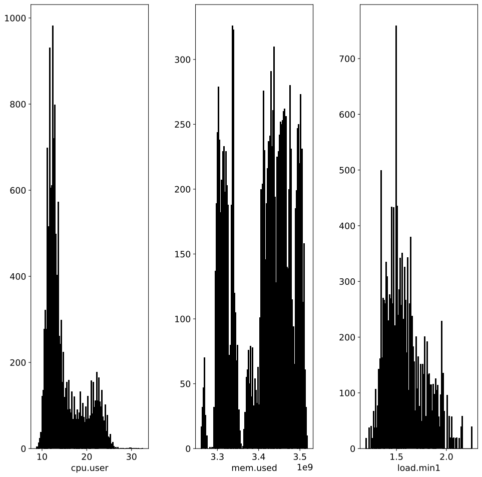

Fig. 3. Distribution of the values of the metrics feature for the node wally117.

B. Logs

Since the logs are semi-structured data, first we try to organize them and observe the range of interesting features that can appear in them. There are 139799 log messages appearing in the sequential execution of the operations. We used Kibana to identify the different features describing them. Each log has its unique identifier referenced by the label `_id`. The `Timestamp` feature has 8 missing values. However, the timestamps provided by Kibana, stored in `@timestamp` contain the relevant information for the moment where the logging happened.

There are a total of 6 services recording their logs in the OpenStack logger: nova, neutron, keystone, glances, placement and cinder. Nova and neutron are services with the greatest number of logs appearing. The logs contain 3 levels of logging (INFO, WARNING and ERROR). There are 5 operation host nodes - `Hostname` (wally113, wally117, wally122, wally123, wally124). Most of the logs originate from the control node wally113. The `python_module` contains the name of the 61 modules that are logging their information into the logs with wsgi related modules being the most frequent ones (neutron_wsgi, nova.osapi_wsgi and server_wsgi). The `programname` refers to the program which operations are being executed. There are a total of 127654 different `Payloads` happened in the system and the most frequent is related to the GET operation.

For the realized HTTP calls there is information for the `http status` with 6 different code values, `http_method` with 4 possible values (GET, POST, DELETE and PUT), `http_urls` with a total of 3655 values and the version of the http protocol stored ins `http_version`. There are columns such as`_domain_id`, `user_domain`, `tenant_id`, `request_id`, `user_id`, `_score`, `_type`, `project_domain`, `Pid` and `domain_id` that have either very large or very small variance in the number of unique values per feature. They represent start and end point in form of IP address or a result from a hash function.

C. Traces

Table II represents the total number of services for each of the traces for the three sequential operations being executed. It

TABLE II
TRACES: COUNTS OF SERVICES PER RALLY ACTION

	wsgi	db	comp. api	nova image	neutron api	neutron db	rpc
image create delete	11436	81321	0	0	0	0	0
network create delete	4692	14101	0	0	0	125321	855
boot delete	46591	125975	21572	752	313744	46642	36560

TABLE III
TRACES INFORMATION: MEDIAN TIME OF A SERVICE PER ITERATION

	wsgi	db	comp. api	nova image	neutron api	neutron db	rpc
image create delete	0.046	0.001	0	0	0	0	0
network create delete	0.285	0.001	0	0	0	0.001	0.001
boot delete	0.0410	0.001	0.039	0.035	0.001	0.002	0.009

is given as a total sum over all the repetition of the experiment. One can be observe that there are different service invoked per operation. For example, for the `image_create_delete` operation the open stack service involved is completely on the controller node, hence the compute nodes are contacted and there is no operation related to them. The most frequently occurring invocation is split between db and wsgi. Second the operations are ordered by complexity and it can be seen that the `boot_delete_task` involves all of the 7 services.

Table III represents the median time of execution for each of the invoked services. The median is chosen since the distributions are skewed and the mean is not representative of the sample distribution. As it can be observed, the wsgi services are slower than the db calls since wsgi relays on http communication. It is interesting to observe that for the network create delete operation the rpc is quite small. One explanation for this is the small rate of rpc call per individual execution. This means that not all executions of this operation involve rpc calls. Since multiple workloads involve invoking different number of individual operation the times should be compared with caution.

VI. 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.

One of the first proposals to use more than one category of observability data in a single model is made in Nedelkoski et al. [13], [14]. A new research direction for anomaly detection is explored. Besides the response time, a second category of data: the tracing data collected during the execution of system operations is analyzed. The results of this new approach are encouraging and provide a direction of development of models that combine and explore additional data categories.

In this section, we shortly describe possible AIOps approaches that 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 [38] 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.

Precision Increase. Ensemble learning [39] can be used to machine learning algorithm results by combining several models applied to the three correlated data sources categories. Such an approach would allow the production of algorithms with better predictive accuracy when compared to the algorithms which process single-data sources.

Feature Extension. Many machine learning algorithms rely on features, which for AIOps are individual measurable characteristics of the behaviour of IT distributed systems at a given time. By using multi-source data, the spectrum of available features to an algorithm is dramatically increased. Thus, we expect the quality of algorithms and their results to increase in the future.

VII. 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] Y. Dang, Q. Lin, and P. Huang, "Aiops: real-world challenges and research innovations," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*. IEEE Press, 2019, pp. 4–5.

[2] "Zabbix." [Online]. Available: https://github.com/zabbix

[3] "Nagios enterprises." [Online]. Available: https://github.com/NagiosEnterprises

[4] C. Sridharan, *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media, 2018.

[5] F. Schmidt, A. Gulenko, M. Wallschlger, A. Acker, V. Hennig, F. Liu, and O. Kao, "Iftm - unsupervised anomaly detection for virtualized network function services," in *2018 IEEE International Conference on Web Services (ICWS)*. New Jersey: IEEE, 2018, pp. 187–194.

[6] A. Gulenko, F. Schmidt, A. Acker, M. Wallschlager, O. Kao, and F. Liu, "Detecting anomalous behavior of black-box services modeled with distance-based online clustering," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. New Jersey: IEEE, 2018, pp. 912–915.

[7] F. Schmidt, F. Suri-Payer, A. Gulenko, M. Wallschlger, A. Acker, and O. Kao, "Unsupervised anomaly event detection for cloud monitoring using online arima," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. New Jersey: IEEE, 2018, pp. 71–76.

[8] F. Pina, J. Correia, R. Filipe, F. Araujo, and J. Cardoso, "Nonintrusive monitoring of microservice-based systems," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, 2018, pp. 1–8.

[9] J. Correia, F. Ribeiro, R. Filipe, F. Arauio, and J. Cardoso, "Response time characterization of microservice-based systems," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. New Jersey: IEEE, 2018, pp. 1–5.

[10] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2017, pp. 1285–1298.

[11] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. New York, NY, USA: ACM, 2016, pp. 1573–1582.

[12] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 207–218, 2016.

[13] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection from system tracing data using multimodal deep learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. New Jersey: IEEE, 2019, pp. 179–186.

[14] ——, "Anomaly detection and classification using distributed tracing and deep learning," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. New Jersey: IEEE, 2019, pp. 241–250.

[15] T. D. Jick, "Mixing qualitative and quantitative methods: Triangulation in action," *Administrative Science Quarterly*, vol. 24, pp. 602–611, 1979.

[16] (2019) Oregon. [Online]. Available: http://odds.cs.stonybrook.edu/

[17] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134 – 147, 2017.

[18] M. Goldstein. (2015) Unsupervised Anomaly Detection Benchmark. [Online]. Available: https://doi.org/10.7910/DVN/OPQMVF

[19] (2019) Elki. [Online]. Available: https://elki-project.github.io/datasets/outlier

[20] (2019) Lmu. [Online]. Available: https://www.dbs.ifi.lmu.de/research/outlier-evaluation/

[21] (2019) Cfdr. [Online]. Available: https://www.usenix.org/cfdr-data

[22] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. Piscataway, NJ, USA: IEEE Press, 2019, pp. 121–130.

[23] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "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*, 2019, pp. 4739–4745.

[24] OpenZipkin, "openzipkin/zipkin," 2018. [Online]. Available: https://github.com/openzipkin/zipkin

[25] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "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.

[26] H. Shen and C. Li, "Zeno: A straggler diagnosis system for distributed computing using machine learning," in *High Performance Computing*. Cham: Springer International Publishing, 2018, pp. 144–162.

[27] F. Li and B. Hu, "Deepjs: Job scheduling based on deep reinforcement learning in cloud data center," in *Proceedings of the 2019 4th International Conference on Big Data and Computing*. New York, NY, USA: ACM, 2019, pp. 48–53.

[28] (2019) Alibaba trace data. [Online]. Available: https://github.com/alibaba/clusterdata

[29] (2019) Google trace data. [Online]. Available: https://github.com/google/cluster-data

[30] N. El-Sayed, H. Zhu, and B. Schroeder, "Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations," in *International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1333–1344.

[31] A. Shrivastwa, S. Sarat, K. Jackson, C. Bunch, E. Sigler, and T. Campbell, *OpenStack: Building a Cloud Environment*. Packt Publishing, 2016.

[32] "Kolla-ansible's documentation." [Online]. Available: https://docs.openstack.org/kolla-ansible/latest/

[33] "Rally." [Online]. Available: https://rally.readthedocs.io/en/latest/

[34] Performa, "os-faults." [Online]. Available: https://opendev.org/performa/os-faults

[35] Nicolargo, "nicolargo/glances," 2019. [Online]. Available: https://github.com/nicolargo/glances

[36] Taraslayshchuk, "taraslayshchuk/es2csv," 2018. [Online]. Available: https://github.com/taraslayshchuk/es2csv

[37] Openstack, "openstack/osprofiler." [Online]. Available: https://github.com/openstack/osprofiler

[38] K. Ishikawa, *Guide to Quality Control*. Tokyo: JUSE, 2012.

[39] C. Zhang and Y. Ma, *Ensemble Machine Learning: Methods and Applications*. New York: Springer Verlag, 2012.