logs2graphs: Data-driven graph representation and visualization of log data

Stefan Andonov
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, North Macedonia
stefan.andonov@finki.ukim.mk

Viktor Jovev
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, North Macedonia
viktor.jovev@students.finki.ukim.mk

Aleksandar Kitanovski
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, North Macedonia
aleksandar.kitanovski@students.finki.ukim.mk

Aleksandar Krsteski
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, North Macedonia
aleksandar.krsteski@students.finki.ukim.mk

Gjorgji Madjarov
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, North Macedonia
gjorgji.madjarov@finki.ukim.mk

***Abstract*—In recent years, AIOps has helped a lot with the exploration of different types of resources, in the processes of optimization and automation of complex IT operations. One of the main resources that AIOps is exploring is system logs. There are many techniques based on machine learning in AIOps that help in logs anomaly detection, logs prediction, and root cause analysis guided by logs, but a majority of them are considering log messages either individually or as log sequences, without exploring the relationships between different types of logs. We believe that those relationships can be expressed via using graph representations of log messages and those representations can be utilized in almost any AIOps operation. Therefore in this paper, we present logs2graphs, an open-source system for the creation and visualization of such graph representations of log messages, which is compatible with several publicly available log sources and expandable to other log sources.**

***Index Terms*—AIOps, logs, graphs, visualization, software engineering, design patterns**

I. Introduction

A. AIOps

AIOps is a field that became popular after 2017 when the term AIOps was coined by Gartner in [8]. As AIOps is a modern concept, there is no widely adopted definition yet. [11] For the purpose of this paper we will use the definition by Nedelkovski et al. in [10]: "Artificial Intelligence for IT Operations (AIOps) combines big data and machine learning to replace a broad range of IT Operations tasks including availability, performance, and monitoring of services. By exploiting log, tracing, metrics, and network data, AIOps enable detection of faults and issues of services". According to the definition, one of the 4 important resources that are explored by AIOps are logs, so therefore this paper focuses on a graph representation of log data which can later be used in various operations related to AIOps. There are many papers on the topic of AIOps which are focused on logs and how they can be used to detect future anomalies or determine the root cause of incidents, but mainly all of them are considering logs just as a sequence of logs. We believe that log messages can be represented as graph structures in which the relationship between log message types will be highlighted. This concept was also recently introduced in [12]. As our goal is a graph representation of logs, the next subsection will give a short introduction to logs and graphs.

B. Logs

The background of computer systems generates logs in order to record notable events and executions indicating what has happened. Therefore, these logs are precious sources to check and investigate system status and detect anomalies in them. Logs usually are in a structured format that is defined on a system-level i.e all the logs that are generated by one system will be in the same format. Log formats are varying depending on the system, but in general they all mandatory contain a timestamp (the moment when the log was produced on the system), content (textual explanation of what happened in the system), type (information about the severity of the log per ex. error, information, etc.)

C. Graphs introduction

A graph is a mathematical structure used to model pairwise relations between objects. There are different types of graphs, the ones used in this project are directed graphs. Directed graphs are formally defined as an ordered pair $G = (V, E)$, where V is a set of nodes (or vertices), and E is a set of links (or edges) which are ordered pairs of nodes (v_1, v_2) meaning there is a link from node v_1 to v_2. For the purposes of our paper, we build graphs from the logs as described in [12] - each alert type is a node, and there is a link between two nodes (r_1, r_2) if an alert of type r_1 appears before an alert of type r_2. A detailed explanation of the graph representation of log sequences is provided in Section 4.

In the following, Section 2 provides an overview of the relevant work for log collection, log parsing, and their application in AIOps. Section 3 offers detailed descriptions of some publicly available log sources which will be available for exploration in our system. Section 4 presents technical details on the system design, architecture, and implementation

of all subsystems part of the logs2graphs tool. Section 5 demonstrates the user interface of our system as well as the visualization of the graph representation of different types of logs, which is the primary goal of our paper. Section 6 states our plans for utilization of this application in future research related to building models for graph representation learning. Section 7 concludes the paper and its results.

II. Relevant work

Logs are an important tool for monitoring and troubleshooting computer system behavior [5, 4]. As a result, there has been substantial work on automated log analysis. Techniques have been proposed for highlighting anomalous messages [3, 2] or patterns of messages [4]. Generally, these techniques are evaluated on proprietary log sources described in fairly general terms. Many recent papers describe the logs in question with a handful of excerpted lines plus a few aggregate statistics [5]. This is unfortunate because it makes it hard to reason about the relationship between log structure and analysis quality. Many research results are based on measurements taken at particular sites with highly customized software environments; there is often a justifiable reluctance to reveal operational details. Additionally, the general workflow is adapted when working with logs as shown in [6]. The phases relevant from that workflow for this work are:

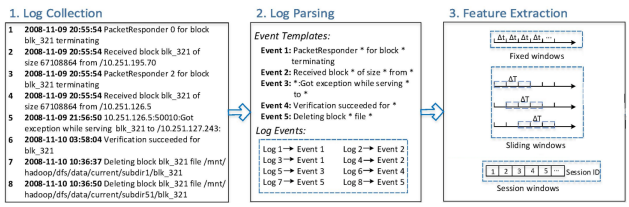

Fig. 1. Workflow (Source: [6])

- Log Collection (This phase is generally completed because in our paper we are using publicly available datasets explained in Section 3)
- Log Parsing (There are several techniques for parsing of the logs and extraction of generic event templates from log collection. In our work we are using DRAIN which will be explained shortly in Section 4)
- Feature Extraction (Based on the extracted event templates, log data is being windowed with the usage of tumbling windows, sliding windows, and session windows, if applicable. Features are being extracted from the logs in the created windows and afterward they are used for supervised and unsupervised tasks in Machine Learning.)

In this work, we are replacing the phase of feature extraction with the phase of the creation of graph representation for logs, as feature extraction is a stage that belongs to a more general terminology related to traditional machine learning processes.

III. Log message sources

A. BGL

BGL represents a log message source that contains logs collected from the Blue Gene/L Supercomputer made by IBM. The supercomputer has 131072 processors and 32768GB of memory. This source has around 4.8 million alert and non-alert messages identified by category tags, collected between 03.06.2005 and 04.01.2006 (nearly 6 months). Details about the Blue Gene/L System, methods of log collection, and alert identification can be found in [1].

B. HDFS

The logs were already published online from [5]. There are 24 million lines of logs. Most of the events in logs indicate run-time performance problems that have been confirmed by Hadoop developers. All errors, warnings, and information are collected between 09.11.2008 and 11.11.2008 (about 3 days).

The log message source is generated in a private cloud environment using benchmark workloads. The logs are sliced into traces according to block ids.

C. OpenStack Nova

OpenStack is an open-source cloud operating system that can control large pools of compute, storage, and networking resources throughout a data center, all managed and provisioned through APIs. Because it is a complex system, OpenStack is divided into 3 subsystems: Nova (for instances management), Cinder (for volumes management), and Neutron (for virtual networks management). [9] introduced the concept of fault-injection in order to generate anomalies (bugs) in the logs. The fault-injections are done based on the bugs that were most reported on StackOverflow and are done with surrounding buggy lines of codes with if statements.

A total of 911 tests (with IDs from 1-911) with fault injections were conducted on testbeds. In each test the system first starts with the bug disabled, then the bug is enabled and finally, the bug is disabled again. In this paper, for the purpose of simplicity, we only use the logs obtained from the NOVA subsystem of OpenStack.

IV. System implementation

Logs2graphs is a platform that enables multiple different representations of log message sequences from time and session windows. These representations are created upon user request where the user specifies the type of the graph, the log messages' source, and the configuration of the windows. The logs collected in the specified type and length of the window are represented as graphs and are served in a stream environment for further automated processing and extensive analysis. Also, the created logs representations are presented on a rich interactive graphical user interface, so the user can manually inspect, monitor, and analyze the log messages relationships. The current implementation supports the representation of the log message sequences into different graph structures that emphasize the relationships that exist between the individual log events in a given time or session window.

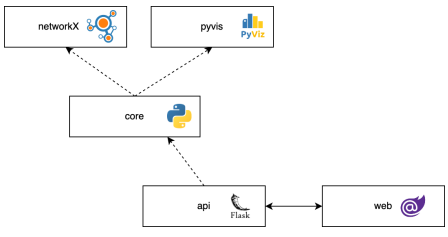

Fig. 2. Diagram of the main system components

A. Software architecture

The software architecture of logs2graphs as shown in Figure 2 is composed of three main components:

- core - The core of the system is a Python library (published as logs2graphs on pypi [1]) that is used for the purpose of log message sources initialization (loading and parsing of the log messages), creation of windows (time and session windows) and graphs from the aggregated logs in the windows. The creation of the windows and graphs is entirely dynamic i.e it is performed on each user request. The core library uses the libraries networkX and pyviz for the representation of graphs and their visualization accordingly.
- API - The API is built in the Flask (Python) framework and it has several endpoints (creating and listing experiments, listing created windows per experiments, generating data for graph visualizations). The API uses the logs2graphs core library for initialization of the log message sources and dynamical creation of the log representations.
- web - The User Interface for the application is created in the .NET Blazor framework and it provides several forms that are used to display the data (including graphs visualizations) obtained from the API or send data to the API when creating experiments.

B. Core library

1) Design of the core library

The core library logs2graphs is a Python package that provides the possibility to download, parse and process all of the 3 log message sources defined in Section 3. This is achieved by the usage of the **Template Method software design pattern** as shown in Figure 3. We have defined

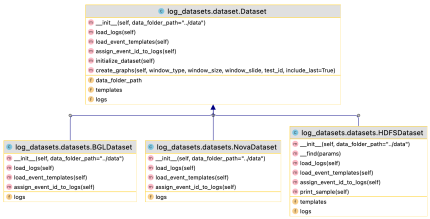

Fig. 3. UML class diagram of the classes used for the representation of the different log message sources in the core library

an abstract class Dataset for the representation of a log message source with the possibility to extend the library by deriving classes for other sources. The class Dataset has a method `initialize_dataset` which is actually the template method participant in the design pattern. This method includes three steps for initialization of the log message sources:

- `load_logs` - used to download the logs locally
- `load_event_templates` - used to perform the log parsing and extraction of event templates with the DRAIN algorithm.
- `assign_event_id_to_logs` - used to assign event template ID to each of the logs, via matching them with the extracted event templates. This is a very important step in the system as it provides a mapping of millions of logs into several important categories (templates) which will be used for creation of a more understandable graph representation.

The three steps are abstract methods in the base class `Dataset` and each of them is overridden in the derived classes in a corresponding way.

For the purposes of event extraction from logs, we used Drain [7] which is an online log parsing method that parses logs in a streaming manner. Drain uses a fixed depth parse tree which encodes specially designed rules in its tree nodes, while the leaves contain log groups. When a new raw log message arrives Drain preprocesses it using simple regular expressions based on domain knowledge. Then the tree is searched by using the specially designed rules encoded in the tree nodes for a suitable log group for the given log message, if a suitable log group is not found then a new log group is created.

The graph creation is done in the method `create_graphs` where based on the `window_type` argument, we are creating a corresponding object from class `GraphsCreator`.

We have utilized the Template Method design pattern again in the definition of the classes `GraphsCreator` and `WindowsCreator` as shown in Figure 4 and Figure 5 accordingly.

In the class `GraphsCreator` the template method is the

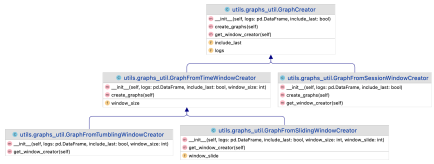

Fig. 4. UML class diagram of the classes used for the creation of graphs

method `create_graphs` while the step that varies in the different graph creators is the `WindowsCreator` object that is used for creation of windows. Therefore, the method `get_window_creator` is overridden differently in all types of graph creator. In the class `WindowsCreator` the template method is the method `create_windows` while the steps that vary in the different windows creators are the start and the frequency (applicable only for the time windows). The creation of the time windows (tumbling and

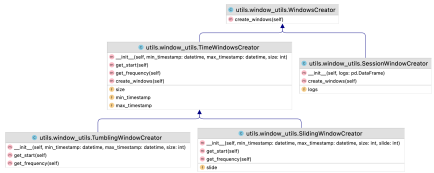

Fig. 5. UML class diagram of the classes used for creation of windows

sliding) depends on the minimal timestamp detected in the logs, as well as on the maximal timestamp.

2) Graph creation

We will explain the process of graph creation for a window of logs through an example. For example, we have obtained a window of 10 logs. We map that sequence of logs to a sequence of the event IDs of the template that was matched

[1] https://pypi.org/project/logs2graphs/0.0.6/

	Including last event ID		Excluding last event ID	
Edge	Weight	Normalized weight	Weight	Normalized weight
A-B	3	0.33	3	0.375
B-A	1	0.11	1	0.125
B-B	2	0.22	2	0.25
B-C	2	0.22	1	0.125
C-A	1	0.11	1	0.125

TABLE I
WEIGHTS OF THE GRAPH FOR THE EXAMPLE SEQUENCE

with each log in the sequence. Let the sequence of event IDs be [A, B, A, B, B, B, C, A, B, C], which means that the first log corresponds to the event template whose id is 'A', the second log corresponds to the event template whose id is 'B' and so forth.

We create graph nodes for each of the unique event IDs present in the sequence. Regarding our example, the graph will have 3 nodes - one for event ID 'A', another for event ID 'B', and the third one for event ID 'C'.

Furthermore, we add directed edges between the nodes. The criteria we use for edge extraction from the log window are the following: if a log corresponding to event ID 'B' came after a log corresponding to event ID 'A' and a directed edge from node 'A' to node 'B' does not exist yet, we add a directed edge from node 'A' to node 'B' with weight 1. Otherwise, if a log corresponding to event ID 'B' came after a log corresponding to event ID 'A' and a directed edge from node 'A' to node 'B' already exists, then we only add 1 to the current weight of the edge. The final step we perform is weight normalization. In this step, we compute the sum of the weights of all edges present in the graph and then we normalize (divide) the weight of each edge with this sum.

The logs2graphs core library allows the users to create graphs from windows with or without the last event ID. As shown in Table I, if we exclude the last event ID we have a smaller weight in the edge B-C. This also causes the normalized weights of the graph to be different.

V. RESULTS

In this section, we will present the user interface of the system with some interesting graphs visualization created and displayed by our system. On Figure 6 a form to submit a new experiment is demonstrated. The user can choose the log messages source (HDFS/BGL/NOVA), the windowing type (tumbling/sliding/session), and other relevant parameters. In this case, the user is submitting an experiment for the creation of graph representation for the BGL log message source which should create sliding windows with a size of 2 hours and a slide of 1 hour. Also, the user wants to exclude the last event ID from the windows when generating the graph representations.

After the request is submitted, the user can open the list of submitted requests for graph representations. When the graph representation creation is completed, the user can open the list of windows created for the requested graph representation as shown in Figure 7. The user on this page can download the full generated graphs data for this graph representation in JSON format, or can select a window and visualize it. In Figure 8, the visualization of the graph created from a selected window is shown.

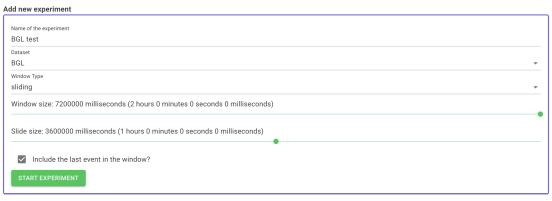

Fig. 6. Form to submit a request for new graph representation

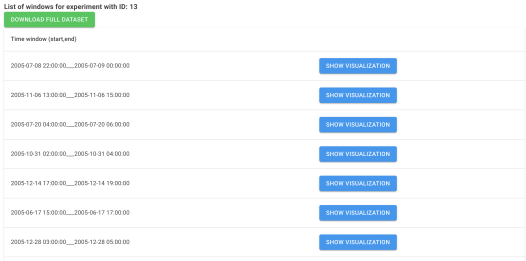

Fig. 7. List of windows generated for the graph representation created in Figure 6

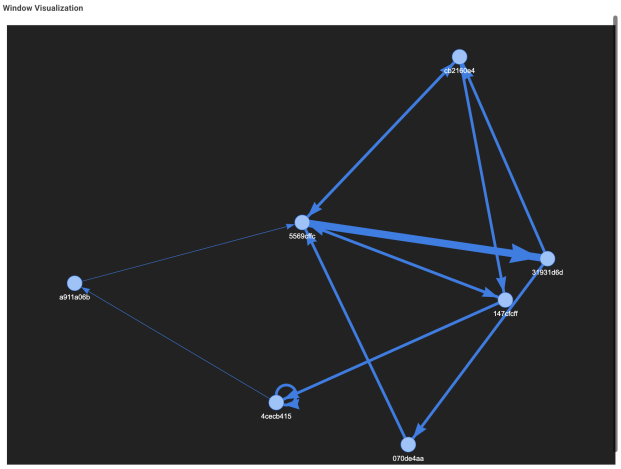

Fig. 8. Graph of the BGL logs in the interval [2005-06-16 18:00:00,2005-06-16 20:00:00) from the graph representation requests in Figure 6

VI. FUTURE WORK

Furthermore, this graph representation of log data can be used for training and testing graph neural networks (GNNs) in order to detect anomalies in logs, predict future logs or perform root cause analysis. Therefore, the future work includes different architectures like GCN (Graph Convolution Networks), GGSNN (Gated Graph Sequence Neural Network), GAT (Graph Attention Networks).

VII. CONCLUSION

The amount of log data created by supercomputers, distributed systems, cloud systems, etc. is massive and cannot possibly be processed by humans, but because the uninterrupted work of these computer systems is crucial for many businesses, hospitals, governments, etc., ways of automated processing and anomaly detection/prediction must be developed. In this paper, we gave an introduction to AIOps as a field, logs and graphs, and an overview of relevant work in those fields. We presented a way to connect the log messages into a graph representation and provided a system with several

components that can be used to create and visualize the graph representations. Finally, we have defined the need for this system in the actions that we plan on taking in our future research.

ACKNOWLEDGEMENTS

This research was partially funded by the Faculty of computer science and engineering, Ss. Cyril and Methodius University in Skopje.

REFERENCES

[1] Adam Oliner and Jon Stearley. "What supercomputers say: A study of five system logs". In: *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE. 2007, pp. 575–584.

[2] Sivan Sabato et al. "Analyzing system logs: A new view of what's important". In: *2nd USENIX workshop on Tackling Computer Systems Problems with Machine Learning Techniques*. 2007.

[3] Michal Aharon et al. "One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2009, pp. 227–243.

[4] Weihang Jiang et al. "Understanding customer problem troubleshooting from storage system logs". In: *Proccedings of the 7th conference on File and storage technologies*. 2009, pp. 43–56.

[5] Wei Xu et al. "Detecting large-scale system problems by mining console logs". In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 2009, pp. 117–132.

[6] Shilin He et al. "Experience report: System log analysis for anomaly detection". In: *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE. 2016, pp. 207–218.

[7] Pinjia He et al. "Drain: An online log parsing approach with fixed depth tree". In: *2017 IEEE international conference on web services (ICWS)*. IEEE. 2017, pp. 33–40.

[8] Andrew Lerner. *AIOps Platforms—Gartner*. https://blogs.gartner.com/andrew-lerner/2017/08/09/aiops-platforms/. [Online; accessed 15-April-2022]. 2017.

[9] Domenico Cotroneo et al. "How bad can a bug get? an empirical analysis of software failures in the openstack cloud computing platform". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 200–211.

[10] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. "Anomaly Detection and Classification using Distributed Tracing and Deep Learning". In: *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2019, pp. 241–250. DOI: 10.1109/CCGRID.2019.00038.

[11] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. "A Survey of AIOps Methods for Failure Management". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 12.6 (2021), pp. 1–45.

[12] Yi Wan et al. "GLAD-PAW: Graph-Based Log Anomaly Detection by Position Aware Weighted Graph Attention Network". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2021, pp. 66–77.