ANM project: DANM! team

AURELIEN VU NGOC, Tsinghua University, China
FABIEN RAVET, Tsinghua University, China

Microservices-operated web services are extremely difficult to monitor and troubleshoot manually, that is why AIOps research are gaining importance recently. This work studies different methods to anomaly detection and root cause analysis in a web service environment. With the help of various performance indicators, we created two models capable of efficiently detecting outliers: the first one uses time series forecasting through a LSTM neural network with a moving average anomaly detection. The second model is hard coded with a set of rules to discover anomalous patterns in the live data. This report demonstrates the main steps of our work on the 2020 ANM project with explicit insights on the utilized methods, clear visual figures and in-depth discussion of the major issues encountered during the process.

CCS Concepts: • **Networks** → *Network algorithms.*

Additional Key Words and Phrases: Advanced Network Management, Anomaly Detection, Time Series Outlier Detection

1 INTRODUCTION

This project aims to design a performant AIOps algorithm to take care of the troubleshooting of a given web service. This work includes taking into consideration the many complex call relations between each microservice and make good use of the various available performance indicators, all while being a live algorithm capable of reacting in real-time. This implies having the capacity to trace back in time the origin of potential anomalies before making precise and quick decisions to help the system recover, and also further develop abilities to recognize potential threats to prevent the system from failing at all. More specifically, this project focuses on pinpointing the anomalies when they happen and analyzes the provided data to find the root cause of the event.

2 PROBLEM BACKGROUND AND RELATED WORK

Digitalization has come to a point where systems are fully interconnected with each other, thus relying on one another to perform as well as possible. Many systems are based on a microservice approach where each subcomponent has a very specific role inside the global work chain. Similar to an assembly line work in the 1900s, this method provides high performance because microservices are specialists at their own task while allowing a heavy global workload because all tasks can be achieved simultaneously.

As appealing as this method can appear at first glance, it comes with some incoveniences and problems as well: the major concern is cascading microservice failures resulting in a global failure. In order to prevent the system from failing, companies were originally hiring system engineers to monitor and repair the system, but with the exponential growth of web services and their various microservices, it has become impossible to monitor all of them at once. That's where AIOps comes in to save the day, using artificial intelligence recent progress to make this monitoring task less tedious, more precise and eventually faster.

Many research have been conducted to design optimal solutions to this kind of problem. Reviews of the best methods are numerous as well: various approaches have been considered throughout history, including autoencoders, to tackle the absence of clean training data issue [Zhou and Paffenroth 2017] and other unsupervised learning techniques through LSTM networks like in [Nedelkoski et al. 2019]. Clustering-based algorithms and other nearest neighbors algorithms are also considered to be very robust for anomaly detection [Chandola et al. 2009], and have even been furthered explored to develop specialized methods such as BIRCH in [Gulenko et al. 2018].

Troubleshooting an aggregate of microservices working together is a non-trivial task that can be performed through an comprehensive analysis of the trace call graph, like in [Manzoor et al. 2016], that need to be extended with root cause investigation such as [Weng et al. 2018] or [Yan et al. 2012].

3 GENERAL ARCHITECTURE

First let's examine the architecture of the web service. Composed of multiple microservices with well-defined APIs that communicate, exchange and interact with each other, the web service produces 3 different kinds of data to deal with:

- ESB business indicator data
- Trace data
- Hosts Key Performance Indicators (KPIs) data

Through a Tencent Cloud Virtual Machine, our program is able to consume real-time data retrieved from a Kafka broker, to analyze these different sources of information as well as to detect anomalies and their root cause within 5 minutes after the anomaly occurrence.
Below we report our solution to this project for the different rounds of evaluation. During the first round, we tried to make use of the ESB data in order to discover anomalies, before analyzing the hosts KPIs to retrieve the source of the anomaly. Whereas in the second and third rounds, our approach was different and, learning from the other groups' presentations, we decided to switch to a method based on the analysis of the trace data to be more reactive.

4 ROUND 1 SOLUTION

This section explicits the details of the first round solution we implemented on the server. Unfortunately the final result is zero due to compatibility issues we encountered with the environment on the Tencent Cloud VM that we did not realize was causing our program to fail. An organizational problem within the team was a major factor in this miscalculation.

4.1 Anomaly Detection using Business Index

The first step of this solution is to discover anomalies using ESB data, which can be achieved through multiple methods including an exponential moving average (EMA), and an unsupervised method specially developped for time series anomaly detection, HBOS (histogram based outlier detection [Goldstein and Dengel 2012]). Both methods are extremely easy to implement, the former being the easiest because it does not require any model to be trained, only statistical consideration are required. We decided this solution should preferably use the `avg_time` feature of the ESB data, rather than the `success_rate`, because unsuccessful business index are extremely rare and it does not help finding anomalies that much. Even though the HBOS approach showed promising results in the first place (see Figure 1), we finally opted for the EMA method. EMA can be implemented effortlessly, showed excellent results (see Appendix A) while being free from any dependencies (as opposed to HBOS which is not implemented in the standard python libraries).

4.2 Candidate Anomalous Host

Once our program correctly detected an anomaly, we needed to draw out candidate anomalous hosts for further testing and analysis. We used Trace data to complete this step by analyzing the call time of each node in the call chain graph using the `elapsed_time` of each node.

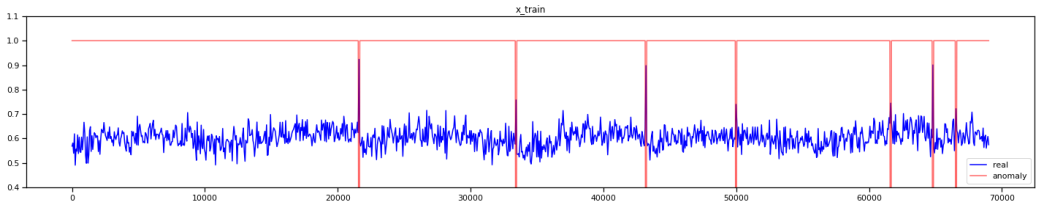


Fig. 1. HBOS (histogram based outlier detection) showed promising initial results, being able to successfully discover major ESB data anomalies.

4.3 Anomalous Key Performance Index Retrieval

Originally, this step of retrieving the anomalous KPIs from the anomalous node was also trying to find the root cause. Finding anomalous behaviours in KPIs would lead to a ranking of the most "anomalous" behaviours that would later be flagged as the source host of the anomaly. However the results did not follow and we found that having the root cause analysis step provided better results.

The core idea behind this first round's solution is to use machine learning models to predict KPIs values and compared them to the received KPIs values from the server. The difference between the true and the predicted value that would force the KPI into being considered anomalous, is measured using a predefined threshold on an error function. We tried various error functions, including: mean absolute error (MAE), mean relative error (MRE), mean squared error (MSE), binary cross-entropy.

The proper functioning of this step essentially resides in the choice of the model that will make such prediction. Since the problem deals with Time Series, there are many models indicated for this type of use including statistical methods (such as ARIMA and its multiple variants [Moayedi and Masnadi-Shirazi 2008]), decision trees (such as XGBoost and LigthGBM) or deep learning (such as LSTM based neural networks and other deep learning techniques [Kim and Cho 2018]). Additionally, the methods detailed below require some data transformation and feature engineering. Incoming data from the server is modified into a "supervised dataset" where the input has columns for each KPI time series, further divided into past values (at times t-1min, t-2min, ...) and output has one columns for each KPI time series with only the current value (t). In order to tackle the missing values issue, we decided to go for forward-fill imputing because every KPI has different update frequency and we considered missing values as an absence of update, i.e. the value has not changed up until now.

LSTM. Long Short Term Memory (LSTM) cells are optimal for time series prediction, in fact, a unique cell is capable of forecasting a time series with very high accuracy. Therefore using such a model on KPIs seems appropriate and we decided to assign one model to each host with each model composed of n independent cells in parallel, n being the number of KPIs describing the given host (see Figure 2). However, because LSTM need supervised datasets as input, we have to transform the time series dataset beforehand, which can be achieved through a succession of pandas operations on the data.

In addition to this design, we also tried various LSTM architecture including Vanilla LSTM, stacked LSTM, CNN together with LSTM, but the single cell design was sufficient. LSTM time series forecasting showed promising results that can be improved with further tuning and analysis (see

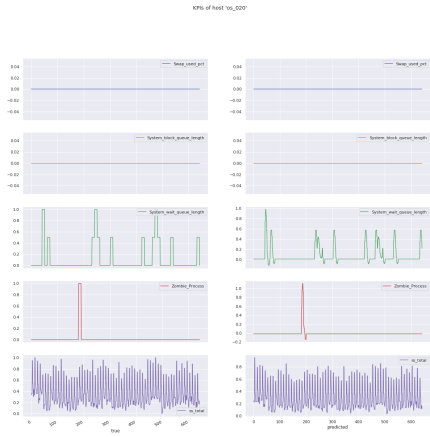


Fig. 2. Architecture of a LSTM model to forecast KPIs values of a given host (true/predicted)

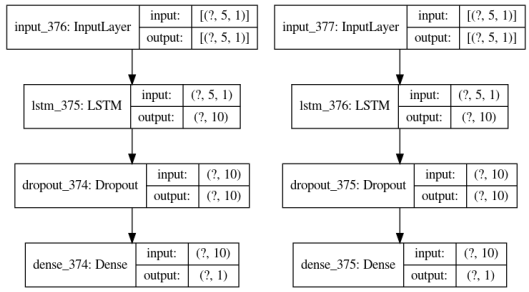


Fig. 3. KPIs: Time series forecasting using LSTM cells

Figure 3). But this method was designed to work hand in hand with the candidate anomalous host step, that was lacking efficiency when it was implemented. In the sake of time and of simplicity, we decided to abandon this approach in the next rounds.

XGBoost. XGBoost is an extremely popular algorithm to solve structured supervised problems that have recently proven to be particularly effective, and rather simple to implement. Again, the given problem does not label exactly as "supervised problem", so we had to transform the data to fit xgboost input. Adapting xgboost to the given problem was also challenging because of the strategy to adopt: how many models do we need? Since xgboost cannot handle categorical data well [Chen and Guestrin 2016] (e.g. host name, KPI name…), do we need to have a model for each? We came up with 2 solutions: use a multioutput wrapper for xgboost to support all hosts and KPIs at once, or switch to LightGBM. Though we tried the multiple xgboost models approach as well as the multiouput single xgboost model approach, the results were not as promising as LSTM. Figure 11 in Appendix B shows a sample of predicted kpis using the multioutput xgboost model where some predictions are not that accurate.

5 ROUND 2 SOLUTION

We worked on our own to design rule-based algorithms mainly analysing trace data to find the anomalous nodes and the KPI affected. Due to a lack of communication from Fabien, we came up with two slightly different approaches.

5.1 Aurelien's implementation

This time around anomalies are detected using trace and host KPIs data only because the business index data has a low frequency update that can make the program skip anomalies. Inspired by the best solutions method, we switched back to a rule-based model. A set of rules analyzes the trace data using different engineered features including: the `elapsed_time` (that is natively inside the trace data), the `call_time` of hosts (i.e. the time spent in edges of the trace call graph), the position of hosts in the call graph (child, parent, cousin) and their own features.
Our program first compares the `elapsed_time` of nodes with all-time recorded values and flags the host as anomalous if its value goes over a certain threshold (the 99.99% quantile). Then a second layer of tests using a z-score analysis inside a 20 minutes moving window ($zscore = (x - mean)/std$) can detect anomalies by comparing the computed z-score to a specific threshold determined using the training data. Since the z-score gives an estimation of how far a value inside a series is from its mean (more specifically, how many standard devation away this value is from the mean), this indicator is easily interpretable and delivers good results.
Afterwards, there is a series of specific tests for particular configuration and well-defined anomalies such as "db close" error, "db connection limit" or "os network delay". These errors have associated KPIs that are easily tractable: `Sent_queue` and `Received_queue` for the os anomalies ; `On_Off_State` and `tnsping_result_time` for the db faults. Sudden peaks have tremendous influence on the system and should be considered outliers. We monitored such peak events through a simple upper bound threshold determined using the training labeled data.
Then comes the engineered features. The core idea behind this approach is to find patterns in the fixed call chain graph that induces faults and errors. Figure 4 shows how microservices call each other. This allows manual analysis using the labeled data to discover anomalous patterns, including:

- high `call_time` often implies a network issue (delay or loss), depending on the status of KPIs `Sent_queue` and `Received_queue`
- increasing incoming and outgoing `call_time` often translates as an intern problem (such as CPU usage issue)
- ...

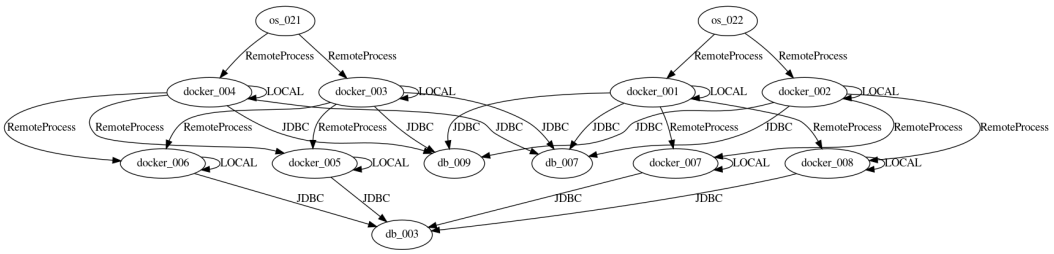


Fig. 4. The web service architecture has a fixed call graph.

5.2 Fabien's implementation

This solution is aimed to be working on the training and testing data provided, and has not yet been implemented on the Tencent VM.

Microservice Architecture. As microservice-based softwares grow in popularity, such architecture generates new problems, and thus new approaches are needed to manage them. A microservice is a small scalable software service which is part of a larger application. The intermittent communication between the services makes it difficult to identify the root cause of a problem, so it is important to first study the specific architecture of the application and visualize the connections between microservices, before addressing log anomaly detection and location.
After running some tests on the trace data, we can understand the relations between the different types of calls (see Figure 5 and Figure 6). For example, a span with a Remote Process call type can only have span children with Fly Remote, CSF or Local call types. Furthermore, without much surprise, only the Remote Process spans have a parent span on a different host. More importantly, we can analyze the calls between nodes to assert their relations, which will be important to address network anomalies for example.

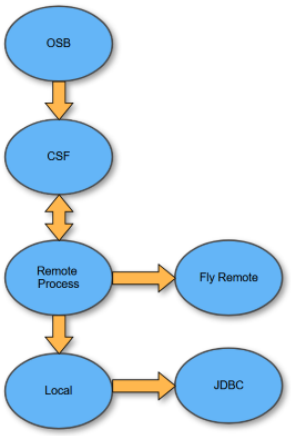


Fig. 5. The hierarchical relations between call types.

Anomaly detection with ESB. We kept using ESB business rate indicators to tackle the anomaly detection problem. Although the number of submitted requests and the success rate are not reliable enough to find all anomalies, the average time spent processing a request always matches the upcoming of an anomaly. Moreover, it doesn't induce much latency compared to the evolution of trace global processing time.

Exponential moving average is a great tool that detects the appearance of an anomaly down to the minute (see Figure 9). It takes some time to stabilize after an anomaly, thus resulting in an increase of the expected duration of the anomaly. However, this is not a problem since the anomalies duration is 5 minutes long and the continuation of the algorithm only requires 2 minutes of anomalous traces.

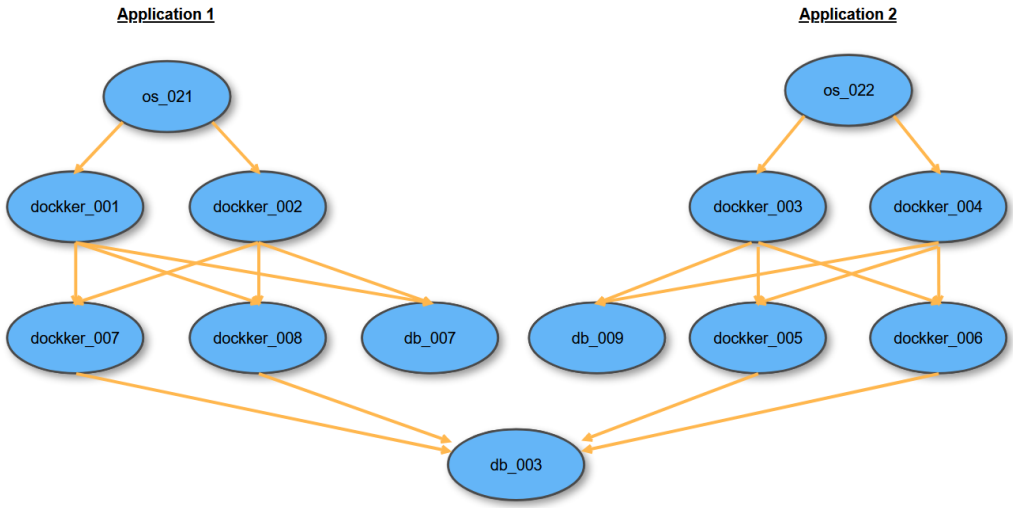


Fig. 6. The hierarchical relations between nodes.

Data preprocessing. A host can be considered anomalous if the running time of its spans is very low or high. In order to obtain the true running time of a microservice on its host, we first need to retrieve the children spans elapsed time from its own elapsed time. For that purpose, we group trace data according to trace id and sort spans according to their start time to easily find their parent.

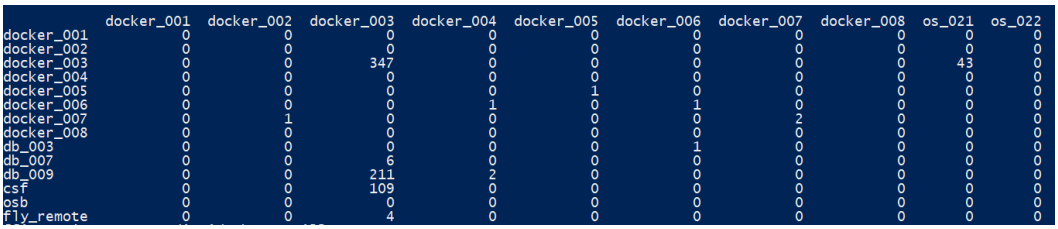
Defining abnormal call times. We can build a matrix matching the hosts (columns) with the instances they call (rows), and fill it with the percentile of the running times of the corresponding spans. Setting a threshold for each type of call, we are able not only to detect the anomalous hosts, but also the very nature of the calling problem. This matrix is conceived on non anomalous trace data in order to set coherent thresholds, we built several of them with different percentiles and compared them later with the anomalous traces.

	docker_001	docker_002	docker_003	docker_004	...	docker_007	docker_008	os_021	os_022
docker_001	199.000	0.000	0.000	0.000	...	0.000	0.000	0.000	1766.168
docker_002	0.000	193.147	0.000	0.000	...	0.000	0.000	0.000	1398.440
docker_003	0.000	0.000	198.000	0.000	...	0.000	0.000	1512.050	0.000
docker_004	0.000	0.000	0.000	194.000	...	0.000	0.000	1083.200	0.000
docker_005	0.000	0.000	70.000	63.932	...	0.000	0.000	0.000	0.000
docker_006	0.000	0.000	82.924	76.000	...	0.000	0.000	0.000	0.000
docker_007	73.031	56.000	0.000	0.000	...	16.876	0.000	0.000	0.000
docker_008	78.316	74.844	0.000	0.000	...	0.000	12.922	0.000	0.000
db_003	0.000	0.000	0.000	0.000	...	11.000	11.000	0.000	0.000
db_007	94.000	91.879	100.000	98.000	...	0.000	0.000	0.000	0.000
db_009	92.000	90.000	93.000	93.000	...	0.000	0.000	0.000	0.000
csf	359.763	312.009	357.897	366.895	...	0.000	0.000	1552.422	1912.472
osb	0.000	0.000	0.000	0.000	...	0.000	0.000	1280.030	1184.967
fly_remote	99.000	124.600	198.475	148.788	...	0.000	0.000	0.000	0.000

Fig. 7. The 99.9th percentile matrix.

Anomaly location. For each anomaly detected, we select two minutes of anomalous trace data already preprocessed. We initialize another matrix matching the hosts and call types and use it to count the anomalous spans. A span is considered anomalous as soon as its calling time exceeds the percentile thresholds defined on the non anomalous training trace data.

As there are very few different types of anomalous KPIs we can try to deduce from the matrix of abnormal spans both the host and type of affected KPI according to different distribution scenarios. The easiest type of failing KPI to detect is `container_cpu_used` which triggers anomalous local calls in docker hosts. On the other hand, the network problems affecting docker hosts is detected if a docker instance has huge remote process running time.



	docker_001	docker_002	docker_003	docker_004	docker_005	docker_006	docker_007	docker_008	os_021	os_022
docker_001	0	0	0	0	0	0	0	0	0	0
docker_002	0	0	0	0	0	0	0	0	0	0
docker_003	0	0	347	0	0	0	0	0	43	0
docker_004	0	0	0	0	0	0	0	0	0	0
docker_005	0	0	0	0	1	0	0	0	0	0
docker_006	0	0	0	1	0	1	0	0	0	0
docker_007	0	1	0	0	0	0	2	0	0	0
docker_008	0	0	0	0	0	0	0	0	0	0
db_003	0	0	0	0	0	1	0	0	0	0
db_007	0	0	6	0	0	0	0	0	0	0
db_009	0	0	211	2	0	0	0	0	0	0
csf	0	0	109	0	0	0	0	0	0	0
osb	0	0	0	0	0	0	0	0	0	0
fly_remote	0	0	4	0	0	0	0	0	0	0

Fig. 8. The docker 3 calls on itself are anomalous: container CPU used error.

6 WORK ALLOCATION

	Aurelien VU NGOC	Fabien RAVET
round 1	HBOS, xgboost, LSTM	EMA for esb anomaly detection
round 2 & 3	Rule-based model (implemented in Tencent VM)	Another rule-base model (not yet implemented in Tencent VM)
other	This report Presentation Tencent VM code design	Presentation

Table 1. Project work allocation

An overview of the work allocation can also be found in the GitHub commit history at the following url: https://github.com/DANM-team/ANM-project.

7 CONCLUSION

This study aims to implement a troubleshooting algorithm for a microservices-operated web service. We developped the many solutions we tried during the semester, discussed the different approaches and their pro & cons, as well as analyzed the issues we faced. Our final solution is a rule-based approach to outlier detection that uses a set of directives, preliminarily determined using the provided labeled training data, to discover anomalies. It is helped by engineered features as well as specific hard coded patterns to perform accurate and fast anomaly detection.
All in all, though we wished for better results in the final test, conducting this project has been an unprecedented opportunity to work with real data coming straight from influencial real-life service. The many research and testing that have staked out our journey have been a priceless chance to understand the challenges and major concerns in AIOps. Finally we believe the ANM project has been a real occasion to put all the knowledge accumulated throughout the semester into practice, and more importantly, to let this practical and truly valuable experience serve future problems.

REFERENCES

Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *Comput. Surveys* 41, 3 (July 2009), 1–58. https://doi.org/10.1145/1541880.1541882

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug. 2016), 785–794. https://doi.org/10.1145/2939672.2939785 arXiv: 1603.02754.

Markus Goldstein and Andreas Dengel. 2012. *Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm.*

A. Gulenko, F. Schmidt, A. Acker, M. Wallschläger, O. Kao, and F. Liu. 2018. Detecting Anomalous Behavior of Black-Box Services Modeled with Distance-Based Online Clustering. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 912–915. https://doi.org/10.1109/CLOUD.2018.00134 ISSN: 2159-6190.

Tae-Young Kim and Sung-Bae Cho. 2018. Web traffic anomaly detection using C-LSTM neural networks. *Expert Systems with Applications* 106 (Sept. 2018), 66–76. https://doi.org/10.1016/j.eswa.2018.04.004

Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1035–1044. https://doi.org/10.1145/2939672.2939783

H. Zare Moayedi and M. A. Masnadi-Shirazi. 2008. Arima model for network traffic prediction and anomaly detection. In *2008 International Symposium on Information Technology*, Vol. 4. 1–6. https://doi.org/10.1109/ITSIM.2008.4631947 ISSN: 2155-899X.

S. Nedelkoski, J. Cardoso, and O. Kao. 2019. Anomaly Detection from System Tracing Data Using Multimodal Deep Learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 179–186. https://doi.org/10.1109/CLOUD.2019.00038 ISSN: 2159-6190.

J. Weng, J. H. Wang, J. Yang, and Y. Yang. 2018. Root Cause Analysis of Anomalies of Multitier Services in Public Clouds. *IEEE/ACM Transactions on Networking* 26, 4 (Aug. 2018), 1646–1659. https://doi.org/10.1109/TNET.2018.2843805 Conference Name: IEEE/ACM Transactions on Networking.

H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates. 2012. G-RCA: A Generic Root Cause Analysis Platform for Service Quality Management in Large IP Networks. *IEEE/ACM Transactions on Networking* 20, 6 (Dec. 2012), 1734–1747. https://doi.org/10.1109/TNET.2012.2188837 Conference Name: IEEE/ACM Transactions on Networking.

Chong Zhou and Randy C. Paffenroth. 2017. Anomaly Detection with Robust Deep Autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax NS Canada, 665–674. https://doi.org/10.1145/3097983.3098052

A EMA RESULTS

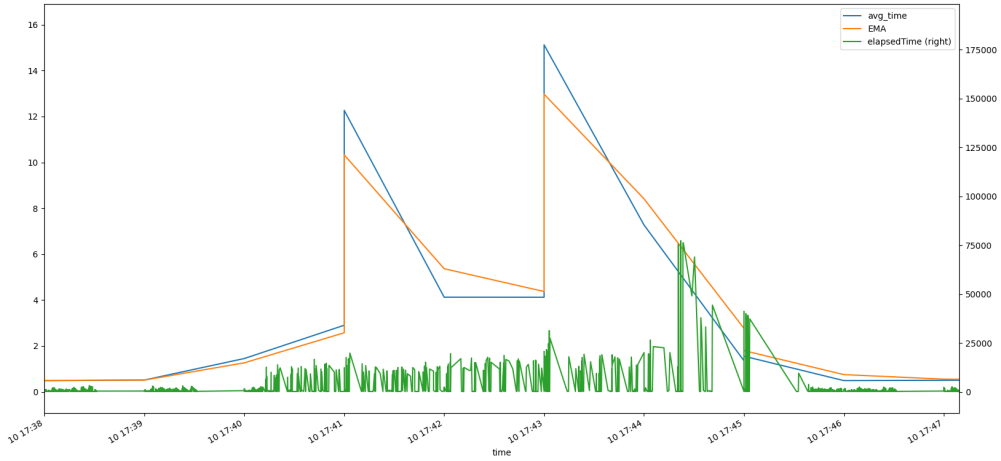


Fig. 9. The exponential moving average present dissimilarities on the anomaly.

	start	end
0	2020-04-10 16:05:00	2020-04-10 16:14:00
1	2020-04-10 16:35:00	2020-04-10 16:41:00
2	2020-04-10 17:11:00	2020-04-10 17:16:00
3	2020-04-10 17:40:00	2020-04-10 17:47:00
4	2020-04-10 18:16:00	2020-04-10 18:21:00
5	2020-04-10 18:50:00	2020-04-10 18:58:00
6	2020-04-10 19:21:00	2020-04-10 19:26:00
7	2020-04-10 19:56:00	2020-04-10 20:03:00
8	2020-04-10 20:41:00	2020-04-10 20:47:00
9	2020-04-10 21:05:00	2020-04-10 21:13:00
10	2020-04-10 21:50:00	2020-04-10 21:51:00

Fig. 10. EMA promising results: the timestamps of occurring anomalies exactly correspond to given anomalies in the training data.

B XGBOOST RESULTS

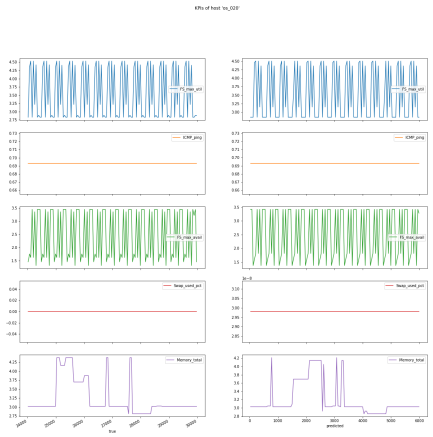


Fig. 11. Sample of Multioutput xgboost forecasting KPIs of *os*020 (true/predicted)