# ANM project: DANM! team

AURELIEN VU NGOC, Tsinghua University, China

TODO: abstract, conclusion, related work and insert some figures + furnish round 2 solution part

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

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

## 1 INTRODUCTION

This project aims at designing 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 relation 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 a potential anomalies before making precise and quick decisions to help the system recover, and also further developping abilities to recognize potential threats to prevent the system from failing at all. More specifically, this project focus on pinpointing the anomalies when they happen and analyze 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 good 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 work load because all taks can be achieved in parallel.

As appealing 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 grow 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.

## 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 source of information as well as to detect anomalies and their root cause as fast as at most 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 the second and third rounds, our approach was different and rich in the other groups' presentation, 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 realized was causing our program to fail. An organizational problem with the team members 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 doesn't help finding anomalies that much. Even though the HBOS approach has proven promising results in the first place, we finally opted for the EMA method. EMA can be implemented effortlessly 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 need to draw out candidate anomalous host for further testing and analysis. We use 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.

### 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), decision trees (such as XGBoost) or deep learning (such as LSTM based neural networks). Additionally, the methods discussed 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 hasn't changed up until now.

Fig. 1. Architecture of a LSTM model to forecast KPIs values of a given host (true/predicted)

*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 and composed of n independent cells in parallel, n being the number of KPIs describing the given host (see Figure 1). 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 operation on the data. In addition to this design, we also tried various LSTM architecture including Vanilla LSTM, stacked

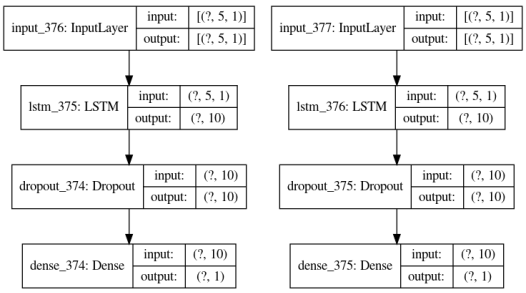


Fig. 2. KPIs: Time series forecasting using LSTM cells

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 Figure 2). But this method was designed to work hand in hand with the candidate anomalous host

step, that was lacking efficiency at the moment 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 particularily 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 4 in Appendix A shows a sample of predicted kpis using the multioutput xgboost model where some predictions are not that accurate.

## 5 ROUND 2 SOLUTION

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 switch 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 spend 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 monitor 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 3 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)
- …

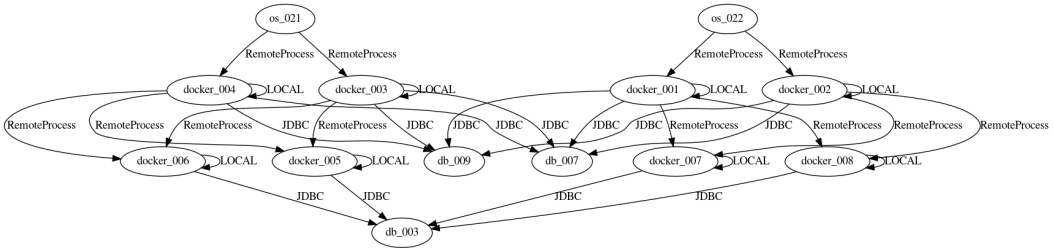


Fig. 3. The web service architecture has a fixed call graph.

## 6 WORK ALLOCATION

| | Aurelien VU NGOC | Fabien RAVET | Aleksander |
|---|---|---|---|
| round 1 | HBOS, xgboost, LSTM | EMA for esb anomaly detection | |
| round 2 & 3 | Rule-based solution | | |
| other | This report, Tencent VM code design | | |

Table 1. Project work allocation

An insight 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 at implementing a troubleshooting algorithm for a microservices-operated web service. We developped the many solution we tried all along 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 researches 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

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 XGBOOST RESULTS

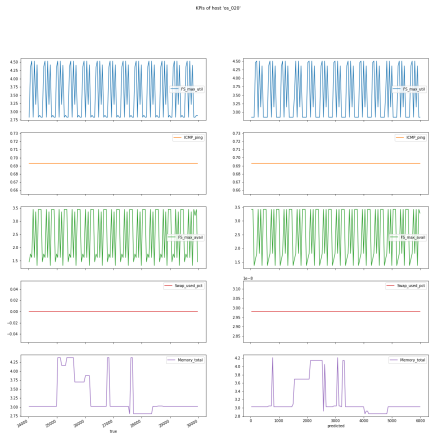


Fig. 4. Sample of Multioutput xgboost forecasting KPIs of *os*020 (true/predicted)