Learning Failure: Finding the root cause of anomalies in a real-world production environment

Tiago Antunes
Tsinghua University
vazama10@mails.tsinghua.edu.cn

Héctor Martel
Tsinghua University
hkt20@mails.tsinghua.edu.cn

Malte Meng
Tsinghua University
mengxr20@mails.tsinghua.edu.cn

Abstract

The goal of this work is to build a system that can detect anomalies in a microservice architecture, as per the AIOps challenge. We initially assimilate this problem to a general purpose semi-supervised anomaly localization setting. We explore and implement various methods from the AIOps research literature, adapting them to the needs of our problem, and make observations about their effectiveness in our use-case scenario. After some unfruitful attempts, we further analyze the data to incorporate domain knowledge in our final solution. The new simplifications made the final solution faster and more accurate, although environment limited.

1 Problem definition

Under the scope of the AIOps challenge, we are given 3 types of data from microservice system and we are required to detect anomalies timely when they occur. The data contains ESB, business indicators that provide a high-level view of the system status, traces spans which describe the different call paths and individual response time of each microservice, and KPIs of each host in which the services are running. For each detected anomaly, the system must submit a pair containing the *(host, kpi)* found to be the root cause. The performance is evaluated according to the correctness of the detections as well as the time taken to find the root cause.

The execution environment is a virtual machine from Tencent Cloud with limited resources and no available GPU. Therefore, we prioritize the computational cost of all the approaches presented in this work, even at the expense of degrading the methods performance. Many proposed solutions that use Machine Learning operate outside the band of the system similar to how information is served via Kafka. Cloud Processing systems provide both the power and the information for these approaches, but in our case, due to having low resources, performance is a requirement to make sure we can detect anomalies timely.

As to evaluate the performance of our models, and given the labeled data available, we extracted time windows from the tests as to make a collection of smaller tests that could easily be run to evaluate the precision and performance. For this, we set up a local Kafka Environment and created a server that streams the ESB, Trace and KPI data into our consumer and this one can execute as if in the official environment. We also decided to ignore the time of each message and just send the message when available, allowing the speed of testing to be superior to the real-time of the test. All the code that was developed is available in our GitHub repository and its organization is described in the README.

Final Project for Advanced Network Management, Master in Advanced Computing, Tsinghua University (2020-2021).

This report is structured as the following: in section 2 we describe our different approaches and our difficulties when implementing each one of them; in section 3 we describe our final approach; in section 4 we express some of the complications we found, our discoveries, what was learned during this challenge and about the mistakes we made when trying out different approaches.

2 Attempted approaches

In this section we intend to provide the motivation and the lessons learned after trying several approaches from the AIOps research literature.

We are particularly interested in how these approaches are adapted to our problem and what are the underlying reasons of them being unfeasible or unsuccessful. We hope that our efforts will serve as a reference to the readers who consider implementing these same approaches in the future to avoid running into the same issues again.

2.1 System architecture overview

At the beginning of this project, we considered that a logical structure of the system would be a 3-stage processing pipeline. At each stage we can decide whether or not to continue processing based on a result evaluation. The main purpose of establishing this hierarchy is to avoid the computation of unnecessary steps when the system receives normal operation data.

The first stage is the anomaly detection of the Business Index (ESB), which provide a global status of the microservices success rate, the number of requests and the average time it takes to process them. Given that this data is sent at a low rate (once every minute), performing anomaly detection on the ESB is a very lightweight computation that serves to filter out the normal data. At this point, the only decision to be made is whether or not there is an anomaly, without any other considerations about the cause.

When an anomaly is detected in the ESB data, the Trace data is passed to the second stage to perform Trace Root Cause Analysis. The method analyzes the trace data within a time window of 5 minutes before the anomaly was detected in order to build a representation of the traces. This representation largely depends on the specific implementation, and it can be a high dimensional vector or a graph. After the analysis the output is a list of candidate hosts that present an anomalous behavior, typically with an associated score that indicates the likelihood of a particular host being the root cause of the anomaly.

In case that there is one or multiple hosts in the candidate list, we proceed to pass the KPI data to the last stage to perform KPI Anomaly Detection. Having a non-empty list of host candidates implies that there will be a submission even if it is not possible to obtain a precise KPI as the cause. For each host, we consider each KPI as being an independent time series and compute time series anomaly detection to find the metric that is the most likely cause of the system failure. In contrast to the trace data, here we use a time window of the 60 minutes. This decision is motivated both by the low amount of KPI information (each unique KPI had a rate of 1 minute or 5 minutes) and also due to the fact that some approaches required a minimum sized time window for it to run.

The flowchart in Figure 1 illustrates the decision process and the role of all the stages in the proposed system.

2.2 ESB Anomaly Detection

The primary source of information about the system status comes from the ESB. The updates are received periodically every minute and contain information about the global system operation. Some valuable insights can be obtained by analyzing the time evolution of 3 variables, corresponding to the number of requests that are processed, their average times and the relative number of successful requests.

We start by naïvely considering anomalous any point in time in which the relative success rate of the requests is smaller than 1.0. This assumption, despite obvious, is also very effective to save time while flagging anomalies at this point of the processing pipeline. A key observation is that we need to

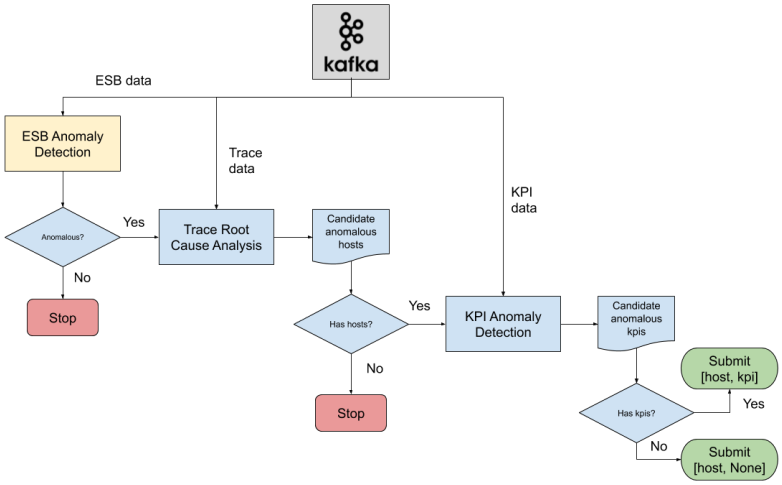


Figure 1: Overview of the data processing pipeline.

optimize the system for a high recall, since it is crucial not to miss an anomaly when it occurs. If there is a false positive it can be discarded after further processing.

The success rate is not the only indicator of a possible anomaly in the system. For example, there may be some scenarios in which the anomaly causes the requests to take longer to process, or the volume of requests to experience sudden increase or decrease, but without the requests being unsuccessful. We tackle this by further analyzing the ESB data in terms of number of requests and average time.

2.2.1 Variational Auto Encoders (VAE)

The use of unsupervised learning methods such as Variational Auto Encoders (VAE) is common for time series anomaly detection [2]. They can learn the structure of unlabeled data by means of a latent representation from which it can be reconstructed. We find this characteristic to be very suitable to be applied to our problem, as we are provided with the ESB data without labels.

Some progress has been done also in multivariate time series [7], where the objective is to capture the dependencies that may exist between different series in the data.

The advantage of using a VAE over other unsupervised anomaly detection methods such as Support Vector Machines (SVM), K-Means or Gaussian Mixture Models, is that sampling from the latent representation provides a robust reconstruction that considers both the temporal dependence and the stochasticity of the time series data. After obtaining a reconstruction, the method can determine whether or not it deviates from the normal pattern. The detections of our implemented VAE over a fragment of the average time test data are displayed in Figure 2.

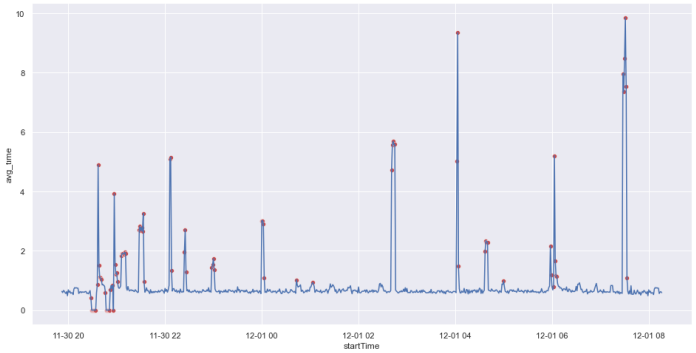


Figure 2: Variational Autoencoder anomaly detections (red dots) on the average time.

2.2.2 Heuristic Methods

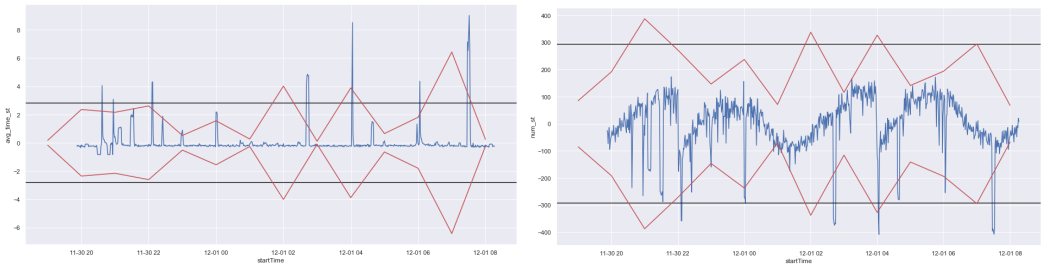
In order to alleviate the computation of the ESB anomaly detection stage, we have opted for a heuristic approach that is significantly cheaper to compute. We use statistical measures obtained from the data to calculate a threshold value, and declare any point that crosses that boundary to be anomalous.

A traditional way to calculate this threshold is to assume that the training data follows a normal distribution. The data needs to be further transformed to have zero mean and unit standard deviation, also called standardization. For each data point $x_i \in X$, the data mean $\overline{x}$ is subtracted and the result is then divided by the standard deviation σ, thus obtaining $\hat{x_i}$.

$$\hat{x_i} = \frac{x_i - \overline{x}}{\sigma}$$

The threshold is determined to be the value for which 99.7% of the data points are considered normal, namely, the boundary is located 3 standard deviations away from the mean.

Figure 3 shows the ESB data with the threshold values overlayed. Figure 3a has the standardized average time and Figure 3b the number of requests. In both cases, the figures display the data extracted from the test set. The fixed threshold is a single value obtained after processing the entire training dataset, whereas the rolling window threshold is obtained considering time windows of 60 minutes, i.e. 60 ESB data points.



(a) Standardized average time.

(b) Standardized number of requests.

Figure 3: Fixed and rolling window threshold detection on average time and number of requests. Test data (blue line); Fixed threshold (black horizontal line); 60-min rolling window threshold (red line).

We finally compare the Heuristic method and the VAE and we conclude that the accuracy of both approaches was similar, and provided detection of ESB anomalies with an acceptable error rate. The VAE, however, required more computation and due to the limitations of the hardware, it was considerably slower than the heuristic method, and thus we opted out for the latter.

2.3 Trace Root Cause Analysis

The trace root cause analysis part of the system is arguably the most critical component. The purpose of this part is to detect the anomalies present in the trace data and obtain one or more candidate hosts to be the root cause of the problem. One of the challenges here is that, once an anomaly happens in the system, the effects can propagate to other nodes due to the strong interdependence. Thus, the goal is to find the origin and be able to filter out the noisy data caused by the propagation. We study 2 approaches, namely TraceAnomaly and MicroRCA, and implement a modified version MicroRCA that we thought fitted our problem needs.

2.3.1 TraceAnomaly

The authors of TraceAnomaly [4] propose the use of Deep Bayesian Networks to analyze and localize anomalies in trace data. The main idea is to construct a vector representation called Service Trace Vector (STV) of the trace using the response time and the invocation paths. A call path is defined as the sequence of call messages ordered by time, that happen before a particular service is called. That is, the sequence from the root to that service. By combining these response times and call path into a compact representation, it is possible to learn both service-level and trace-level patterns to perform the anomaly detection and localize the root cause.

Another advantage of the method lies on it being the first machine learning trace anomaly detection approach to be deployed in a real production environment. We believe that this fact makes the model particularly suitable to be adapted to our problem, as we need to handle a large amount of real-time data. Moreover, the model can be trained in an unsupervised setting, which is a desirable characteristic provided that we have few or no labels on the training dataset.

TraceAnomaly starts by considering the trace data and passing it through a preprocessing stage to construct the STVs. To perform the model training, this process is repeated for all the traces present in the training dataset and the resulting vector representations are fed into the neural network. In case of online inference, the trace is processed to obtain its corresponding STV, and then passed to the anomaly detection stage. There is an additional step in the middle which is accounts for unseen call paths. The systems keeps track of historical data and declares an anomaly when the input trace contains a new unseen call path, to be handled in a whitelist, which was not required in our case. The model outputs a likelihood (typically log-likelihood) of the trace vector being an anomaly, which is interpreted as its anomaly score. Depending on its value, the trace is considered anomalous and analyzed to find the root cause.

Both training and inference are performed regularly to account for changes in the trace data structure after system upgrades, although we do not consider this periodic retraining of the system to be particularly helpful in our problem due to the data, services and calls remaining unchanged during the challenge. The architecture diagram of the whole system is shown in Figure 4.

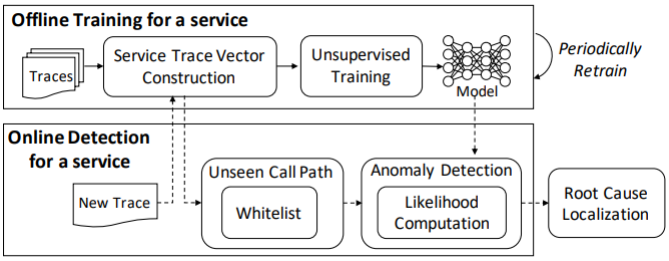


Figure 4: Architecture diagram of TraceAnomaly. (Extracted from [4]).

Despite the approach looking promising at first, we found that the STVs were generated with a very high dimensionality. We found the required dimensions to be 1.5 million, and managed to reduce them to 9000 with some data transformation. However, this STV size was still large and clearly made it unfeasible to apply the model in our case. A critical limitation is that our server does not have a GPU, so that any deep learning approaches would take a long time to perform the computations and even its training would take a long time in dedicated hardware.

2.3.2 MicroRCA

In contrast to the previous approach, MicroRCA [8] uses an attributed graph representation to obtain the anomalous service given a collection of traces. The idea is to construct a graph containing the services and hosts to capture the anomaly propagation. The authors claim that their method is effective to identify anomalies that not only propagate along the service call paths, but also propagate because they share resources on a physical or a virtual machine.

The first step is to build the attributed graph representing all the services and hosts as nodes. A directed edge connecting the parent to the child node is added between a pair of services that communicate in a span, or between a service and the host it runs on. Thus, we represent all parent-child relationships present in the trace data.

The trace data from a certain time frame before the anomaly is detected is considered to be the normal pattern. The anomaly detection is performed for each node in the graph using the node response time and assigning it to a cluster using the BIRCH algorithm [10], which is efficient and works in an online fashion.

In our implementation, we decided to use the node elapsed time instead of the response time as an optimization. We observed that, since the anomaly propagates from children to parent node, it is sufficient to run the clustering on the root node of each trace to determine if the entire trace is

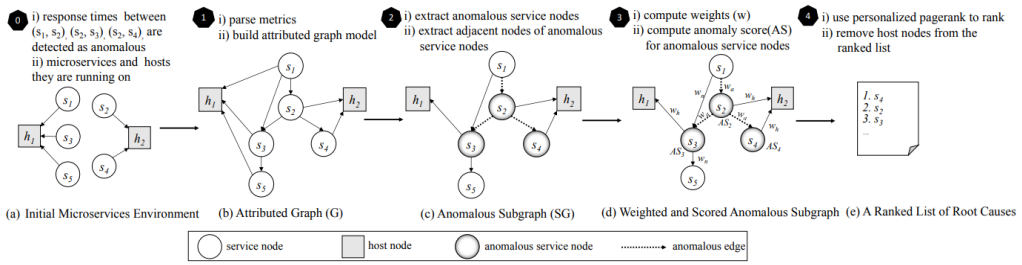


Figure 5: Steps of the MicroRCA root localization procedure. (Extracted from [8]).

anomalous. In case of finding an anomalous trace, we proceed to the by-node anomaly detection. This differs from the original paper in that the authors consider by-edge detection instead of by-node detection. The change is motivated by the fact of computing the elapsed time of the nodes, as opposed to the response time, which was much easier to calculate from the structure of our data.

After each node has been classified, a connected subgraph containing the anomalous nodes is extracted from the original graph to obtain a reduced set of nodes that represent the propagation of the anomaly. Therefore, the graphs contains all the anomalous nodes, their parents, and the edges connecting either 2 anomalous nodes or an anomalous and a normal node.

Once the anomalous subgraph is extracted, each edge is assigned a weight according to the similarity between the pair of nodes it connects. It must be noted that, in this case, the value of the weights represents the importance of the connection rather than a cost. A high weight path is preferred over a low weight one. The weighting of the anomalous subgraph follows these 3 rules:

- Anomalous edges. An edge is anomalous if both source and destination are anomalous nodes. In this case, a constant value $\alpha \in (0, 1]$ is assigned to be the weight of the edge. The choice of this constant is related to the anomaly detection confidence. In implementation, we follow the author's guidelines and set $\alpha = 0.55$.
- Anomalous service to normal service. The weight is computed as the correlation between the response time of the pairs of nodes. Therefore, a high correlation in response times indicates that the normal node is affected by the anomalous one.

  Note: In the paper, the authors state that they use the average response time in the computation of the correlation. To the best of our knowledge, using a single value instead of a series is conceptually wrong and should be avoided. Our implementation uses the values of the response times directly instead of the mean to compute the correlation.
- Anomalous service to a normal host. The weight assigned to the edge corresponds to the maximum correlation between the anomalous service response time and the host KPIs. The idea is to obtain the resource utilization that has the highest similarity with the anomaly detected in the service.

The final step as described in the paper is to localize the most likely root cause of the anomaly by running Personalized PageRank [1] on the weighted anomalous subgraph. The algorithm requires a personalization vector to be calculated before iterating over the graph. This vector represents the prior root cause score for each node, and it is computed as the normalized outbound edges weight of the nodes that are connected.

$$\begin{cases} P_{ij} = w_{ij} / \sum_j w_{ij} & \text{i is connected to j} \\ P_{ij} = 0 & \text{otherwise} \end{cases}$$

The PageRank algorithm is run using the personalization vector with a teleportation probability $c = 0.15$.

The output of the algorithm is a list of candidate nodes with their respective rank, which we interpret to be the anomaly score of the node. Since the original MicroRCA paper's approach is concerned about detecting faulty *services*, the hosts are removed from the list after ranking. However, in our

case the objective is to detect the anomalous *hosts*. We discovered that simply trying the opposite by removing the hosts did not provide accurate results. After further analysis, we consider the weighted contribution of all the anomalous services and the host they are connected to. The host is selected as the one with maximum weighted score across all hosts.

After implementing our model, we concluded that its performance was lacking. It was not being able to accurately localize the anomalous hosts and it was being very slow in doing so (mostly due to the usage of pandas). We also noticed that, opposing to what the original paper had, self-connections were causing our results to be different every run. After removing them, we obtained not only a deterministic score, but also some scores that made more sense. Some further analysis is done in section 4.

2.4 KPI Anomaly Detection

After the trace root cause analysis is complete, a list of host's would be passed to the KPI anomaly detection system. Using a single or multiple host's, we would then run anomaly detection on each of the contained KPI's. Optimally we would discover a single or few anomalous KPI's and then return the suspect Host x KPI pairs for submission. We first analyse the provided KPI to try and gauge data features and characteristics. After doing so we tried two approaches, anomaly detection with variational auto-encoders and spectral-residue mapping. Since neither proved effective, we also studied ROCKa as a possible method to cluster KPI's and LOUD to detect the propogation of anomalies throughout the system.

2.4.1 Data Analysis

The first step we took related to the KPI data was an exploratory data analysis. Here we manually inspected the data, trying to determine useful features and characteristics. The first discoveries were that there were a total of 2000+ unique KPI's being measured. This led us to believe that training any ML model on individual KPI's would be impractical. A significant portion of these KPI's were also flat (had a zero standard deviation) over the 24 hours of provided training data. We decided to first omit them from our anomaly detection and possibly report anomalous behaviour if the standard deviation ever changed. The intervals in which we received the data was also variable and could be broken down into four frequencies every minute, every two minutes, every 5 minutes, or at most every 30 minutes. We also discovered that many of the KPI's were dropping data at some points.

Host Type	KPI	Interval	Flat
os_linux	Processor_load_5_min	1min	False
db_oracle_11g	ACS	1min	False
db_oracle_11g	Sess_Used_Temp	2min	False
os_linux	Memory_total	>=30min	False
dcos_docker	container_thread_total	1min	True.

Table 1: Sampled extracts from KPI summary data

To determine what the data looked like, we randomly sampled from the KPI's and generated line plot visualisations. KPI's related to the CPU would consist of large jumps without any seeming relation. KPI's related to disk or memory usage tended to stay unchanged or increase / decrease slightly. Looking at these graphs we came to the conclusion that a purely heuristic approach would be infeasible. Doing so would require an enormous amount of manual labour and since we were treating the problem as an unsupervised one, we couldn't be sure that it would work.

2.4.2 Variational Auto-Encoders

In the sampled KPI's we visualised, there seemed to be discoverable patterns. Taking the idea presented by [9] we decided to train variational auto-encoders on the 24 hours of clean training data. Since we had over 2000 unique KPI's we decided to cluster by KPI type, doing so reduced the number of models we had to train and possibly deploy to around 100. After omission of all the zero standard deviation KPI's we further reduced that number.

We first normalised the KPI data, then generated the training set for each model by taking all windows of a fixed size from all host's containing that KPI and concatenating them. After our model converged

we calculated the average reconstruction error for each of the windows and took the maximum as our threshold for that model. A later approach we also attempted was taking the percentile as the threshold, which didn't prove more effective. The difference between the reconstruction and original can be seen below in Figure 6

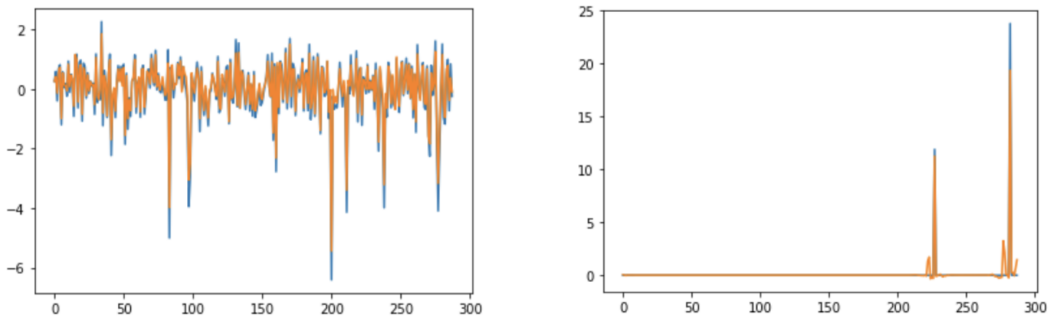


Figure 6: VAE reconstruction (orange) and original (blue)

Although the originally trained models seemed to be effective at reconstruction, decreasing the window size to levels were we could store even infrequent data points lowered the performance significantly. Either the models wouldn't converge on the training data, or the maximum average reconstruction error would be so high that it wouldn't return a single anomaly for the whole testing set.

2.4.3 Spectral Residue Mapping

After running into so many obstacles with the variational-auto-encoders we decided to try something different. In the paper Time-Series Anomaly Detection Service at Microsoft [6] an approach using saliency maps is introduced. As seen in Figure 7 they first generate a saliency map with the spectral residual and finally apply a CNN to determine anomalous behaviour.

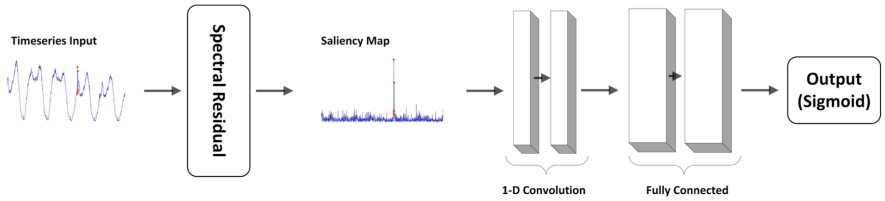


Figure 7: Proposed anomaly detection method

To calculate the saliency map we first extract the amplitude spectrum A and the phase spectrum P given a window of data. Using the amplitude spectrum we can determine the spectral residue through the following formula:

$$R(f) = \log(A(f)) - h_q(f) * \log(A(f)$$

Here $h_q(f)_{ii} = 1/q^2$. Using the spectral residual we can then use the inverse Fourier Transform F^{-1} to return the sequence back into the spatial domain with the following formula:

$$S(x) = ||F^{-1}(\exp(R(f) + iP(f)))||$$

An example of the generated saliency map and original signal can be seen in Figure 8.

Although the method introduced in the paper applies a CNN on the saliency map, doing so wasn't feasible for us. As an alternative we decided to take the simpler approach of using a threshold on the saliency map to detect anomalous behaviour. To determine the thresholds, we first created the saliency map on a window size of 10 for each Host KPI pair. We then stored the thresholds for four different percentiles to see which would prove the most accurate in deployment.

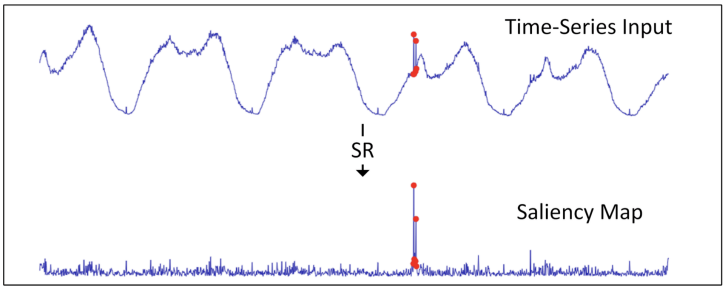


Figure 8: Saliency map compared to the input

Sadly, neither of the thresholds proved effective for discovering the correct KPI. Retrospectively, we believe this could be due to multiple reasons. Firstly Fourier Transforms work on seasonal data, if there is no seasonality however the effectiveness of the approach is questionable. Secondly, many of the KPI's normal behaviour consists of large jumps in value. This leads to saliency maps with likewise large jumps and thresholds that are blind to actual abnormal behaviour. Finally, when trying out this method and even the previous one we did minimal pre-processing, ignoring possible missing values and not smoothing out noise.

2.4.4 ROCKa

Since training individual models for each KPI was unfeasible, we clustered based on KPI type when training our VAE's. In "Robust and Rapid Clustering of KPI's for Large-Scale Anomaly Detection [3]" the authors propose a smarter approach. After standardizing the data, the authors remove extreme values and apply a moving average with a small window on the KPI's. They then used DBSCAN, a density based clustering method, with SBD as the distance metric to determine multiple KPI clusters. Then for different anomaly detection methods, they can be trained on the centroid of each cluster and applied to each KPI model in it.

Although we ended up going for a different approach for our final submission, doing similarity based clustering instead of type based clustering would've definitely been beneficial to try out. Our VAE approach might've seen better results had we done so. Additionally the removal of extreme values and the extraction of a baseline by applying a rolling window average was a great insight. We tried applying different rolling window averages on our spectral residual approach, but it sadly didn't lead to any improvements.

2.4.5 LOUD

Considering the possibility of discovering which KPIs were anomalous, we also considered using LOUD [5] as a method of detecting the propagation of anomalies throughout the system. The base concept of LOUD is that it allows to detect non-linear correlation between KPIs given that anomalies in one system usually affect anomalies in another one.

We generated a propagation graph with all the KPIs. Each KPI was an individual node and the edges have a score corresponding to the similarity. Due to our limited knowledge on this area, we decided to use **Dynamic Time Warping** which is commonly used in signal processing to determine similar signals. Due to it being a very heavy computation process, we applied FastDTW which is an approximation of the real value and is available as a python package. Due to the high number of KPIs finding all the correlations is extremely heavy. Although this approximation was faster, the process was still incredibly slow, and so we used the following approximation to determine the values:

$$DTW(x, y) = DTW(x, r) + DTW(r, j)$$

Where r is a reference point. Since there is a symmetry in the similarity, $DTW(a, b) = DTW(b, a)$, we calculated it with a single reference KPI which sped up the overall process.

This graph was then saved and could be used during inference. The problem after this was that we never managed to find a way to detect an anomaly in each KPI which resulted in this method never being used in the final solution.

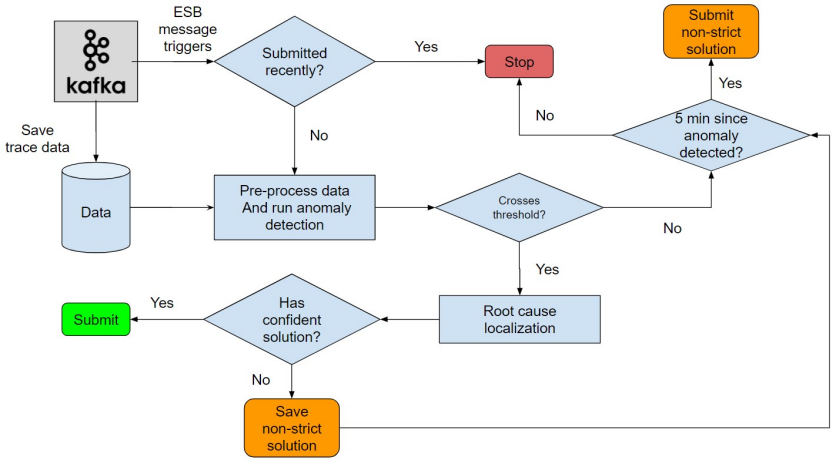


Figure 9: Architecture of final solution

3 Final solution

After attending the presentations of the other teams that are taking part in the challenge, we came to the conclusion that we needed to adapt our approach and do some simplifications. Our previous attempts show that building a robust and general purpose anomaly localization system is excessively time consuming and hard to debug for the current problem. Instead, the findings by the other teams suggested that we need to do more analysis on the data and incorporate domain knowledge to the system logic. By doing analysis on the labeled data the problem can be simplified significantly.

Due to the complexity of the task and the performance requirements, our solution works on a best-effort way: our thresholds for submission are tight and the system only submits when it's very confident that's the true cause. But to avoid not submitting answers and maximize our score, it also saves the results from when it can't decide which submission is correct and submits it after 5 minutes since the detection of the anomaly - it does its best to find the root cause but there is no guarantee that it finds it. We selected the 5 minute as the maximum since in the analyzed system anomalies only extend for the duration of 5 minutes.

3.1 Data Processing

In this section we shall analyze how each data input was used in the final solution. A common point raised by many teams was that the ESB data, even though it is provided in the problem statement, does not carry much information that can be used to reach an answer. While it is true that computing the our previously proposed ESB anomaly detection step can serve as a first filter between normal and abnormal data, this extra step adds some complexity to the program logic and it is not guaranteed to be free of errors. In our case, any missed detection, i.e. false negatives, had no chance to be flagged as anomalies afterwards, which imposed a hard limitation in the overall accuracy. For that reason we are ignoring the ESB data. Similarly, KPI data is also being ignored. Performing anomaly detection in each KPI is extremely difficult as it was explained in subsection 2.4 and we have decided that, due to the past experience and time constraints, that it would be acceptable to not use as it brings more harm than benefit.

Remaining, there is the trace data, which is the data used to solve the problem. The trace information brings both time and structural information about the problem and this can be used to correctly model the problem. The types of anomalies are also limited, so there is no need of determining the anomalous KPI, just understanding which situation is occurring given the patterns observed in the available label data. To extract the network structure from the trace data, our preprocessing consists of the following steps:

1. Spans with call type **JDBC** and **LOCAL** have a *service name* equal to their *database name*;

2. The sum of the *elapsed time* of the children of a span are removed from this span's *elapsed time*, giving us the *true processing time*;
3. **CSF** calls have a *service name* equal to the value of the *host* of its child (there is only one child);
4. **RemoteProcess** calls have a *service name* equal to the value of their host.

This selection of preprocessing techniques allow us to extract important features from the trace information and thus model the problem with it, without requiring any extra analysis of both ESB and KPI data.

3.2 Anomaly detection

Given the preprocessing done in subsection 3.1 the intuition of this approach is the similarity of the service. Given that in the context of the problem there is only one service, **OSB_001**, this service will have a limited number of functions. Therefore, its trace call path will be limited and similar calls shall have similar behaviours. This is similar to the intuition from TraceAnomaly [4] which generates the call paths. Our approach consists of building a trace tree and looking into the *(depth, call type)* pairs.

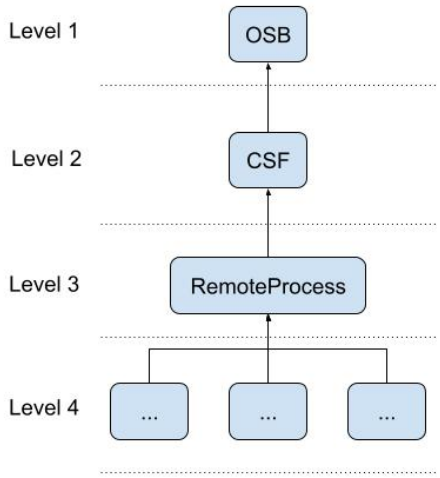


Figure 10: Tree Path example. We differentiate between levels and call types.

We pre-processed all the training data and got each individual trace, then from each trace we built a trace tree, to which we shall name **tree path**. Then for each tree path we saved the elapsed times of each node to be able to extract its quantiles (upper and lower bounds), thus giving us a way to check if a node is behaving anomalously.

Given these transformations, we can now determine if each span is anomalous or not. We count both the number of anomalous spans and the number of total spans, both per *(service, host)* key, and we can now get a percentage value for each individual service/host. The choice of this key pair is so that it can represent the components of the system, which can point us in the direction of the root cause. The percentage value that is obtained is the indicator of the bad behavior of each span. To identify anomalies, we defined a threshold to which the maximum anomalous span surpass for it to be considered an anomaly. We assumed 40% as a good value, as anomalies would cause many traces to be failing or have increased times. This threshold also comes to our aid: because we decided not to use the ESB data as the trigger of an anomaly, we are running this method every minute with a 1-minute window, and we can then decide whether to perform root cause or to stop here. The threshold isn't very low to avoid detecting random failures which might happen due to many reasons but it's not really an anomaly happening. As stated before, we also implement a best-effort logic. We added a second threshold, this time of 70%, which we consider to be a sufficiently high confidence

level. Root cause solutions then have two types: **strict** and **non-strict**. Strict anomalies are those that belong to the $[0.7, 1.0]$ interval and are of the same type (there aren't multiple options). The non-strict anomalies are all the other ones.

3.3 Root Cause Localization

In case the maximum anomalous percentage crosses the threshold, our solution then starts performing root cause localization to discover the source of the anomaly. Given this maximum value, we extract all the other existing percentages within 10% distance from it as the anomaly propagation. This allows us to look into the overall anomalous status of the system, instead of limiting to the maximum value (as this maximum might not be the real root cause due to the propagation of anomalies).

Then, given this collection of high-percentage elements of the system, we used domain knowledge of the system to determine what is the root cause. The elements can be seen as a table service-host and some properties can be extracted from it. We have summarized our approach on Table 2. The possible combinations of anomalies was very reduced, so it was possible to input this logic. Databases have different combinations, which some other cases don't, but due to time limitations we had to do a trade-off between implementation time and accuracy of the results, as the evaluation formula takes the accuracy into consideration. That means that, whenever a database is regarded as a root cause, it will submit all KPIs as anomalous, effectively reducing the complexity but also the accuracy.

Situation	Result
Docker remote calls	Network error in docker
Docker local call	CPU error
Multiple docker in the same VM	VM error
os_021 remote call	os_021 error
os_021 and os_022 remote call	os_001 error
fly_remote	os_009 error
Database	Database error

Table 2: Root cause summary

4 Discussion

The different approaches presented in this document have been considered as a solution to the challenge, but in many cases we encountered issues that imposed hard limitations. Our final solution achieved **55 points** in the daily testing, thus placing us in 6th place across all teams.

rank	group name	score	highest score	round 1	round 2
1	学堂路车神	211	224	985	985
2	meow meow	196	200	624	765
3	Veritaserum	130	142	535	515
4	MSSherlock	72	102	369	325
5	flower group	72	72	161	181
6	Learning Failure	55	55	0	0
7	The Anomalies	45	45	70	66
8	study group	42	44	0	321
9	ANM小组	34	35	153	107
10	ANMG	30	30	78	140
11	DANM!	0	0	0	0

Figure 11: Final leaderboard

Our first solution using all the data following a hierarchical set of steps presented a complicated logic, despite being an intuitive way to decompose the problem. The main finding is that by cascading the detections, errors may accumulate causing the end-to-end detection to lose accuracy, or not find any anomalies. It also became difficult to debug and fine tune the system.

Takeaway 1: It is not worth using deep learning on a simple problem. The implementation of the Variational Autoencoder for ESB detection resulted in an acceptable detection performance, although the improvement with respect to a heuristic method was not significant. Since the VAE used in subsubsection 2.2.1 required deep learning to work, the real-time inference on the server had overhead. The simple 3 sigma rule from subsubsection 2.2.2 had comparable performance while being orders of magnitude faster to compute, which made us decide to abandon the VAE for this first task.

Takeaway 2: Domain knowledge can help turn a complicated problem into a simpler one. We spent a lot of time working on the initial solution architecture and reading and implementing other papers that proposed general solutions to the anomaly detection/localization and root cause problem. The result was that we not only understood that they did not fit very well our problem due to the different data format and information, but also that if we did not feed in any architectural knowledge, we would never find the solution to the problems: some of the anomalies occurred in the parents of a host and we had to infer that knowledge from our extra knowledge instead of from the KPIs themselves. The end result is much different than the original approach: at the beginning we over-complicated the problem and approached it in an incorrect way. The final solution is not only more efficient but also more accurate. The problem is that this last solution is not system independent and requires some hand-tuning.

Takeaway 3: Do not use pandas for live inference. Unfortunately, due to our limited knowledge on the technology, we ended up using pandas for our initial solution. The problem is that this framework is very slow for real-time inference. Consequently, we wasted a lot of time trying to improve its performance instead of working on the problem itself, since we had many errors showing up from the lack of speed required from our program. By switching to simple python objects and structures (dictionaries and lists) we were able to improve the performance from a constant CPU usage of 100%, loss of information and long time data preprocessing to a mere 2% CPU usage while reading the data and almost instantaneously data preprocessing. Both solutions included multi-threading to have a Kafka Consumer constantly reading the data, and another thread to perform the analysis (which would run on every time-interval of 1 minute).

Takeaway 4: Partial data is sufficient to solve the problem. Initially we tried to define a general solution to each type of data to perform an analysis. After watching the presentations from some groups, we noticed that they were being able to perform the detection while ignoring parts of the data. In our case, since we had a very high difficulty trying to detect anomalies in the KPIs, we believe it most likely would have been wise to simply filter out which KPIs require advanced techniques for its detection and use a threshold for the remaining, simplifying greatly and most likely would have been able to improve the performance. We could have also opted for a KPI-driven approach instead of Trace-data driven approach, as some other groups have done. By the end of this project, we also took a look back to what we have done to evaluate why the previous attempts were not working. Only close to the deadline we noticed the existence of an excel file describing the structure of the system. We believe that this lack of knowledge impacted our approach to the problem and that our previous implementations are failing to consider the problem as a whole. Besides what has been described throughout the rest of this report, we think that our modified MicroRCA described in subsubsection 2.3.2 would most likely be able to localize the anomalous hosts if given the extra logic of the system.

More specifically about TraceAnomaly described in subsubsection 2.3.1, we believe our data preprocessing was done incorrectly. The amount of different call paths that exist could have been fixed by taking a similar approach to TreePath (Figure 10) where we take into consideration the depth of the trace to differentiate each node. By making this modification, the amount of call paths that exist in the data would be reduced and would most likely make possible the use of TraceAnomaly, thus fixing the lack of precision of the timestamp.

Acknowledgments

We would like to thank Qianyu Ouyang and Dan Pei for their valuable feedback and support during this semester under the scope of the Advanced Network Management course at Tsinghua University. We would also like to thank our classmates Yixiong Ji and Henry Zheng for their help when discussing ideas about the project that proved valuable for our total understanding of the matter.

References

[1] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, page 271–279, New York, NY, USA, 2003. Association for Computing Machinery.

[2] Z. Li, W. Chen, and D. Pei. Robust and unsupervised kpi anomaly detection based on conditional variational autoencoder. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pages 1–9, 2018.

[3] Zhihan Li, Youjian Zhao, Rong Liu, and Dan Pei. Robust and Rapid Clustering of KPIs for Large-Scale Anomaly Detection. In *2018 IEEE/ACM 26th International Symposium on Quality of Service, IWQoS 2018*, 2019.

[4] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In Marco Vieira, Henrique Madeira, Nuno Antunes, and Zheng Zheng, editors, *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*, pages 48–58. IEEE, 2020.

[5] Leonardo Mariani, Cristina Monni, Mauro Pezzé, Oliviero Riganelli, and Rui Xin. Localizing faults in cloud systems. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 262–273. IEEE, 2018.

[6] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at Microsoft. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[7] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2828–2837, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Budapest, Hungary, April 2020.

[9] Haowen Xu, Yang Feng, Jie Chen, Zhaogang Wang, Honglin Qiao, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, and et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, 2018.

[10] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, page 103–114, New York, NY, USA, 1996. Association for Computing Machinery.