Predicting failures in multi-tier distributed systems: an analysis

Mariana Ávalos Arce
Distributed Computing

Based on the article *Predicting failures in multi-tier distributed systems* by Mariani, Pezze, Riganelli and Xin[1], the following analysis is performed on the common faults and key features for Fault Tolerance in layered distributed systems, with the example of PreMiSE project presented in the article.

1 Common failures on layered distributed systems

A layered distributed system is a system formed by distributed nodes, and each of these is organized in layers. Each layer provides services to the layer above it, while using services from the layer below it as well. Failures are the norm instead of the exception in these systems because of their distributed nature and because it is a common practice to use cheap components as nodes.

Predicting failures in these systems commonly use two techniques: **Anomaly or Signature-based Strategies.** The first predicts failures behaviours that differ from the normalcy of the system, which are considered the symptoms of failure; while the second predicts failures from matching behaviours to known patterns of failure (signatures). A **common failure** arises especially while using the latter technique, because distributed layered systems have performance indicators that are continuous variables important in future faults, but these variables are **not considered** in signature-based approach, as it only considers discrete variables for fault prediction.

Another common failure mentioned is **Network Packet Loss**, which usually happens when an excessive re-transmission of network packets happen at the same time where there is a lack of system response coming from a service of an specific layer. Then, a node's network may encounter packet loss, which leads to a system failure in the future, such as the decrease of client requests of the promised service and decrease CPU idle time. Inside these packet failures, other common failures are caused when the system may present **anomalous rates on packet re-transmission or aborted operations**, which could happen due to an unexpected peak of requests or communication (network) problems.

Other common faults in layered distributed systems are summarized in the following categories: Network Issues, Resource Leak, High Overhead, Host Hanging/crashing, Virtualization Management, High Availability, Guest Hang/crash, Protocol Mismatch, Resource Race, Hardware, Delayed Transitions. The most common out of these are: Network, Resource Leaks and High Overhead faults.

- **Network Faults**: they are a set of networking issues that affect the network and transport layers, such as the mentioned packet loss problem. Common faults of this nature include Packet Loss due to hardware and excessive workload conditions, Packet Latency due to network delay and Packet Corruption due to errors in transmission and reception.
- **Resource Leaks**: they occur when promised resources that should be available are not obtainable, such as faulty process that does not release memory. Common faults of this nature include Memory Leaks.

- ☐ **High Overhead**: these faults occur when a system component cannot reach its objectives because of improper performance, such as poorly developed APIs or activities of high resource demand. High overhead faults due to CPU Hogs.

2 Key features of the PreMiSE project

PreMiSE is a proposed tool to predict failures in layered distributed systems, by tackling both the challenges inside single-layer and multi-layer systems. PreMiSE improves also the signature-based technique, by taking an approach to predict failures that include these continuous performance indicators. Key approaches of PreMiSE include, in a more general sense:

- ☐ Monitoring the status of the system: PreMiSE collects a large set of performance indicators from all the nodes, such as CPU usage for example. In the article, these are called Key Performance Indicators (KPI).
- ☐ Identification of deviations: PreMiSE identifies deviations from the normal behaviours of the system by pointing to anomalous KPIs with anomaly-based approach.
- ☐ Identification of incoming failures: PreMiSE identifies symptomatic and suspicious KPIs with an improved signature-based technique.

Some of the more specific key features of PreMiSE:

- **KPI Monitoring Activity**: PreMiSE collects various KPIs from different resources, that are used to estimate the status of the system. PreMiSE does this by relying on lightweight monitoring infrastructure, available at the different layers. Then, the collected data from these KPIs is elaborated in a separate node of the distributed system that executes data processing routines, relegating the computation costs to a different and independent node. Each monitored KPI is a time series, which is a sequence of numeric values associated with a timestamp and a resource. A KPI is a pair of values or a data structure that contains [resource, metric]. The set of monitored KPIs can be customized.
- **Anomaly Detection Activity**: PreMiSE uses multivariate time series analysis for finding anomalies. PreMiSE has a training phase with the purpose of building a baseline model out of the collected KPIs, and then defines this model as the legal behaviour of the system. Then, when out of the training phase, PreMiSE uses multivariate time series analysis to combine the new KPIs with the baseline model, which will then reveal anomalies. However, these anomalies are not yet separated between malign and benign anomalies. For that, the next feature was added.
- **Failure Prediction Activity**: for this feature, PreMiSE uses signature-based techniques to distinguish good and bad anomalies, depending to which type the incoming failures match to. In the case of bad anomalies, the resources are located. For this, PreMiSE uses historical data about successful and failed behaviours in the past in a learning process to relate these behaviours with future failures.
- **False Alarm Reduction**: when PreMiSE distinguishes faults coming from software and faults coming from exceptional behaviours that do not lead to harm, PreMiSE reduces the amount of false fault predictions and therefore involves an optimization of the recovery techniques.
- **Fault Type Identification**: PreMiSE correlates these predicted anomalies to specific types of faults, in order to simplify the corresponding corrective actions.

- **Responsible Resource Identification**: from the prediction of the KPIs, PreMiSE points out the likely resources that might be responsible of detected faults, and therefore it also simplifies the analysis and investigation for problem recovery.

In a more technical sense, the key characteristics or features involve two main components:

- **Offline Model Training**: PreMiSE builds baseline and signature models that capture the legal system behaviour.

  The baseline model identifies anomalous behaviours and the signature model associates these anomalous behaviours to just exceptions or to symptoms of future failure together with pointing to the likely responsible resources. During this phase, PreMiSE monitors KPIs over time under fault-free execution so that the baseline is built and also **seeds faults** of their corresponding types to build the signature model. In this way, the baseline is used to calculate the expected values as opposed to the current values from the execution. Then, if the calculated and current values of the KPIs differ significantly, an indicator of a future failure rises. Then, the baseline model learner generates models with different solutions to capture certain relationships in the time series involved. It includes trends extracted during the legal behaviour of the system. It actually applies *Granger Causality Tests* to know if a time series of a KPI can predict another variable, which involves regression analysis and statistic information of one variable over another and its correlation.

  The signature model requires training considering fault-free and faulty behaviours. This model is the one that classifies the predictions as novel and unexpected but benign behaviours or the failure prompt behaviours and its classes of failures. An important feature also is that PreMiSE can build the models from **different KPIs** and monitors them as a series of variables in time, and also from different **seeded faults**. But in a more general sense, the signature model is extracted by this feature from a set of anomalies that correspond to faults, and that were identified by the Anomaly Detector. An Anomaly is a tuple of KPIs without timestamp that are detected during execution, that also contains its type and resource. During the training, at least a fault is seeded and the model also collects the consequent anomalies.

  Another nested feature would be the **Probabilistic Classifiers** that generate a probability distribution of the collected faults during training, which will be used for the predicting of that fault by the **Failure Predictor**.

- **Online Failure Prediction**: PreMiSE uses the baseline model to detect anomalies and the signature model to predict faults. Its nested main features are the **Anomaly Detector** and the **Failure Predictor**.

  For the **Anomaly Detector** feature, the baseline model is used, and by now it has a set of time series of KPIs that define the legal behaviour. This detector signals multivariate (many KPIs) and univariate (one KPI) anomalies when the current KPIs differ significantly from the baseline model, as explained before. Univariate are detected as samples out of range, and multivariate are detected when KPIs violate the Granger correlation in a Granger graph.

  As for the **Failure Predictor** feature, it is in charge of reporting failures that match an anomaly in the signature model, and refines the type of failure in time, until it converges. It analyses the set of anomalies in a window during training as well. A feature of PreMiSE comes also here, because it gives both **General and Specific Failure Alerts**, one when an initial prediction is made and the other when a prediction stabilizes and converges to a certain type.

3 Recommendations When Designing a Distributed Architecture

- A distributed system should be ordered in layers when it is expected grow in size, which means that the system is likely to cover an increasing amount of requests. This is mainly because the layers will keep the components of the system organized in the services that each step of the process will need. A general approach for this would be to implement independent programs that connect with each other, similar to the PreMiSE's *Activities* described in the previous section, that is, a modular approach by each of the layers.

- A reliable distributed system should tolerate errors but also predict them. The simplest and most intelligent approach I saw in this article is to monitor the performance of different indicators that each server has. Probably dedicating a layer to gather the data of the machine's node during runtime would be a good beginning for this approach. Then, using a **time series** of this gathered data to detect significant anomalies compared to a previously determined baseline would be a safe way to predict failures in the nodes.

- More specifically, the developers should always focus on tolerating the most common failure type: Network failure. An interesting way I saw of handling simple but common errors of network is to do certain connection operations twice or even between time lapses, so that if there's a network connection problem, maybe giving it some time would clear the error and the system would appear as machine-intelligent as it can be.

- Taking advantage of the time series idea, a good extra feature to detect anomalies would involve calculating the **Covariance** of two indicators that are being monitored, say a and b, in order to get how dependent one is with respect to the other, so that if indicator a is reaching anomalous values, the system should also know that b is also expected to reach anomalies in some degree given by the covariance calculated. In this way, the system could **avoid calculating extra time series of a dependent variable**, for example.

- More generally speaking, a high quality distributed system is one that keeps its nodes **load balanced**, so that the system most common failures are avoided. Therefore, a final recommendation I make would be to dedicate one node from the system to monitor and balance the work that the other nodes are handling, so that the distributed architecture does not reach a bottleneck and wastes the distributed architecture. This node can be in charge of gathering the state of each of the nodes and based on this, manage a multi-thread function that constantly updates the load of all machines and the atomic operations these may need.

References

[1] L. Mariani. "Predicting failures in multi-tier distributed systems". In: *The Journal of Systems and Software* (2019). DOI: https://doi.org/10.1016/j.jss.2019.110464.