Detecting anomalies in microservices with execution trace comparison

This paper's purpose is to use execution tracing to detect anomalies in microservices caused by a group of different faults. The authors attempt to accomplish this task by representing traces as call trees, and using this approach to characterize similar traces. They first define a method to represent traces as call trees in a consistent manner. Two types of anomalies are identified: trace structure and response time. The authors of the paper explain that they will calculate an execution trace's anomaly degree as the tree edit distance between that execution trace and a benchmark trace. They also explain how they calculate tree edit distance. Next, the authors state that they classify a response time anomaly as an execution trace where the coefficient of variation of the execution time is over a certain threshold. In other words, the anomaly occurs where there is significant variation in the time it takes to process requests. The authors validated their system using the Social Network benchmark application in DeathStarBench and were able to detect CPU hog, network congestion, memory leak and service failure faults in that application. Finally, the authors explain shortcomings in their approach, and list a series of related research.

This paper could prove very useful for our research. First of all, it lists several bad smells to detect, namely CPU hog, network congestion, memory leak and service failure faults. It also defines a method which uses execution tracing to detect these smells in a microservice environment. Representing traces using call trees may be a worthwhile topic to explore. Additionally, the paper contains a list of research, some of which use execution tracing to find anomalies in different kinds of systems. This research may also be worthwhile to look into. Overall, this paper presents a number of topics that could prove to be useful.

The paper also presents a number of challenges to the authors' approach. First, the approach requires a benchmark tracing structure to be defined. This is not a trivial task, as the more inaccurate the benchmark is the, the more inaccurate the entire system will be. Next, constructing a call tree representation of a trace is a complex and expensive computation. Because of this, the approach is only able to monitor small scale microservice environments. Finally, the study uses relatively simple systems to test the approach. The authors' method might not be able to scale to a more complex system. These are all challenges to using the call tree representation approach.

https://www-sciencedirect-com.proxy.library.brocku.ca/science/article/pii/S0167739X20330247