System Analysis and Modeling Group
Prof. Dr. Holger Giese
Masterprojekt SoSe 2020

Monitoring Large Dynamic Systems at Runtime

Large, Dynamic Systems

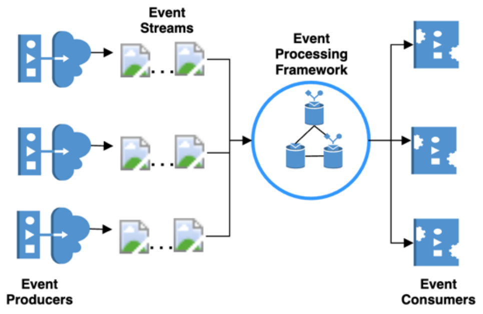
Software has become an integral part of our everyday lives. Societies rely on software for an astoundingly large number of services: from making bank transactions or ordering a taxi to making scientific discoveries. This vast array of capabilities is made possible by systems where legacy as well as current technology is bundled and made available as a service, and subsequently where services interoperate with each other to achieve a common goal. In modern systems, service tend to be self-contained, implement a single capability, and deployed according to demand, a paradigm also known as *microservices* (cf. [1]). The focus of this project is large microservices systems with a dynamic structure, that is, where new or existing microservices can be massively deployed to cater for changing demands.

Monitoring at Runtime

In this project we assume that each service logs its actions in the form of event streams. Scalable deployment of microservices make for a highly dynamic architecture that can potentially generate numerous and voluminous streams. Although there are highly efficient tools and frameworks to aggregate and visualize such streams, the challenge of *efficiently monitoring them to verify at runtime whether they satisfy complex conditions* remains mostly open. By complex conditions, we refer not only to simple operations, e.g. filter or aggregate, but instead to data contained in the events or an event's relationship to other events. An example of such a condition is: "For every query from component A, there is a response (for that same query) from component B within the next 20 seconds.".

Project Outline

The project entails the creation of a pipeline that can monitor large streams of data. The pipeline will consist of components and each component will be implemented by a web-service. A mandatory component will be for monitoring, where we will apply a technique known as Runtime Verification (RV) [2] and use existing RV tools. For the rest of components, *students can choose to focus on one of the following two Focus Points (FP)*:



Typical event processing pipeline [3]

FP1: Generating the stream data using an existing experimental microservices setup
FP2: Using existing real-world data and instead focus on inspecting and processing this data, as well as visualize the monitoring results.

The choice of tools and frameworks, as well as the pipeline structure, is up to the students.

Project Goal and Acquired Skills

The project overall goal is two-fold: a) familiarize with an innovative and powerful testing technique and b) use this technique together with state-of-the-art technologies to tackle the challenge of efficiently monitoring complex conditions for very large streams of data. In the course of the project, the students will a) learn to formulate complex, real-world conditions in a precise (logical) language, b) familiarize with Runtime Verification tools, and c) practice their skills in state-of-the-art technologies in real-world application scenarios.

Prerequisites

There are no explicit prerequisites for the project but an interest in one or more of the following topics might be helpful:

- Testing
- Temporal Logics and Formal Specification
- For FP1: Familiarity with tools like Kubernetes, Docker, ActiveMQ, Spring Boot 2.0, etc.
- For FP2: Familiarity with data stream processing and presentation tools like Jupyter, awk, etc.

Contact

The project is designed for up to 4 students that will be closely supervised by Prof. Dr. Holger Giese and Lucas Sakizloglou (lucas.sakizloglou@hpi.de).

References

[1] Microsoft Azure, *Microservices architecture style*, https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices . Accessed on: 2020-01-06.
[2] Falcone, Ylies, Klaus Havelund, and Giles Reger. "A Tutorial on Runtime Verification."*Engineering Dependable Software Systems* 34 (2013): 141–175.
[3] Steramlio, *Event-driven Architecture using Steamlio*, https://streaml.io/blog/event-driven-architecture. Accessed on: 2020-01-06.