Fault Detection in Microservices using Petri Nets

Sulochan Naik[1], Meenakshi D'Souza[2]

International Institute of Information Technology, Bangalore, India

Abstract

Despite higher adoption of Microservices worldwide, there is a lack of work on possible fault detection during early phases such as at requirements and design phase. We propose a novel way to detect faults using Petri nets' traversal analysis and model checking CTL specifications. Requirements of a system involving microservices are mapped to Netflix's Conductor specification which is later converted into Petri nets for further analysis and detection of faults. We tried our methodology on the TrainTicket benchmark system and were able to detect seven faults.

1. Introduction

Microservices adoption has become a new industry standard owing to its inherent properties. Detection of faults in microservices plays a major role in an effective functioning of service oriented industry. In this paper, we propose a novel way to detect possible faults using traversal analysis of Petri nets derived from microservice requirements. We use Netflix's Conductor orchestration [1] to achieve this. Requirements specification of any microservice application is initially mapped to Conductor workflow specification and later converted to Petri nets. Traversal analysis on this converted Petri nets is performed to derive various properties as CTL Formulas. Using these derived properties, faults for microservices are identified.

2. Methodology and Results

Figure 1 gives complete description of our proposed methodology. Based on our analysis on nature of microservices flow, we have framed the following CTL specifications:

- P1 - Communication behavior can be verified by using appropriate path formulas. For e.g., AF(payment $> 0 \Rightarrow$ receipt) indicates that receipt microservice always follows payment.
- P2 - Failure of a microservice can be identified using : EF(failure state > 0).
- P3 - Decentralized property can be verified by: AF(end is reached) for all microservices.
- P4 - Feasible execution path can be verified by: AF(end is reached).
- P5 - Verification of deadlock state can be done by: EF(end is not reached).
- P6 - Bounded response time can be verified using : AG(some state) $<$ desired token.

Using our methodology, we were able to detect seven faults on a benchmark microservices application called TrainTicket [2] which is depicted in Table 1.

sulochan.naik@iiitb.org (S. Naik); meenakshi@iiitb.ac.in (M. D'Souza)
0000-0001-5959-9570 (S. Naik); 0000-0003-3640-6240 (M. D'Souza)

International Workshop on Petri Nets and Software Engineering 2023 ,PNSE'23

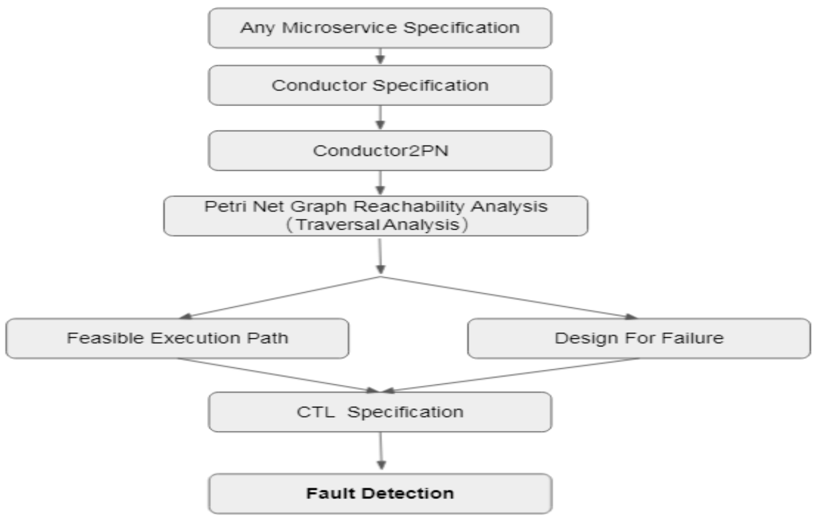

Figure 1: Proposed Methodology

Table 1
Mapping of TrainTicket faults with properties identified through our proposed method

Sl.No	Microservice Errors [3]	Mapping
1	F1 (Messages are displayed in wrong order)	P1
2	F2 (Some information displayed in a report is wrong)	P3
3	F4 (The response time for some requests is very long)	P6
4	F5 (A service sometimes returns timeout exceptions for user requests)	P6
5	F7 (The payment service of the system fails)	P2
6	F16 (The file-uploading process fails)	P2
7	F20 (Nothing is returned upon workflow data request)	P5

3. Conclusion and Future Work

We have verified our methodology on the train ticket benchmark system with satisfactory results. We plan to extend this work as a part of formal verification of microservices.

References

[1] Netflix, Inc., Conductor, 2016. URL: https://netflix.github.io/conductor/.
[2] Microservice System Benchmark, Trainticket, 2018. URL: https://github.com/FudanSELab/train-ticket/.
[3] Microservice System Benchmark, Train ticket fault replicate, 2018. URL: https://github.com/FudanSELab/train-ticket-fault-replicate.