Graph-based Resilience Analysis of Cloud-based Microservices Applications

1st Lorenzo Goglia
Department of Engineering
University of Sannio
Benevento, Italy
logoglia@unisannio.it

2nd Eugenio Zimeo
Department of Engineering
University of Sannio
Benevento, Italy
zimeo@unisannio.it

***Abstract*—The ability of Smart Cities to deliver effective and reliable services depends on the ability of their governing software systems to guarantee a set of quality properties. In this paper, we propose the use of graph-based runtime models and complex networks theory to build a framework for resilience assurance. In particular, the idea is to verify whether graph metrics can be used as surrogate functions for resilience. Results obtained through fault injection testing reveal significant correlation when considering response time as figure of merit.**

***Index Terms*—Models@runtime, Complex Networks, Centrality Metrics, Resilience**

I. Introduction

The growing pervasiveness of ICT makes the performance of modern Smart Cities heavily dependent on that of their supporting software systems. The latter are required to properly deal with the high dynamism characterising their operating environment in a way so as to ensure resilience, *i.e.*, to deliver computational services by minimising and/or adequately tolerating the occurrence of software-related Quality of Service (QoS) violations that may lead to degradation of domain user services. Run-time models, a.k.a. models@runtime, have proven to be a valid solution, allowing for continuous monitoring of the state of a system during its evolution over time and enabling self-adaptation capabilities. In particular, graph-based models and graph theory have been successfully used to *i)* identify the most critical system components from the point of view of the impact on the system working conditions in the event of their failure; *ii)* predict network resilience; and *iii)* prioritise scaling operations [1], [2]. The idea is to model a software system with a property graph that is augmented with outputs generated downstream the execution of graph processing algorithms. In this extended abstract, we report on the analysis of graph metrics to support resilience assurance, specifically whether they can be used as surrogate functions for a given figure of merit. In particular, we evaluated a medium-sized microservice-based application (TrainTicket) by conducting fault injection tests with the aim of verifying the ability of graph metrics to identify potentially critical microservices. The latter are indeed the software units that usually compose the digital ICT layer supporting Smart Cities.

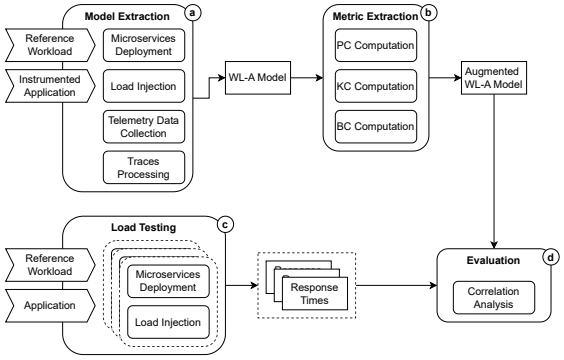
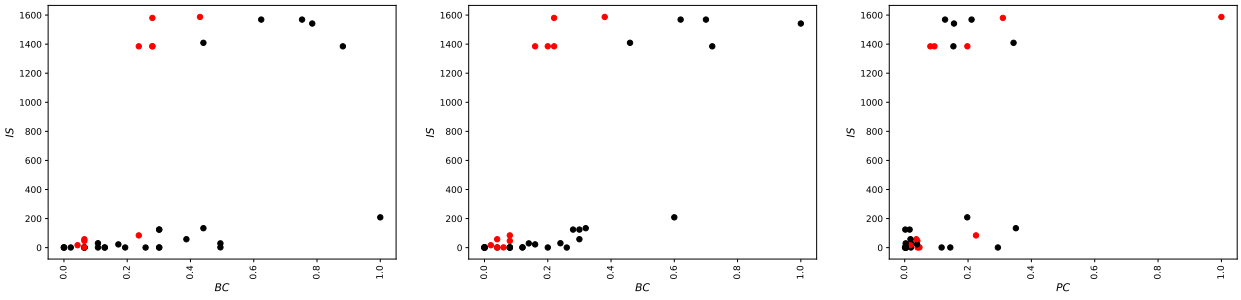


Fig. 1: Framework for evaluating microservice resilience.

II. Evaluating microservice resilience

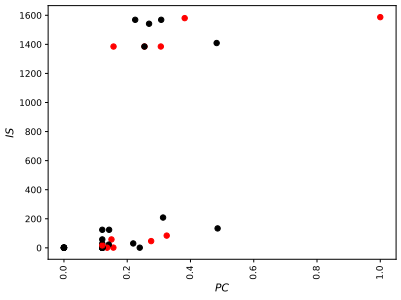
To carry out our study, we designed a conceptual framework for evaluating microservice resilience, whose working steps are reported in Figure 1. In particular, Step *a)* involves the collection and processing of execution traces, which are generated via distributed tracing by submitting a reference workload to the System Under Test (SUT), with the aim of building a graph-based runtime model, called *Workload-Application* (WL-A) model, that can be exploited to study and analyse the SUT. Such a model is a graph whose nodes, edges, and edge weights represent microservices, endpoint invocations, and invocation frequencies, respectively. The second phase *b)* involves the computation of graph metrics, which are used to enrich the starting model. Specifically, we considered Betweenness Centrality (BC), PageRank Centrality (PC), and Katz Centrality (KC). In the third phase *c)*, the SUT is subject to load testing. This operation is repeated $n + 1$ times, where n is the number of microservices, in order to obtain the response times of each microservice endpoint either under normal (*i.e.*, deploying the application with all its microservices) and faulty (*i.e.*, deploying the application by removing one microservice at a time) conditions. Finally, in the fourth phase *d)*, a correlation analysis is performed to check whether microservices with a higher negative impact on application response time also have high values for one or more of the selected metrics.



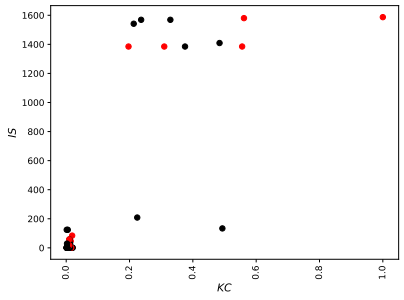
(a) Correlation plot between IS and BC computed on static graph ($dCor^2 = 0.34$ with dbs; $dCor^2 = 0.52$ wout dbs).

(b) Correlation plot between IS and BC computed on dynamic graph ($dCor^2 = 0.45$ with dbs; $dCor^2 = 0.72$ wout dbs).

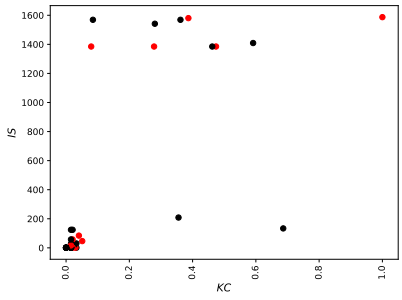
(c) Correlation plot between IS and PC computed on static graph ($dCor^2 = 0.39$ with dbs; $dCor^2 = 0.37$ wout dbs).



(d) Correlation plot between IS and PC computed on dynamic graph ($dCor^2 = 0.40$ with dbs; $dCor^2 = 0.41$ wout dbs).



(e) Correlation plot between IS and KC computed on static graph ($dCor^2 = 0.73$ with dbs; $dCor^2 = 0.66$ wout dbs).



(f) Correlation plot between IS and KC computed on dynamic graph ($dCor^2 = 0.57$ with dbs; $dCor^2 = 0.50$ wout dbs).

Fig. 2: Correlation plots. Red points refer to microservices hosting databases.

III. Correlation analysis

For each configuration, the average response time RT for the reference workload is computed, weighted by the endpoint invocation frequency. Subsequently, for each faulty microservice, an impact score (IS) is computed by dividing the value of RT obtained in the specific faulty setup by that obtained in the nominal setup. IS has a lower bound equal to 1. In fact, when the impact of the removal of a microservice is not significant, the individual response times are similar to those obtained in the nominal condition. Conversely, when the impact is significant, IS increases away from 1. In the worst case, *i.e.*, when all requests in the workload involve a call to any endpoint of the failed microservice, response times are all equal to K times their respective maximums in the nominal setup, where K is a large constant introduced to properly deal with response times collected during fault injection[1]. As for graph metrics, BC, KC and PC, are calculated both on the WL-A model and a model built by considering only structural static information. Specifically, they are computed on a graph built from the analysis of the source code. In this case, nodes, edges, and edge weights represent the microservices, the dependencies between them, and the number of such dependencies, respectively. To verify the existence of correlation, we plotted IS against graph metrics and we calculated the *distance correlation* ($dCor^2$) as performance indicator to determine the degree of dependence between the score and the metrics (see Figure 2). In particular, $dCor^2$ ranges from 0 to 1: $dCor^2 = 0$ indicates independence, while $dCor^2 = 1$ suggests strong linear or non-linear dependence. As it can be observed, BC seems to be the most effective one for analysing application logic components, especially when *i)* the WL-A model is considered and *ii)* microservices hosting databases (those coloured in red) are not considered. This behaviour is motivated by the way the different metrics are defined. BC targets bridging components, *i.e.*, nodes that are pointed by and that points to other nodes, in a finer grain than PC and KC, which instead are more focused on pointed components. In any case, the importance of having models efficiently populated with runtime information emerges. In fact, all metrics perform better (or similarly, as in the case of KC) when computed on dynamic rather than static graphs. These encouraging results confirm the possibility of using graph theory to monitor the behavior of cloud-native and microservices-based applications with the aim of preventing them from entering bad states, thus ensuring continuity in the delivery of Smart City services.

[1]In faulty setups, the response times of removed microservices should be infinite. However, due to timeout mechanisms, they may be comparable to those of working microservices. This depends on the complexity of the underlying operations and means that it is not necessarily true that faulty microservices always exhibit higher response times.

References

[1] Zargarazad, M. and Ashtiani, M. An auto-scaling approach for microservices in cloud computing environments. Journal of Grid Computing, 2023. 10.21203/rs.3.rs-3020374/v1.

[2] Cassé, C. and Berthou, P. and Owezarski, P. and Josset, S. A tracing based model to identify bottlenecks in physically distributed applications. International Conference on Information Networking, 2022. 10.1109/ICOIN53446.2022.9687217.