Frameworks and High-Availability in Microservices: An Industrial Survey

Gastón Márquez[1], Jacopo Soldani[2], Francisco Ponce[1], and Hernán Astudillo[1]

[1] *Toeska Research Group*
Universidad Técnica Federico Santa María,
Avenida España 1680, Valparaíso, Chile
gaston.marquez@sansano.usm.cl, francisco.ponceme@usm.cl,
hernan@inf.utfsm.cl
[2] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
soldani@di.unipi.it

Abstract. While building microservice-based applications, architects need to choose among different frameworks to provide generic functionalities to address quality attribute concerns, such as high-availability. Although using frameworks brings various benefits, it is not clear how they actually impact on the properties characterising the high-availability of microservices. To this end, this article reports the level of agreement among practitioners on the positive and negative impact of frameworks on the high-availability of microservices. More precisely, we first systematically identify 12 properties characterising the high-availability of microservices. We then present the outcomes of an industrial study showing how existing open-source frameworks positively/negatively impact on the 12 properties characterising high-availability of microservices. Results indicate that practitioners agree on the fact that frameworks do positively/negatively impact on high-availability of microservices, and provide a first body of knowledge for suitably selecting frameworks while developing and deploying microservices.

Keywords: Microservices · High-availability · Frameworks · Survey

1 Introduction

Microservices are gaining momentum in enterprise IT, with core businesses of major IT players (e.g., Amazon, Netflix, Spotify and Twitter) already being delivered through microservice-based solutions [23]. This is because microservices (if properly setup) can help ensuring quality attributes of applications, e.g., availability, portability, scalability [19].

A key question is hence how to properly setup microservices to ensure quality attributes, and (as for the scope of this paper) to ensure their high-availability. While developing and deploying microservice-based applications, various decisions must indeed be tackled to make them truly featuring high availability properties. Such decisions include suitably selecting *frameworks* for implementing their components and deployment infrastructure [17, 21]. We hereafter refer to frameworks as collections of reusable software elements providing generic functionalities to address recurring domain and quality attributes [13]. In this same concept, we also include *technological tools* that use sets of frameworks to provide their respective functionalities (such as Kubernetes, MongoDB, among others).

Although there exist frameworks geared towards ensuring high-availability in microservices-based applications [16], to the best of our knowledge, there is currently no body of knowledge recapping the actual impact of frameworks on the high-availability of microservices, from the industrial practitioners' viewpoint. Different frameworks indeed target different properties characterising both high-availability and other quality attributes, and the lack of the above mentioned body of knowledge makes it not clear which of the properties characterising high-availability are positively or negatively affected by a particular framework. This may result in incorrect selections of frameworks, with application administrators that may select frameworks not negatively impacting on one of the properties characterising the high-availability of microservices, which they consider crucial.

A support for suitably selecting frameworks is hence needed, and this paper tries to make a first step in this direction by focusing the following question: *Is a framework positively or negatively impacting on the properties defining high-availability of the microservices forming an application?*

To answer the above question, we first identified the properties characterising the high-availability of microservices with a systematic multivocal review. We then considered the microservice-related frameworks identified in [16], and we systematically elicited their positive and negative impact on the identified high-availability properties by analysing the issues and comments raised within the repositories of 18 open-source microservice-based projects. Finally, to evaluate our findings, we conducted a survey targeting 40 practitioners daily working with microservices, aimed at analyzing their level of agreement on the elicited positive and negative impacts of frameworks on the properties characterising high-availability of microservices.

The main contributions of this paper are hence (i) a list of properties characterising the high-availability of microservices, and (ii) a two-stage exploratory study illustrating positive and negative impacts of existing frameworks on such properties. Our contributions provide a first body of knowledge on the actual impact of framewoks on the properties characterising the high-availability of microservices, and confirm that selecting different frameworks differently impacts on the different properties.

The rest of this article is structured as follow: Section 2 sets the stage of our study by listing the open-source frameworks known to relate to the high-availability of microservices; Section 3 elicits the properties characterising the high-availability of microservices, and Section 4 analyses the impact of frameworks on such properties; Section 5 describes and discusses the results of our study; Section 6 and 7 discuss threats to validity and related work, respectively; Section 8 draws some concluding remarks.

2 Setting the stage

In our previous study [16], we conducted an empirical study aiming at identifying frameworks used to develop microservices-based systems, and their relation with microservice patterns and quality attributes. Figure 1 recaps the process we conducted to obtain frameworks.

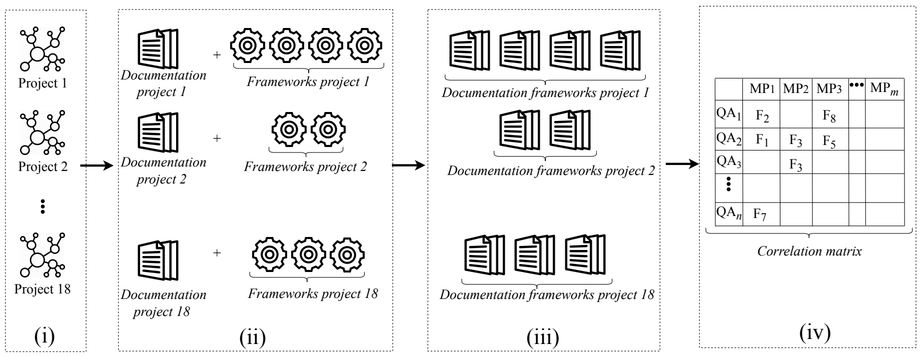

Fig. 1. Process to obtain frameworks. "QA", "MP", and "F" represents quality attributes, microservices patterns, and frameworks, respectively.

We firstly (i) explored 18 open-source microservices projects chosen using the benchmark requirements for microservices research proposed in [1]. We then (ii) analysed documentation and configuration files of each project into elicit the frameworks used in such projects used, and we (iii) analysed the documentation of each framework to identify which microservice patterns are implemented by each framework and which quality attributes are addressed by such framework. Finally, we (iv) created a correlation matrix recapping the results of our analysis by relating framework to quality attributes and microservices patterns.

The outcome of our previous study [16] is a mapping of 23 open-source frameworks to 17 microservice patterns and 6 quality attributes. Such a mapping sets the stage for the study presented in this paper, as it allows to focus on the open-source frameworks known to be related to the high-availability of microservices. Such frameworks are listed in Table 1. We selected these frameworks because, according to their documentation, the main functionalities they address are associated with high availability features.

3 Eliciting high-availability properties

The first activity in our study was to identify the properties characterising the high-availability of microservices (hereafter called *high-availability properties*, for brevity reasons). The activity consisted in conducting a systematic multivocal review, which (due to space limitations) we briefly present hereafter.

Table 1. Open-source frameworks related to the high-availability of microservice.

Framework	**Description**
RabbitMQ	Open-source message-broker application framework
Apache Zookeper	A service for distributed systems offering several functionalities
Docker	A set of platform as a service (PaaS) products
Apache Cassandra	Distributed NoSQL database management system
Netlifx Eureka	Service registry for resilient mid-tier load balancing and failover
Kubernetes	Container-orchestration system
MongoDB	Cross-platform document-oriented MoSQL database application

The search in our systematic review was driven by availability checklist described in [3]. The checklist provides guidelines for designing and analysing the availability of system, organised among the following categories: allocation of responsibilities, coordination model, data model, mapping among architectural elements, resource management, binding time, and choice of technology. For each category, there is a further checklist to be analyzed when availability requirements or properties should be satisfied.

We exploited the checklist for searching for white literature (i.e., peer-reviewed academic research papers) and grey literature (i.e., industiral white paper and blog posts) discussing properties related to the element of the checklist. The search for white literature was done on well-known indexing sources, i.e., Google Scholar (primary), ACM Digital Library, IEEE Xplore and Scopus. The search for grey literature was instead run on Google (with analogous stopping criteria to those in [19]) and on renowned blogs, i.e., InfoQ (`https://www.infoq.com`) and Dzone (`https://dzone.com`).

Finally, we conducted inter-rater reliability assessment and brainstorming sessions to refine the list of identified high-availability properties. To this end, we held (virtual) meetings to identify and describe the properties. Each property was discussed and negotiated until all research team members agreed on the property. As a final result, we elicited 12 high-availability properties, which are listed and described in Table 2.

4 Analysing the impact of frameworks on high-availability

We hereafter illustrate our approach to analyse the impact of frameworks on high-availability properties. We first illustrate how we elicited the impact of frameworks from existing open-source microservice-based projects (Sect. 4.1). We then show how we evaluated the actual perception of industrial practioners of such impact, based on an online survey (Sect. 4.2).

4.1 Eliciting the impact of frameworks on high-availability, from existing projects

We analysed the impact of frameworks on high-availability properties by starting from 18 existing open-source microservice-based project, and by collecting the

Table 2. High-availability properties in microservices-based systems

Id	Name	Description
P1	*High detection of failed host*	This property defines the capability of detect failed hosts in order to a load balancer can stop requests to them.
P2	*Intermittently asynchronous data transmission*	Communication property where a message sender does not wait for a response.
P3	*Regular snapshots*	Property related to recovering the system from planned host maintenance owing to hardware upgrade, soft reboot, etc.
P4	*Efficient duration of timeouts periods*	Property that prevents remote procedure calls from waiting indefinitely for a response.
P5	*High isolation*	The property where each microservices is its own encapsulated application.
P6	*Effective load balancing*	Efficiently distributing incoming network traffic among groups of backend servers.
P7	*Quick broken state recovery*	Capability to restart states when they are broken for a more extended period.
P8	*High control of failure propagation*	Property which indicates the capability of isolate failures through a good definition of service boundaries.
P9	*High service monitoring visibility*	Property that allows visibility into the health of the microservice architecture.
P10	*Periodic heartbeat signal*	Property related to the periodic signal to check the status of services.
P11	*Low application restarting*	This property refers to the low rate of restart services when a failure occurs.
P12	*Efficient resources consumption*	Property related to the impact on the resources consumption (hardware/software) of a system.

(*positive* and *negative*) issues and comments on developers on frameworks and high-availability. We used the following inclusion and exclusion criteria to select the 18 open-source projects we considered:

– *Inclusion criteria*: Benchmark requirements for microservices projects proposed in [1], and projects with source code available.
– *Exclusion criteria*: Projects with no robust information (basic examples, projects in process, others), tools to build microservices (instead of frameworks), component-based projects to build microservices (e.g., projects just for gateway components to microservices), and projects used as an example for lectures or talks.

To collect and organise the issues and comments on high-availability of microservices (hereafter simply called *high-availability issues*), we followed the process shown in Figure 2. For each project, we explored open and closed issues to identify, collect and organise high-availability issues reported by developers and related to the at least one of the microservices frameworks listed in Table 1

From a total of 4,188 candidate high-availability issues, we filtered out all such issues where developers were not describing the rationale for selecting frame-

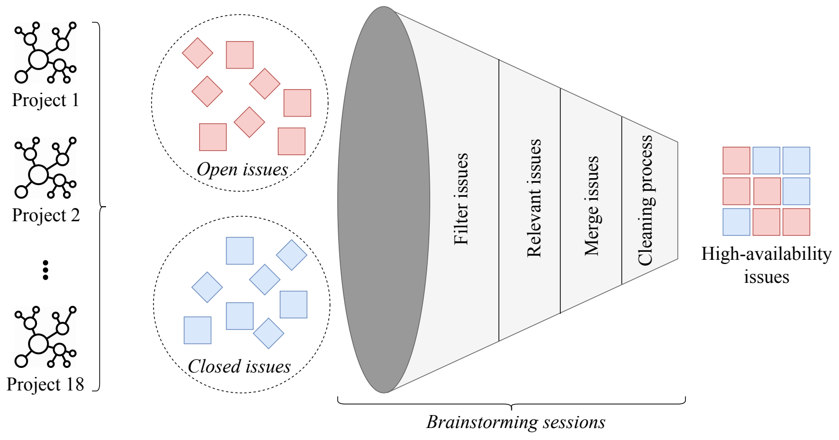

Fig. 2. Process to obtain high-availability issues

works, reducing the scope to 958 ($\approx$ 22.8%) candidate issues. We further refined the scope by restricting to those issues where we found posts explicitly relating one of the frameworks in Table 1 to one of the high-availability properties in Table 2. This allowed us to restrict the scope to 87 high-availability issues, examples of which (concerning RabbitMQ) are reported below:

- Positive issue: "*the Event bus is based on async communication based on RabbitMQ [...] It is used for integration events derived from transactions that happened in any of the microservices and when other microservices need to be aware of those events*".
- Negative issue: "*If RabbitMQ is unreachable when the service tenant starts up, the existing connection is lost for whatever reason, it won't reconnect*"

Afterwards, we proceed by identifying and merging similar/analogous candidate high-availability issues. We clustered issued based on the frameworks they correspond to and we analysed clusters to elicit positive and negative high-availability issues. This allowed us to obtain 8 merged high-availability issues, an example of which is the following: "*RabbitMQ permits implementing asynchronous communication among microservices. But, if a connection is lost, RabbitMQ can affect the number of long-lived connections.*"

From the obtained high-availability issues, we extracted 17 statements concerning the impact of frameworks on high-availability properties. Such statements are listed in Table 3. The table also indicates whether a statement denotes a postive or negative impact over corresponding high-availability properties.

4.2 Evaluating high-availablity issues, based on a survey

To evaluate the actual recognition of the statements in Table 3 among industrial practitioners, we conducted an online survey (following the guidelines described

Table 3. Statements on the positive/negative impact of open-source frameworks on high-availability properties.

Statement	Impact	Prop.
S1: RabbitMQ permits implementing asynchronous communication among microservices	Positive	P2
S2: RabbitMQ can affect the number of long-lived connections if a connection is lost	Negative	P1
S3: Apache Zookeeper supports session timeouts	Positive	P4
S4: Apache Zookeeper can significantly impact on the resources consumption (hardware/software) of a system	Negative	P12
S5: Docker is a good solution for packaging and isolating microservices into containers	Positive	P5
S6: Docker can significantly impact on the resources consumption (hardware/software) of a system.	Negative	P12
S7: Docker can hamper the heartbeat-based monitoring	Negative	P10
S8: Kubernetes supports services monitoring	Positive	P9
S9: Kubernetes can significantly impact on the resources consumption (hardware/software) of a system	Negative	P12
S10: Apache Cassandra supports database service replication	Positive	P6
S11: Apache Cassandra can significantly impact on the resources consumption (hardware/software) of a system	Negative	P12
S12: Netflix Eureka helps monitoring microservices, by allowing to locate them and their logs	Positive	P9
S13: Netflix Eureka can significantly impact on the resources consumption (hardware/software) of a system	Negative	P12
S14: MongoDB supports the isolation of database services	Positive	P5
S15: MongoDB can hamper the heartbeat-based monitoring of database services	Negative	P10
S16: MongoDB automatically routes requests to the appropriate databases	Positive	P2
S17: MongoDB does not support fanning simple transactions out to different database partitions	Negative	P8

in [14, 15]). We hereafter present the survey setting, while its results are discussed in Sect. 5.

Scope and target audience The scope of the survey falls within the IT industry, and in particular that regarding the usage of microservices to deliver core businesses. Within such scope, the focus was on the usage of open-source frameworks to develop and deploy microservices.

The target audience hence entailed industrial practitioners daily working with microservices, either directly part of the authors' relationship networks or reachable through professional networks, frameworks-oriented social networks and technology-related portals. Furthermore, practitioners were also reached by exploiting the snowballing sampling approach [15]. The target audience was given the possibility to participate in the survey in the period lasting from the 1st of March 2019 to the 31st of October 2019.

Goal and research questions The goal of the survey was to evaluate the impact of open-source frameworks on the high-availability of microservices, as per its actual perceiving by industrial practitioners daily working with microservices. In particular, we aimed at evaluatiing their level of agreement with the statements we elicited in Table 3. To this end, we established the following research questions:

- **RQ1**: *Which is the level of agreement of respondents concerning positive contributions related to specific frameworks and high-availability properties?* By answering this research question, we aimed at identifying to which level frameworks are considered to positively impact on high-availability properties.
- **RQ2**: *Which is the level of agreement of respondents concerning negative contributions related to specific frameworks and high-availability properties?* As for RQ1, the goal of this research question was to understand to which level frameworks are considered to negatively impact on high-availability properties.

Survey questions and response format The survey questions were built by directly submitting the statements in Table 3. Respondents were then given the possibility to make their level of agreement with such statements by marking one of the following 3 Likert-scale responses: *Strongly agree*, *Agree*, *Not agree.* We selected this type of scale to obtain more concise answers and, in turn, to facilitate the analysis of the respondents [12].

We used an online questionnaire as survey methodology since we intend to ask precise questions in the context of availability. Furthermore, such a methodology is known to allow to search and explore widespread opinions on a specific topic. For specific domains (including that considered by our study), online questionnaires offer several other advantages, e.g., the effort to handle the questionnaire is reduced (for the participants), it is possible to navigate easily through the questionnaire [18].

Data analysis For each response, we proceed to perform a descriptive analysis based on the frequency of responses in order to better understanding of results. In addition, with the aim of further elaborating the key findings regarding the survey, we performed a series of brainstorming sessions. Obtained results are presented and discussed in the following section.

5 Results and discussion

We hereby illustrate the outcomes of our survey. More precisely, we first show the data about the recognised positive and negative impacts of frameworks (Section 5.1), and we then discuss the outcomes of the study (Section 5.2).

In doing so, we shall establish the agreement on a statement by defining a threshold based on the absolute majority of respondents (as in [7]). The latter means that a candidate statement must obtain over a half of the votes to get agreed, i.e., given n the total amount of respondents, it must get $(n+1)/2$ votes if n is odd, or $(n+2)/2$ votes if n is even. As a total of 40 practitioners participated to our survey, this means that a candidate statement is agreed if at least 21 respondents agree with it.

5.1 Practitioners' recognition on considered impact of framewors

Figures 3 and 4 plot the frequencies of answers on the statements concerning the impact of frameworks on high-availability properties. In both cases, we can

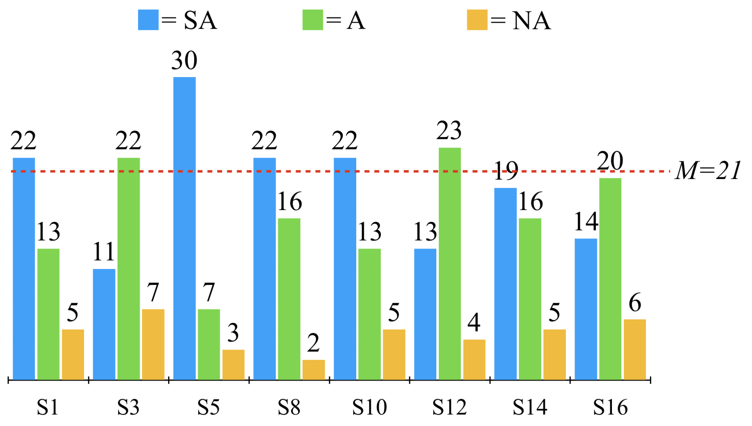

Fig. 3. Survey results regarding positive contribution of frameworks. "SA" correspond to *Strongly agree*, "A" to *Agree*, "NA" to *Not agree*, and "M" to the absolute majority

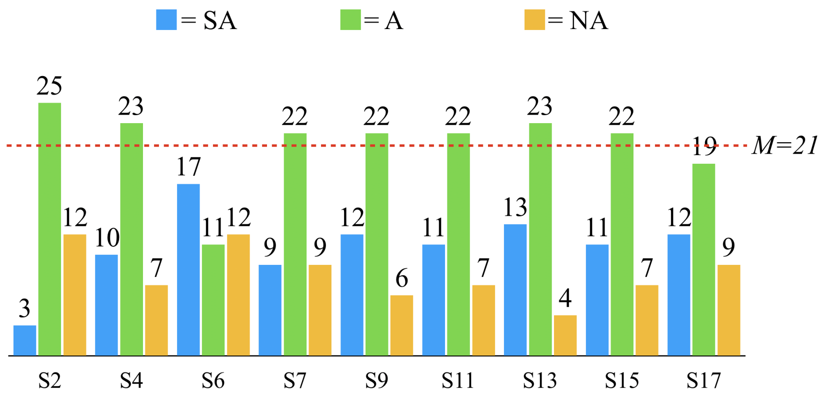

Fig. 4. Survey results regarding negative contribution of frameworks. "SA" correspond to *Strongly agree*, "A" to *Agree*, "NA" to *Not agree*, and "M" to the absolute majority

observe that the vast majority of respondents agrees on the proposed statements (if we considers the sums of *Strongly Agree* and *Agree* answers), hence confirming that our statements on the impact of frameworks on high-availability properties are reflected by the community.

Focusing on Figure 3, we observe that, according to the responses, there is a "strong agreement" on statements S1, S5, S8, S10 and S14, and an "agreement" on statements S3 and S12. Statements S14 and S16 instead do not have a majority of *Strongly Agree* or *Agree* answers, but their sum fairly outgoes the threshold.

Concerning Figure 4, we observe that respondents where lighter in agreeing with statements, as their majority "agree" (not strongly) on all statements but S6 and S17. However, if we consider the sum of *Strongly agree* and *Agree* answers, we observe an overall agreement on all statements.

5.2 Discussion

On the positive compensations side, the survey results indicate that RabbitMQ, Docker, Kubernetes, and Apache Cassandra positively satisfy (*Strongly agree* selection) properties P2, P5, P9, and P6, respectively. Regarding Apache Zookeeper and Netflix Eureka, although respondents indicate that these frameworks are a positive contribution to properties P4 and P9, results indicate that there is no unanimous agreement (*Agree* selection) regarding the positive effect of these frameworks for the P4 and P9.

About negative compensations, the most affected property is P12. This indicates that Apache Zookeeper, Docker, Kubernetes, Apache Cassandra, and Netflix Eureka compromise resource consumption. However, the level of agreement illustrates that the respondents are not categorical with this situation since everyone selected the *Agree* option. This is because, according to what we could investigate, property P12 will be affected depending on the size of the microservices project. For properties P1 and P10, the same situation occurs.

According to the results described in Figures 3 and 4, in the statement S6, S14, S16, and S17 it is not possible to identify the level of agreement. 3 of the 4 statements correspond to MongoDB and 1 to Docker. In order to investigate why there is no agreement in these statements, we conducted a rapid review [4] to establish the final decision regarding which level of agreement should be assigned for these statements.

Regarding statements of MongoDB, Ueda et al. [24] conducted an empirical study with attention to understand the characteristics to design an infrastructure optimized for microservices. The authors mention that, since MongoDB implements a transactional ACID model(Atomicity, Consistency, Isolation, and Durability) on different documents, the way on how the model reference documents to ensure integrity is done by nesting documents, promoting the isolation of the data. Likewise, the same authors describe that one of the significant disadvantages of MongoDB is that it does not support transactions at the moment of updating more than one document or collection. Gadea et al. [8] also mention this observation in the conclusions of his experience of using microservices architecture in a collaborative document editing system. Nevertheless, new MongoDB releases (such as 4.0[3]) provide the ability to perform multi-document transactions against replica sets.

[3] https://docs.mongodb.com/manual/release-notes/4.0/#multi-document-transactions

Other empirical studies, such as [26], mention that MongoDB has the capability to aggregate, route and indexes the unique document ID automatically, improving the performance and scalability in queries against indexed and non-indexed collections, for various (virtual) server hardware configurations [9].

About the Docker resource consumption statement, although some authors (such as [2] and [20]) describe that this situation depends on the project's characteristics, Casalicchio et al. [5] provide an extensive empirical study related to docker performance. One of the conclusions emerged from the authors' experiments is related to the tendency of Docker to use high disk I/O overhead. The authors mentioned that Docker heavily penalizes the execution time when the number of container increase.

Therefore, the studies' results corroborate the answers of sentences S6, S14, S16, and S17. This means that we could take into account the answers with the highest tendency for these sentences (S6→SA, S14→SA, S16→A, and S17→A) in the first instance. Nevertheless, although the studies provide significant research related to the statements' frameworks, there is not sufficient data and evidence to conclude irrefutably the level of agreement of the aforementioned sentences. As future work, we plan to further investigate this finding in order to establish a level of agreement to these sentences.

Finally, we find contributions in 9 of 12 high-availability properties. The properties where we do not find evidence are P3, P7, and P11. The three aforementioned properties are related to specific scenarios of fault recovery. To best of our knowledge, *how to recover microservices-based applications from failures* is one of the research challenges regarding availability in microservices (as also discussed in [19]).

6 Threats to validity

We hereby discuss potential threats to the validity of our study, following the taxonomy in [27]. *Threats to internal validity* describe factors that could affect the results obtained from the study. To mitigate these threats, the participation in the survey was voluntary and anonymous to ensure that answers were as honest as possible. Given that the availability issues are associated with specific technologies, the survey was promoted (i) in communities related to such technologies and (ii) among developers of microservice-based applications. Furthermore, we reinforced the results through interviews with practitioners with microservices-based projects development experience.

Construct validity refers to the extent to which operationalizations of a construct do actually measure what the theory says they do. The threats related to the construct validity were mitigated by careful questionnaire design. Moreover, we piloted the questionnaire internally several times in order to refine the vocabulary and make it as simple possible.

Threats to external validity are conditions that limit our ability to generalize the results. In this context, the survey participants may not represent the entire population of microservices practitioners. To mitigate this risk, we promoted the survey in different communities to include practitioners from different software domains. Furthermore, we decided to interview practitioners (with experience in the development of microservices-based applications) and discuss with them our findings in brainstorming sessions.

7 Related work

Usman et al. [25] presented DRSR, a highly available and fault-tolerant data replication strategy for service registry in a microservice architecture. Similarly, Wu et al. [28] proposed an extensible fault tolerance testing framework for microservice-based applications based on the non-intrusive fault injection.

Tang et al. [22] proposed a distributed service discovery mechanism, which improves the Raft algorithm according to the characteristics of the data, ensures the strong consistency of data between cluster nodes, and improves the availability of service discovery. Heinrich et al. [11] argued why new solutions to performance engineering for microservices are needed. Furthermore, the authors identified open issues with regard to performance aware testing, monitoring, and modeling of microservices.

Despite the significant contribution of previous studies, our study differs from the previous ones in the discussion of positive and negative compensations in high-availability properties related to microservices-based systems. It also differs from other existing industry-driven studies on microservices, such as [19], [21], [6], [10] and [17]. Such studies indeed focus on identifying and discussing advantages and disadvantages of microservices, and/or on distilling best practices. We instead focus on identifying and distilling the positive and negative impact of frameworks on the high-availability of microservices.

8 Conclusions

Our main contributions in this paper are (i) 12 properties characterising the high-availability of microservices (obtained by means of a systematic multivocal review), and (ii) an exploratory study to understand how the usage of existing open-source frameworks positively/negatively impacts on such properties. The exploratory study consisted in a systematically analysing the issues and comments in the repositories of 18 open-source microservice-based projects, which we exploited to extract 17 statements indicating how existing open-source framework impact on the high-availability of microservices. We also conducted an online survey involving practitioners daily working on microservices, whose vast majority agrees with the statements we identified, hence confirming that the impact indicated by such stamements is recognised by the community.

In addition, we provide a first body of knowledge on the actual impact of framewoks on the properties characterising the high-availability of microservices, and confirm that selecting different frameworks differently impacts on the different properties. We plan to work towards extending such a body of knowledge, by including other quality attributes peculiar to microservices, e.g., scalability and technology freedom. We also plan to exploit such a body of knowledge to devise a decision support system, aimed at helping researchers and practitioners finding suitable trade-offs for such quality attributes when designing and developing microservice-based applications.

Acknowledgments. This work was partly supported by the following CONICYT PCHA/Doctorado Nacional through grants 2016-21161005 and 2019-21191132; by CONICYT PIA (Basal FB0821 CCTVal); and by the projects *AMaCA* (POR-FSE, Regione Toscana) and *DECLware* (PRA_2018_66, University of Pisa).

References

1. Aderaldo, M., Mendonça, C., Pahl, C., Pooyan, J.: Benchmark requirements for microservices architecture research. In Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE'17) pp. 8–13 (2017)
2. Anderson, C.: Docker [software engineering]. IEEE Software **32**(3), 102–c3 (2015)
3. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice (3rd Edition). SEI Series in Software Engineering (2013)
4. Cartaxo, B., Pinto, G., Soares, S.: The role of rapid reviews in supporting decision-making in software engineering practice. EASE pp. 24–34 (2018)
5. Casalicchio, E., Perciballi, V.: Measuring docker performance: What a mess!!! Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion pp. 11–16 (2017)
6. Di Francesco, P., Lago, P., Malavolta, I.: Migrating towards microservice architectures: An industrial survey. In: 2018 IEEE Int. Conf. on Software Architecture (ICSA). pp. 29–38 (2018)
7. Dougherty, K.L., Edward, J.: The properties of simple vs. absolute majority rule: cases where absences and abstentions are important. Journal of Theoretical Politics **22**(1), 85–122 (2010)
8. Gadea, C., Trifan, M., Ionescu, D., Cordea, M., Ionescu, B.: A microservices architecture for collaborative document editing enhanced with face recognition. EEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI) pp. 441–446 (2016)
9. Gan, Y., Delimitrou, C.: The architectural implications of cloud microservices. IEEE Computer Architecture Letters **17**(2), 155–158 (2018)
10. Ghofrani, J., Lübke, D.: Challenges of microservices architecture: A survey on the state of the practice. In: Proc. of the 10th Workshop on Services and their Composition (ZEUS 2018). pp. 1–8. CEUR-WS.org (2018)
11. Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L.E., Pahl, C., S., S., Wettinger, J.: Performance engineering for microservices: Research challenges and directions. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion pp. 223–226 (2017)
12. Joshi, A., Kale, S., Chandel, S., Pal, D.K.: Likert scale: Explored and explained. British Journal of Applied Science & Technology **7**(4), 396 (2015)
13. Kazman, R.: Rapid software composition by assessing untrusted components,. SEI (2018), `https://insights.sei.cmu.edu/sei\_blog/2018/11/rapid-software-composition-by-assessing-untrusted-components.html`
14. Kitchenham, B., Pfleeger, S.L.: Principles of survey research part 4: questionnaire evaluation. ACM SIGSOFT Software Engineering Notes **27**(3), 20–23 (2002)

15. Kitchenham, B., Pfleeger, S.L.: Principles of survey research: part 5: populations and samples. ACM SIGSOFT Software Engineering Notes **27**(5), 17–20 (2002)
16. Márquez, G., Astudillo, H.: Actual use of architecture patterns in microservices-based open source projects. 25th Asia-Pacific Software Engineering Conference (APSEC) pp. 31–40 (2018)
17. Neri, D., Soldani, J., Zimmermann, O., Brogi, A.: Design principles, architectural smells and refactorings for microservices: a multivocal review. SICS Software-Intensive Cyber-Physical Systems (2019), *[In press]*
18. Punter, T., Ciolkowski, M., Freimut, B., John, I.: Conducting on-line surveys in software engineering. International Symposium on Empirical Software Engineering pp. 80–88 (2003)
19. Soldani, J., Tamburri, D.A., Heuvel, W.J.V.D.: The pains and gains of microservices: A systematic grey literature review. Journal of Systems and Software **146**, 215 – 232 (2018)
20. Stubbs, J., Moreira, W., Dooley, R.: Distributed systems of microservices using docker and serfnode. 7th International Workshop on Science Gateways pp. 34–39 (2015)
21. Taibi, D., Lenarduzzi, V.: On the definition of microservice bad smells. IEEE Software **35**(3), 56–62 (2018)
22. Tang, W., Wang, L., Xue, G.: Design of high availability service discovery for microservices architecture. In: Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences. pp. 253–257. ICMSS 2019, ACM, New York, NY, USA (2019)
23. Thönes, J.: Microservices. IEEE Software **32**(1), 113–116 (2015)
24. Ueda, T., Nakaike, T., Ohara, M.: Workload characterization for microservices. IEEE international symposium on workload characterization (IISWC) pp. 1–10 (2016)
25. Usman, A., Zhang, P., Theel, O.: A highly available replicated service registry for service discovery in a highly dynamic deployment infrastructure. In: 2018 IEEE International Conference on Services Computing (SCC). pp. 265–268 (July 2018)
26. Viennot, N., Lecuyer, M., Bell, J., Geambasu, R., Nieh, J.: Synapse: a microservices architecture for heterogeneous-database web applications. Proceedings of the Tenth European Conference on Computer Systems p. 21 (2015)
27. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science and Business Media (2012)
28. Wu, N., Zuo, D., Zhang, Z.: An extensible fault tolerance testing framework for microservice-based cloud applications. In: Proceedings of the 4th International Conference on Communication and Information Processing. pp. 38–42. ICCIP '18, ACM, New York, NY, USA (2018)