Benchmarking Microservice Systems for Software Engineering Research

Xiang Zhou[1,2], Xin Peng[1,2], Tao Xie[3], Jun Sun[4], Chenjie Xu[1,2], Chao Ji[1,2], Wenyun Zhao[1,2]
[1] School of Computer Science, Fudan University, China
[2] Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China
[3] University of Illinois at Urbana-Champaign, USA
[4] Singapore University of Technology and Design, Singapore

ABSTRACT

Despite the prevalence and importance of microservices in industry, there exists limited research on microservices, partly due to lacking a benchmark system that reflects the characteristics of industrial microservice systems. To fill this gap, we conduct a review of literature and open source systems to identify the gap between existing benchmark systems and industrial microservice systems. Based on the results of the gap analysis, we then develop and release a medium-size benchmark system of microservice architecture.

CCS CONCEPTS

• **Software and its engineering → Cloud computing**; **Software testing and debugging**;

KEYWORDS

Microservice, benchmark, tracing, visualization, debugging, failure diagnosis

ACM Reference Format:
Xiang Zhou[1,2], Xin Peng[1,2], Tao Xie[3], Jun Sun[4], Chenjie Xu[1,2], Chao Ji[1,2], Wenyun Zhao[1,2]. 2018. Benchmarking Microservice Systems for Software Engineering Research. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194991

1 INTRODUCTION

Despite the prevalence and importance of microservices in industry, there exists limited research, with only a few papers on microservices in the software engineering research community, and even fewer in major conferences, partly due to lacking a benchmark system that reflects the characteristics of industrial microservice systems. Given that most microservice systems developed so far are proprietary and not easily accessible to the research community [2], the existing research on microservices is usually based on small systems with a few microservices, most of which are not publicly available. A recent study by Francesco *et al.* [6] examined benchmark microservice systems used, discussed, or proposed in the literature. They found only one such system that is open source and publicly available, and it has only 5 microservices, far fewer than those in industrial systems.

To fill this gap, we construct a benchmark microservice system that can facilitate research on microservice development and operation. To construct such a system, we conduct a review of literature and open source systems to learn the gap between existing benchmark systems and industrial systems. Based on the survey results, we develop a benchmark microservice system called TrainTicket [3], which includes 24 microservices.

2 REVIEW OF LITERATURE AND OPEN SOURCE SYSTEMS

To review the research literature related to microservices, we systematically search through four of the largest and most comprehensive digital libraries and scientific databases for computer science: the ACM Digital Library, IEEE Xplore, Web of Science, and Scopus. We apply the search string "microservi* *OR* micro-servi* *OR* micro servi*" in the title, abstract, and keywords of the papers. For each paper in the search results, we manually examine the microservice systems described in the paper and check whether they are available online. We find only five microservice systems described in the papers available online, which are all open source systems on GitHub as the first five rows shown in Table 1.

To review open source systems related to microservices, we also search for candidate benchmarks on open source platforms (including GitHub and BitBucket). We search with two keywords "micro" and "service" and manually examine the top 100 returned projects on each platform. For each project, we manually examine the source code to confirm the characteristics (e.g., languages, the number of microservices, interaction modes) of the corresponding system. We find that many projects being claimed as microservice projects actually employ a monolithic architecture and just use some microservice-related techniques such as Docker-based deployment. As our targets are microservice systems, we exclude microservice infrastructure projects (e.g., development tools, operation frameworks) and microservice systems with fewer than 5 microservices. Note that when counting the number of microservices, we omit infrastructure microservices (e.g., service registry, service discovery, load balancing) and consider only the microservices that implement some business functionalities.

Table 1 shows the resulting microservice systems. The table shows that all these systems are quite small, with no more than 9 microservices (the number of microservices is shown in the "#S" column). For example, Acme Air, a widely used benchmark system, has only 5 microservices. Each of these systems implements one or two interaction modes (as shown in the "Interaction Mode" column),

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194991

Table 1: Benchmark Systems Available Online

Name	#S	Interaction Mode
Acme Air [1]	5	Sync
Music Store [7]	6	Sync, Async
Spring Cloud Demo Apps [9]	6	Sync
Bifrost Microservices Sample Application [4]	5	Sync, Async
Socks Shop [8]	8	Sync, Queue
Staffjoy [10]	9	Sync, Queue
NServiceBus [5]	8	Queue

including synchronous invocations (Sync), asynchronous REST invocations (Async), and message queues (Queue).

Based on the result, we identify the following gaps between the existing benchmark systems and industrial systems.

Incomplete Interaction Modes. The existing benchmark systems do not make good use of different interaction modes, while many microservice problems are related to asynchronous communication or a hybrid of synchronous and asynchronous communications.

Insufficient Scale and Complexity. The existing benchmark systems have only a small number of microservices and very short invocation chains, while many microservice problems manifest only through complex interactions.

Insufficient Application of Microservice Design Principles. The existing benchmark systems do not fully follow the principles of the microservice architecture. For example, the microservices should be modularized and organized around business capability; a single microservice should be small enough to be developed, deployed, and operated by a single team. These principles are important for the evolution along with runtime scaling and adaptation.

Insufficient Testing. The existing benchmark systems do not provide sufficient unit test cases or integration test cases. These test cases are important to ensure the quality of the systems and at the same time provide the basis of research on testing and debugging.

3 BENCHMARK SYSTEM

To facilitate research on microservices, we develop a benchmark system for railway ticketing called TrainTicket. A user can use TrainTicket to inquire about the train tickets from city A to city B on a certain day. By selecting the passenger and the class of the seat, the user can reserve a ticket that the user needs. Once the booking is successful, the user is required to pay as soon as possible. After the successful payment, the user gets an email notifying that the user has successfully reserved the ticket. The user is allowed to change the ticket before the departure time or within a certain time period after the train departs.

TrainTicket contains 24 microservices related to business logic (excluding all infrastructure microservices), more than any existing benchmark. Furthermore, many microservices in TrainTicket are interacting with each other, resembling microservice systems in industrial practices. According to the dependencies among microservices, we divide these microservices into five layers, as shown in Figure 1. If microservice A sends a request to microservice B, we say that A depends on B. The bottom layer contains those microservices that do not depend on any other microservice. The upper-layer microservices depend on the lower-layer microservices. Microservices at the same layer may depend on each other. For simplicity, Figure 1 does not show the databases in TrainTicket.

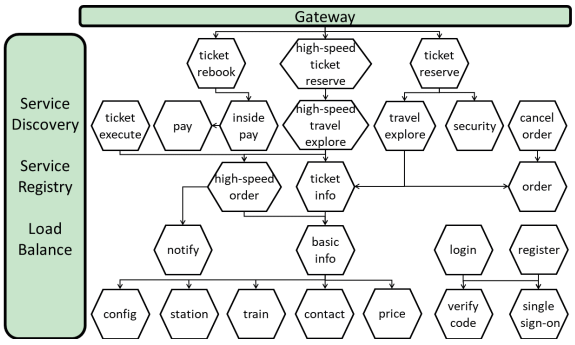

Figure 1: The Architecture of the TrainTicket System

TrainTicket is designed using microservice design principles. For instance, there are two sets of microservices designed for reserving/ordering/exploring high-speed trains and normal-speed trains. The reason for the design is that we can then choose a more flexible deployment strategy, depending on the numbers of customers for high-speed trains and normal-speed trains, respectively. In other words, we can deploy more Docker instances for high-speed trains for better performance if many users are reserving high-speed trains and few are reserving normal-speed trains. Otherwise, the users for reserving normal-speed trains might be negatively affected. Such a design reflects the flexibility of microservices.

TrainTicket is designed to cover many features of microservices. For instance, as shown in Table 1, the existing benchmark systems use only one or two interaction modes, whereas TrainTicket makes use all of synchronous invocations, asynchronous invocations, and message queues.

We use Spring Boot to develop TrainTicket. The system is implemented in Java and Node.js, and the current implementation has 61,136 lines of code. In total, we develop 37 unit test cases and 14 integration test cases with 5,218 lines of test code, and all of them are publicly available in our open source project's repository [3].

ACKNOWLEDGMENTS

This work is supported by National High Technology Development 863 Program of China under Grant No. 2015AA01A203, and NSF under grants no. CCF-1409423, CNS-1513939, CNS-1564274.

REFERENCES

[1] Acme.Com. 2015. Acme Air. (2015). Retrieved August 16, 2017 from https://github.com/acmeair/acmeair
[2] Carlos M. Aderaldo, Nabor C. Mendonça, Claus Pahl, and Pooyan Jamshidi. 2017. Benchmark Requirements for Microservices Architecture Research. In *ECASE@ICSE 2017*. 8–13.
[3] MicroService System Benchmark. 2017. (2017). https://github.com/microcosmx/train_ticket
[4] Bifrost.Com. 2017. bifrost. (2017). Retrieved August 16, 2017 from https://github.com/sealuzh/bifrost-microservices-sample-application
[5] EShopOnContainers.Com. 2017. eShopOnContainers. (2017). Retrieved August 16, 2017 from https://github.com/dotnet-architecture/eShopOnContainers
[6] P. D. Francesco, I. Malavolta, and P. Lago. 2017. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In *ICSA 2017*. 21–30.
[7] MusicStore.Com. 2017. Music Store. (2017). Retrieved August 16, 2017 from https://github.com/aspnet/MusicStore
[8] SocksShopGit.Com. 2017. Socks Shop Git. (2017). Retrieved August 16, 2017 from https://github.com/microservices-demo/microservices-demo
[9] SpringDemo.Com. 2017. Spring Cloud Demo Apps. (2017). Retrieved August 16, 2017 from https://github.com/kbastani/spring-cloud-microservice-example
[10] Staffjoy.Com. 2017. Staffjoy. (2017). Retrieved August 16, 2017 from https://github.com/Staffjoy/v2