July 24, 2024

1 A Quantitative Approach for the Assessment of Microservice Architecture Deployment Alternatives by Automated Performance Testing

Publication Details:

- **Authors:** Alberto Avritzer, Vincenzo Ferme, Andrea Janes, Barbara Russo, Henning Schulz, and André van Hoorn
- **Title:** A Quantitative Approach for the Assessment of Microservice Architecture Deployment Alternatives by Automated Performance Testing
- **Publication Year:** 2018
- **Venue:** 12th European Conference on Software Architecture (ECSA 2018)

Research Focus:

The paper focuses on assessing the performance of various microservice architecture deployment alternatives using a quantitative approach. It introduces a method for using operational profiles to generate load tests that automatically assess the scalability of different deployment configurations.

Methodology:

The approach involves four main steps: analysis of operational data, generation of experiments, computation of baseline requirements, and execution of experiments. The performance is evaluated using a domain-based metric that quantifies the ability of a configuration to satisfy scalability requirements under a given operational profile.

Tools and Frameworks:

The study uses BenchFlow, an open-source framework that integrates Docker, Faban, and Apache Spark to automate performance testing, collect performance data, and compute metrics.

Findings:

The study found that increasing the number of Docker containers or CPU allocation does not necessarily improve performance and can sometimes degrade it. The results emphasize the importance of carefully evaluating deployment alternatives and operational profiles.

Metrics:

The key performance metric used is a domain-based metric that ranges from 0 to 1, reflecting the ability of a configuration to meet performance requirements. This metric considers factors such as the response time and the scalability of the system under different workload situations.

Categorization:

- **Impact on Quality and Reliability:** This paper evaluates different deployment alternatives for microservice architectures using automated performance testing. It measures the impact of these alternatives on system quality and reliability.

2 A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development

Publication Details:

- **Authors:** Mazedur Rahman, Jerry Gao
- **Title:** A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development
- **Publication Year:** 2015
- **Venue:** IEEE Symposium on Service-Oriented System Engineering (SOSE 2015)

Research Focus:

The paper addresses the need for reusability, auditability, and maintainability in Behavior-Driven Development (BDD) frameworks applied to microservices. It introduces a reusable automated acceptance testing architecture to improve the management of BDD scenarios across multiple microservices.

Methodology:

The proposed architecture involves the creation of a dedicated repository for automated acceptance tests (AAT), which includes directories for features, step implementations, and the BDD framework library. This structure allows for the reuse of common BDD steps across multiple microservices, simplifying refactoring and maintenance.

Tools and Frameworks:

The paper mentions the use of BDD frameworks such as Cucumber, Behat, and Behave for implementing automated acceptance tests. However, no specific tools are used to generate these tests.

Findings:

The architecture significantly reduces the maintenance burden associated with BDD acceptance tests for microservices. By centralizing the BDD steps, the proposed architecture enhances reusability, eases auditability, and improves collaboration among developers, testers, and business analysts.

Metrics:

None.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper presents a reusable architecture for automated acceptance testing in microservices, focusing on the Behavior-Driven Development (BDD) process. It discusses the tools and techniques used to achieve reusability, auditability, and maintainability in the testing framework.

3 An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study

Publication Details:

- **Authors:** Stefan Fischer, Pirmin Urbanke, Rudolf Ramler, Monika Steidl, Michael Felderer
- **Title:** An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study
- **Publication Year:** 2024
- **Venue:** 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024), Lisbon, Portugal

Research Focus:

The paper performs a systematic literature search to identify microservice-based systems used for testing and monitoring. It provides a comprehensive overview of these systems, their characteristics, and the contexts in which they have been used to aid future research in evaluating testing and monitoring approaches.

Methodology:

The study employs a systematic mapping approach, conducting a literature search across multiple databases (Scopus, IEEE, ACM, Springer) and applying inclusion and exclusion criteria to filter relevant studies. The selected studies are then analyzed to extract information about the systems used, their characteristics, and the contexts of their usage.

Tools and Frameworks:

None.

Findings:

The study identifies 134 systems used in testing and monitoring research, with 29 of them being open-source and microservice-based. It provides a detailed list of these systems, including their characteristics such as size, technology stack, and usage in research. The study highlights the most commonly used systems and discusses their adherence to the FAIR Guiding Principles (Findable, Accessible, Interoperable, Reusable).

Metrics:

None.

Categorization:

- **Systematic Studies:** This paper is a systematic mapping study that provides an overview of various microservice-based systems used for testing and monitoring. It synthesizes existing research and categorizes the different evaluation methods employed.

4 An Architecture to Automate Performance Tests on Microservices

Publication Details:

- **Authors:** André de Camargo, Ronaldo dos Santos Mello, Ivan Salvadori, Frank Siqueira
- **Title:** An Architecture to Automate Performance Tests on Microservices
- **Publication Year:** 2016
- **Venue:** 18th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2016), Singapore

Research Focus:

The paper presents an architecture designed to automate performance tests in microservices environments. It aims to address the challenges of performance testing by proposing a method that allows each microservice to provide a test specification used for executing performance tests.

Methodology:

The proposed architecture consists of two main components: a specification that allows external applications to access the test parameters for microservices and a mechanism to attach and expose these specifications. The architecture leverages the HTTP OPTIONS method to provide the test specifications in a non-intrusive manner, allowing automated tools to retrieve and use them for performance testing.

Tools and Frameworks:

The study introduces a framework called the Framework for Performance Test Specification (FPTS) implemented in Java using Spring Boot. The framework operates as a filter for HTTP requests to microservices, generating test specifications for OPTIONS requests and forwarding other requests to the service.

Findings:

The framework facilitates the automation of performance tests without impacting the service's performance. The evaluation demonstrated that the framework could be integrated into microservices with minimal overhead and did not affect the response time or throughput of the services. The framework's key features include API capabilities for dynamic data, Java annotations for specifying test parameters, and validation data for ensuring accurate test results.

Metrics:

The evaluation focused on response time and throughput as the primary performance metrics. The results indicated no significant impact on these metrics when the framework was used.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper presents a framework for automating performance tests in microservices, highlighting its implementation and key features.

5 An Automatic Test Data Generation Method for Microservice Application

Publication Details:

- **Authors:** Huixia Ding, Lei Cheng, Qinyuan Li
- **Title:** An Automatic Test Data Generation Method for Microservice Application
- **Publication Year:** 2020
- **Venue:** 2020 International Conference on Computer Engineering and Application (ICCEA), Beijing, China

Research Focus:

The paper focuses on developing an automatic test data generation method for microservice applications. It aims to address the challenges of traditional manual testing approaches by proposing an automated solution that extends the WADL (Web Application Description Language) file and uses an improved pairwise algorithm for efficient test data generation.

Methodology:

The methodology includes extending the WADL file to enhance the expression ability of microservice interface parameters. The test data generation process is divided into four parts: WADL file extension and parsing, single parameter test data generation, test data validity judgment, and combined parameter test data generation using an improved AETG (Automatic Efficient Test Generator) algorithm.

Tools and Frameworks:

None.

Findings:

The proposed method improves test efficiency and management by automating the test data generation process. The improved AETG algorithm (AETG_I) reduces the count of test data by 22% to 39% compared to the original AETG algorithm, while maintaining a high error coverage rate of 95% to 100%. This demonstrates the method's effectiveness in generating efficient test data for microservice applications.

Metrics:

The primary metrics used for evaluation are the error coverage rate and the count of generated test data. The error coverage rate indicates the proportion of detected errors out of the total embedded errors, while the count of test data measures the efficiency of the test data generation process.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper presents a novel method for automatically generating test data for microservice applications, emphasizing the use of an improved pairwise algorithm for efficient and effective testing.

6 Analysis for Microservice Architecture Application Quality Model and Testing Method

Publication Details:

- **Authors:** Jiayu Gong, Lizhi Cai
- **Title:** Analysis for Microservice Architecture Application Quality Model and Testing Method
- **Publication Year:** 2023
- **Venue:** 26th ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter 2023), Taiyuan, China

Research Focus:

The paper analyzes the challenges in measuring and testing the quality of microservice architecture applications. It refines the existing ISO software product quality model to better suit microservice applications and introduces testing methods tailored to these architectures.

Methodology:

The study enhances the ISO/IEC 25010-2011 quality model by adapting it to the specific needs of microservice architectures. It also proposes a detailed methodology for testing microservice applications, which includes unit testing, integration testing, component testing, contract testing, and end-to-end testing. Each testing level is designed to address the unique challenges posed by the distributed nature of microservices.

Tools and Frameworks:

The paper mentions the use of tools like PACT for contract testing and automation tools such as Selenium and Robot Framework for end-to-end testing.

Findings:

The paper identifies key quality attributes specific to microservice architectures, such as coupling, cohesion, and traceability, and adapts existing metrics to measure these attributes. It highlights the importance of automated testing and continuous integration in maintaining the quality of microservice applications.

Metrics:

The metrics used include:

- Coupling (measured by Message Passing Coupling - MPC)
- Cohesion (measured by Lack of Cohesion Of Services - LCOS)
- Traceability (measured by the implementation of service link traceability features)
- Scalability (measured by the benefit of capacity growth relative to the cost)
- Availability (measured by the number of instances of key services and percentage of time in the available state)
- Fault tolerance (measured by the mechanisms of service fusion, flow limitation, degradation, and load balancing)

Categorization:

- **Impact on Quality and Reliability:** The paper refines quality models and introduces comprehensive testing methods to enhance the quality and reliability of microservice architecture applications.

7 Assessing Black-box Test Case Generation Techniques for Microservices

Publication Details:

- **Authors:** Luca Giamattei, Antonio Guerriero, Roberto Pietrantuono, Stefano Russo
- **Title:** Assessing Black-box Test Case Generation Techniques for Microservices
- **Publication Year:** 2022
- **Venue:** QUATIC 2022, Conference on the Quality of Information and Communications Technology, Lisbon, Portugal

Research Focus:

The paper proposes the uTest stateless pairwise combinatorial technique for test case generation for functional and robustness testing of microservices. It also compares the performance of uTest with four state-of-the-art black-box testing tools for RESTful web services (EvoMaster, RestTestGen, RESTler, and bBOXRT) using three open-source microservice architectures (Train Ticket, SockShop, FTGO).

Methodology:

The methodology includes developing the uTest tool, which uses a pairwise combinatorial strategy to generate test cases. The study involves parsing OpenAPI specifications to extract input space models, creating test case specifications based on these models, and generating test cases that are executed and analyzed. The experiments compare coverage metrics, average failure rates, and test generation/execution costs across different tools and scenarios.

Tools and Frameworks:

The study introduces the uTest tool, which implements a pairwise combinatorial testing strategy for microservices. The other tools compared include EvoMaster, RestTestGen, RESTler, and bBOXRT.

Findings:

The uTest tool achieves better average failure rates with fewer test cases compared to other tools. While coverage metrics were comparable across tools, uTest was more cost-effective. The study highlights that tools designed for generic RESTful services may not perform optimally for microservices, underscoring the need for MSA-specific testing techniques.

Metrics:

The coverage metrics used include:

- Path coverage
- Operation coverage
- Parameter coverage
- Parameter value coverage
- Request content-type coverage
- Status code class coverage
- Status code coverage
- Response content-type coverage

Additional metrics for evaluation include the average number of executed tests and the average failure rate.

Categorization:

- **Impact on Quality and Reliability:** The paper assesses the effectiveness of various black-box test case generation techniques for microservices, discussing their impact on the overall quality and reliability of the systems.

8 Automated Grey-Box Testing of Microservice Architectures

Publication Details:

- **Authors:** Luca Giamattei, Antonio Guerriero, Roberto Pietrantuono, Stefano Russo
- **Title:** Automated Grey-Box Testing of Microservice Architectures
- **Publication Year:** 2022
- **Venue:** 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), Lisbon, Portugal

Research Focus:

The paper introduces MACROHIVE, a grey-box testing strategy designed to automate the testing and monitoring of interactions among internal microservices. The focus is on addressing the limitations of black-box testing by providing internal coverage and detailed failure information.

Methodology:

The methodology involves using combinatorial testing to generate both valid and invalid tests from microservices specifications. The tests are executed and monitored via a service mesh infrastructure, which allows for tracing interactions among microservices. The results are then analyzed to provide insights into internal coverage and failure behavior.

Tools and Frameworks:

The study introduces MACROHIVE, which is deployed as a collection of microservices within a service mesh. The infrastructure includes three main components: uTest (for test generation and execution), uSauron (for information collection), and uProxy (for acting as a reverse proxy and logging requests).

Findings:

MACROHIVE performs comparably to state-of-the-art black-box testing tools in terms of edge-level coverage but provides additional benefits by exposing internal failures undetected by black-box testing. It offers detailed internal coverage information and requires fewer tests. The study highlights the advantages of a grey-box strategy in identifying and understanding internal dependencies, errors, and exceptions.

Metrics:

The metrics used in the study include:

- Status code class coverage
- Status code coverage
- Average failure rate
- Internal microservice coverage
- Dependency coverage
- Propagated and masked failures

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper describes a grey-box testing approach for microservice architectures, highlighting the tools and techniques used for automating the testing process and improving test coverage.

9 Automatic Ex-Vivo Regression Testing of Microservices

Publication Details:

- **Authors:** Luca Gazzola, Maayan Goldstein, Leonardo Mariani, Itai Segall, Luca Ussi
- **Title:** Automatic Ex-Vivo Regression Testing of Microservices
- **Publication Year:** 2020
- **Venue:** IEEE/ACM 1st International Conference on Automation of Software Test (AST)

Research Focus:

The paper introduces ExVivoMicroTest, an approach for automatic regression testing of microservices. It focuses on analyzing the execution of deployed services at runtime to generate test cases for future versions of the same services. The approach uses cloud technologies, particularly containers, to generate a mocked environment that isolates the service under test from the rest of the system.

Methodology:

The methodology involves two main phases: in-field monitoring and in-house test execution. The monitoring phase collects interaction traces from the production environment, which are then processed to generate test cases and mock services. The in-house phase involves synthesizing these test cases and mocks to validate the updated version of the microservice.

Tools and Frameworks:

ExVivoMicroTest framework, Docker, ELK, Prometheus, Monasca, CloudHealth, Varys, REST APIs, MongoDB Wire Protocol, Tcpdump, Python for test case generation.

Findings:

The study shows that ExVivoMicroTest can effectively reveal faults by generating in-house executions that reflect field usage scenarios. The approach was evaluated on the Piggy-Metrics open-source microservices application, revealing both real and injected regression faults that might go unnoticed by developers using regular in-house test cases.

Metrics:

The primary metrics used for evaluation include the ability to detect regression faults and the effectiveness of reproducing field usage scenarios in in-house tests.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework (ExVivoMicroTest) for automating regression testing in microservices, including the use of various tools and methodologies for test generation and execution.

10 Automatic Performance Monitoring and Regression Testing During the Transition from Monolith to Microservices

Publication Details:

- **Authors:** Andrea Janes, Barbara Russo
- **Title:** Automatic Performance Monitoring and Regression Testing During the Transition from Monolith to Microservices
- **Publication Year:** 2019
- **Venue:** IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)

Research Focus:

The paper introduces PPTAM+, a tool for continuous performance monitoring and regression testing during the transition from monolithic to microservice architectures. The focus is on ensuring that the performance of the system does not degrade during the migration process by continuously collecting and analyzing performance data.

Methodology:

The methodology involves several steps: logging user interactions with the monolithic system to create a performance baseline, using the extended PPTAM+ tool to test various configurations of the new microservice system, and comparing the performance of the original monolith with the new system. The approach leverages continuous feedback in a DevOps process to tune development and deployment.

Tools and Frameworks:

The paper mentions several tools and frameworks including PPTAM+, ELK stack (Elasticsearch, Logstash, Kibana), Docker, BenchFlow DSL, Aspect Oriented Programming, and RShiny for visualization.

Findings:

The study finds that PPTAM+ can effectively monitor performance degradation and provide feedback during the migration process. The tool helps ensure that the performance of the new microservice architecture meets or exceeds that of the original monolith. The approach allows stakeholders to make informed decisions based on real-time performance data.

Metrics:

Metrics used in the study include response times, usage intensities, and performance thresholds. The tool also supports time series analysis and statistical evaluations using R scripts.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a comprehensive approach and framework (PPTAM+) for automating performance monitoring and regression testing in microservice architectures.

11 Automation of Regression Test in Microservice Architecture

Publication Details:

- **Authors:** Mohammad Javad Kargar, Alireza Hanifizade
- **Title:** Automation of Regression Test in Microservice Architecture
- **Publication Year:** 2018
- **Venue:** IEEE 4th International Conference on Web Research (ICWR)

Research Focus:

The paper proposes an automated method for running regression tests within the Continuous Delivery pipeline in microservice architectures. The approach aims to ensure the reliability and operability of new microservice versions by comparing their performance and functionality to previous versions using various metrics.

Methodology:

The methodology involves placing regression tests within the Continuous Delivery pipeline. The process includes logging user interactions with the monolithic system to create a performance baseline, testing new configurations of the microservice system using the extended PPTAM+ tool, and comparing the performance and functionality of the original monolith with the new microservice system. The methodology leverages continuous feedback within a DevOps framework to tune development and deployment.

Tools and Frameworks:

The paper mentions the use of several tools and frameworks, including Jenkins for Continuous Delivery, Docker for containerization, Kubernetes for deployment, and Diffy by Twitter for regression testing.

Findings:

The study finds that the proposed automated regression testing method can effectively ensure the reliability and operability of new microservice versions. By using input traffic from the production environment, the method significantly reduces the cost and time required for testing. The approach helps maintain system performance and prevents the deployment of faulty versions.

Metrics:

Metrics used in the study include resource usage rate, system failure rate, and system performance metrics such as response times and performance thresholds.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a comprehensive approach and framework for automating regression testing within the Continuous Delivery pipeline in microservice architectures.

12 Benchmarks for End-to-End Microservices Testing

Publication Details:

- **Authors:** Sheldon Smith, Ethan Robinson, Timmy Frederiksen, Trae Stevens, Tomas Cerny, Miroslav Bures, Davide Taibi
- **Title:** Benchmarks for End-to-End Microservices Testing
- **Publication Year:** 2023
- **Venue:** Not specified in the document snippet

Research Focus:

The paper aims to provide the microservices community with a benchmark for end-to-end testing. It introduces a test benchmark containing full functional testing coverage for two well-established open-source microservice systems. The benchmark is designed to simplify experiments with testing and traffic simulation, demonstrating effective strategies and overcoming certain challenges in testing microservices.

Methodology:

The methodology includes creating a comprehensive test suite benchmark for two open-source microservice systems: Train-Ticket and eShopOnContainers. The approach involves functional and load testing using various tools and frameworks. Functional regression testing is performed using automated web-based tests through Selenium. Load testing is conducted using Gatling to simulate different load conditions and measure system response times.

Tools and Frameworks:

The paper mentions the use of Selenium for functional regression testing and Gatling for load testing. Other tools and frameworks include Docker, TestNG for parallelizing tests, HTML Unit WebDriver, and various automation tools for performance monitoring.

Findings:

The study results indicate that the test benchmarks for the Train-Ticket and eShopOnContainers systems effectively cover functional and load testing scenarios. The benchmarks reveal bottlenecks and areas of performance degradation under different load conditions. The study highlights the importance of using comprehensive and automated test suites to ensure the reliability and performance of microservice architectures.

Metrics:

Metrics used in the study include response times, the percentage of requests exceeding a certain response time threshold (800ms), and system performance under varying load conditions. The study also measures test coverage and the effectiveness of parallelized test execution.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for functional and load testing of microservice systems, including the use of Selenium and Gatling.

13 Challenges in Regression Test Selection for End-to-End Testing of Microservice-based Software Systems

Publication Details:

- **Authors:** Daniel Elsner, Daniel Bertagnolli, Alexander Pretschner, Rudi Klaus
- **Title:** Challenges in Regression Test Selection for End-to-End Testing of Microservice-based Software Systems
- **Publication Year:** 2022
- **Venue:** IEEE/ACM 3rd International Conference on Automation of Software Test (AST)

Research Focus:

The paper addresses the challenges in regression test selection (RTS) for end-to-end testing of microservice-based software systems. It introduces a distributed RTS approach called microRTS, which targets automated and manual end-to-end testing in microservice architectures. The focus is on reducing testing efforts by selecting relevant tests using execution traces.

Methodology:

The methodology includes implementing microRTS on top of distributed tracing infrastructure and Java bytecode manipulation libraries. The approach involves collecting per-test execution traces at method-level granularity and performing change-based test selection. The study evaluates microRTS through a case study on the German COVID-19 contact tracing application (Corona-Warn-App) and the TeaStore microservice benchmark application.

Tools and Frameworks:

The paper mentions the use of OpenTelemetry for distributed tracing, Java Agent and ByteBuddy for bytecode instrumentation, and AndroidBuddy for mobile client instrumentation. Additionally, Docker-Compose is used for orchestrating services, and Cypress is used for implementing automated end-to-end tests.

Findings:

The study finds that microRTS can reduce the testing effort by 10-50% through semi-automated filtering of test traces. It highlights that method-level granularity is more effective than class-level for RTS in manual end-to-end testing. However, the precision of test specifications significantly impacts the effectiveness of RTS. The study also identifies the importance of instrumenting client code for comprehensive test traces.

Metrics:

Metrics used in the study include instrumentation overhead during system startup and testing, as well as the reduction in the number of selected tests. The study measures performance impacts such as startup time (+67.5%) and runtime overhead (+18%).

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a comprehensive approach and framework (microRTS) for automated and manual end-to-end regression test selection in microservice architectures.

14 Comparison of Runtime Testing Tools for Microservices

Publication Details:

- **Authors:** Juan P. Sotomayor, Sai Chaithra Allala, Patrick Alt, Justin Phillips, Tariq M. King, Peter J. Clarke
- **Title:** Comparison of Runtime Testing Tools for Microservices
- **Publication Year:** 2019
- **Venue:** IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)

Research Focus:

The paper provides a comparative survey of several open-source tools used for testing microservices at runtime. It focuses on the added complexity of network communication between services and the need for runtime testing to monitor the behavior of the different components in a production environment. The study also describes the design of a polyglot Ridesharing microservices reference prototype and presents the results of a study measuring the overhead of runtime testing on this application.

Methodology:

The methodology includes creating a polyglot Ridesharing microservices reference prototype and using several tools to test the prototype. The tools are evaluated based on their setup, execution, and performance overhead. The study involves component and integration testing on the Ridesharing application, with tools like Gremlin, Hoverfly, Minikube, and Telepresence used to perform the tests. The execution times of system test cases are measured and compared across different tools.

Tools and Frameworks:

The paper mentions the use of various tools for runtime testing:

- **Gremlin**: For fault injection and resilience testing.
- **Hoverfly**: For capturing and simulating HTTP traffic between services.
- **Minikube**: For deploying services inside Kubernetes pods.
- **Telepresence**: For debugging services locally while connecting them to a production environment.

Findings:

The study finds significant variability in the execution times of different testing tools. Telepresence exhibited the highest execution times, while Hoverfly showed the least increase in execution time compared to the baseline (Docker CE). The paper highlights the challenges in setting up and configuring the tools, especially with respect to matching container environments and managing communication ports. It emphasizes the importance of selecting appropriate tools based on the specific requirements and constraints of the microservices system under test.

Metrics:

Metrics used in the study include:

- Total execution time for running system test cases.
- Mean execution time per iteration.
- Standard deviation, maximum, and minimum execution times.
- Percentage increase in execution time over the baseline (Docker CE).

Categorization:

- **Impact on Quality and Reliability:** The paper evaluates the effects of different runtime testing tools on the performance and reliability of microservices.

15 Consumer-Driven Contract Tests for Microservices: A Case Study

Publication Details:

- **Authors:** Jyri Lehvä, Niko Mäkitalo, Tommi Mikkonen
- **Title:** Consumer-Driven Contract Tests for Microservices: A Case Study
- **Publication Year:** 2019
- **Venue:** Product-Focused Software Process Improvement: 20th International Conference, PROFES 2019, Barcelona, Spain, November 27–29, 2019, Proceedings. Lecture Notes in Computer Science. Programming and Software Engineering, vol. 11915, Springer Nature Switzerland

Research Focus:

The paper explores how systems based on microservice architecture and their integrations can be tested more effectively by extending the testing approach with consumer-driven contract tests. It investigates the responsibilities and purposes of each testing method when introducing consumer-driven contract tests to the system.

Methodology:

The methodology involves a case study on a microservice system consisting of eight services and four databases. The system was gradually transitioned from a monolithic architecture to a microservice architecture. The study focuses on two specific endpoints: Save and Order. Consumer-driven contract tests were implemented using Pact JS and Pact Broker to share and verify contracts between services. The tests were run on a local machine and not in a Continuous Integration (CI) environment.

Tools and Frameworks:

The paper mentions the use of Pact JS for implementing consumer-driven contract tests and Pact Broker for sharing the contracts. Other tools mentioned include MySQL databases and Express applications written in JavaScript.

Findings:

The study finds that consumer-driven contract tests can effectively reveal integration defects that other testing methods might miss. They bring confidence to the testing strategy by ensuring that integrations between services are thoroughly tested. The tests were found to be fast, stable, and deterministic. The study also highlights the importance of good communication between teams for implementing and maintaining consumer-driven contract tests.

Metrics:

None.

Categorization:

- **Impact on Quality and Reliability:** The paper evaluates the effects of consumer-driven contract tests on the quality and reliability of microservice-based systems, demonstrating their effectiveness in detecting integration issues.

16 Contract Testing in Microservices-Based Systems: A Survey

Publication Details:

- **Authors:** Manuel Simosa, Frank Siqueira
- **Title:** Contract Testing in Microservices-Based Systems: A Survey
- **Publication Year:** 2023
- **Venue:** IEEE 14th International Conference on Software Engineering and Service Science (ICSESS)

Research Focus:

The paper systematically identifies, analyzes, and classifies publication trends, research topics, approaches, tools, and challenges for communication between microservices, specifically focusing on the use of contract tests. It aims to provide a comprehensive understanding of the current state of research and practices in contract testing for microservices.

Methodology:

The study employs a systematic mapping method to review the literature on contract testing in microservices. It includes snowball sampling stages and demographic filters to qualitatively evaluate the studies. The process involved an initial set of 496 articles, which was narrowed down to 25 primary studies that were analyzed to answer three research questions regarding classification, strategies, techniques, tools, and challenges in contract testing.

Tools and Frameworks:

The paper mentions several tools and frameworks used for contract testing in microservices:

- **Pact and Pact Broker**: These tools are widely used for consumer-driven contract testing.
- Other tools mentioned are used in various stages of development and deployment for validating contracts and ensuring integration.

Findings:

The study identifies three main research themes: Microservices Architecture, Quality Attributes, and Similarity Analysis. It highlights that contract testing is crucial for ensuring integration and managing the evolution of microservices. Challenges identified include defining the granularity of microservices, ensuring integration, and automating tests to reduce costs. The study emphasizes the need for well-defined testing strategies and the use of contract testing to manage the complexity of microservices-based systems.

Metrics:

None.

Categorization:

- **Systematic Study Contributions:** The paper provides a comprehensive survey and systematic mapping study on contract testing in microservices, identifying trends, challenges, and tools in the field.

17 Design of Flight Test Mission Evaluation System Based on Microservice

Publication Details:

- **Authors:** Jun Li, Yaojia Nie
- **Title:** Design of Flight Test Mission Evaluation System Based on Microservice
- **Publication Year:** 2023
- **Venue:** 2023 7th International Conference on Electronic Information Technology and Computer Engineering (EITCE 2023), Xiamen, China

Research Focus:

The paper addresses the challenges of high module coupling, limited data processing capability, and single evaluation function in flight test mission evaluation systems based on monolithic architecture. It proposes a distributed system based on microservice architecture, emphasizing data processing, dynamic extension, and intelligent analysis to improve evaluation quality and efficiency.

Methodology:

The methodology involves designing a logically hierarchical distributed architecture with five layers: facility layer, platform layer, service layer, application layer, and display layer. The system incorporates containerization using Docker and Kubernetes, distributed communication via gRPC, and a plugin framework for interactive terminal design. The evaluation data processing subsystem is core, handling data acquisition, computation, analysis, and storage.

Tools and Frameworks:

The paper mentions the use of several tools and frameworks, including Docker and Kubernetes for container and cluster management, Redis for the message bus, MySQL and Taos for data storage, Nginx for load balancing, and Apisix as the server gateway. Additionally, gRPC is used for service module calls, and the CTK plugin framework is used for visual and intelligent analysis.

Findings:

The system has been effectively implemented in the flight test of various aircraft types, playing a crucial role in reducing test cycles, ensuring safety, and enhancing efficiency. The system can simultaneously handle the evaluation of at least 26 aircrafts' missions, with an average data extraction speed within 80ms. The microservice architecture has significantly improved data processing capability and system scalability.

Metrics:

None.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for designing a flight test mission evaluation system using various tools and methodologies.

18 Design and Implementation of Intelligent Automated Testing of Microservice Application

Publication Details:

- **Authors:** Tingting Duan, Da Li, Jiaxing Xuan, Fanjie Du, Jing Li, Jing Du, Shang Wu
- **Title:** Design and Implementation of Intelligent Automated Testing of Microservice Application
- **Publication Year:** 2021
- **Venue:** IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)

Research Focus:

The paper proposes an intelligent automated testing framework for microservice applications, leveraging a decision tree algorithm to classify and screen different types of test cases. The focus is on improving test efficiency, reducing labor costs, and enhancing the accuracy and comprehensiveness of test case classification.

Methodology:

The methodology includes designing a basic framework for automated testing, divided into four functional modules: Test Case Library, Decision Tree Module, Test Service Module, and Test Log Module. The decision tree module, which is the core of the framework, involves data preprocessing, model training, and test case generation. The ID3 algorithm is used to construct the decision tree, which classifies test cases based on their attributes.

Tools and Frameworks:

The paper mentions the use of the following tools and frameworks:

- **Decision Tree Algorithm (ID3)**: For classification and screening of test cases.
- Docker and Kubernetes: For container and cluster management.
- Redis: For the message bus.
- MySQL and Taos: For data storage.
- Nginx: For load balancing.
- Apisix: As the server gateway.
- gRPC: For service module calls.
- CTK Plugin Framework: For visual and intelligent analysis.

Findings:

The system effectively reduces test cycles and improves test efficiency. The introduction of the decision tree algorithm for test case classification significantly enhances the accuracy of the classification while expanding the scope of test cases. The framework demonstrates high performance with an average accuracy of 94.1

Metrics:

None.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for intelligent automated testing using a decision tree algorithm, along with various tools and frameworks to manage and execute the tests.

19 Design and Research of Microservice Application Automation Testing Framework

Publication Details:

- **Authors:** Yang Wang, Lei Cheng, Xin Sun
- **Title:** Design and Research of Microservice Application Automation Testing Framework
- **Publication Year:** 2019
- **Venue:** 2019 International Conference on Information Technology and Computer Application (ITCA)

Research Focus:

The paper addresses the challenges of ensuring quality and reliability in microservice applications through automated testing. It proposes a hierarchical data-driven testing framework tailored for microservice applications to improve testing efficiency, work efficiency, and effective management.

Methodology:

The methodology includes designing a modular and layered automation testing framework with six layers: adaptation layer, data layer, test case layer, execution layer, analysis layer, and management layer. The framework employs data-driven technology to separate test logic from test data, improving maintainability and reusability. Test case generation is automated based on an extended interface description file, focusing on functional and performance testing.

Tools and Frameworks:

The paper mentions the use of several tools and frameworks:

- WADL (Web Application Description Language): For describing service interfaces.
- Docker and Kubernetes: For container and cluster management.
- Redis: For the message bus.
- MySQL and Taos: For data storage.
- Nginx: For load balancing.
- Apisix: As the server gateway.
- gRPC: For service module calls.
- CTK Plugin Framework: For visual and intelligent analysis.
- WebInject, WStest, SoapUI, TestMaker, QEngine, JMeter: For executing automated tests.

Findings:

The proposed framework significantly improves software testing capabilities and accelerates the implementation of microservice applications. The framework's modularity and data-driven approach enhance test efficiency, reduce the complexity of test management, and ensure comprehensive test coverage. The use of extended WADL files and parameter combination principles effectively automates test case generation, reducing manual effort.

Metrics:

None.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for automated testing of microservice applications, including various tools and methodologies to manage and execute tests.

20 Design, Implementation, and Testing of a Microservices-based Digital Twins Framework for Network Management and Control

Publication Details:

- **Authors:** Alfio Lombardo, Giacomo Morabito, Salvatore Quattropani, Carmelo Ricci
- **Title:** Design, Implementation, and Testing of a Microservices-based Digital Twins Framework for Network Management and Control
- **Publication Year:** 2022
- **Venue:** IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)

Research Focus:

The paper introduces the mBandit framework, which leverages Digital Twins (DT) technology and microservices for managing and controlling Software Defined Networking (SDN). It aims to enhance network management by enabling autonomous operation through interaction between DTs and the network operating system.

Methodology:

The methodology involves designing a microservices-based platform using Docker and Kubernetes to support the implementation of Digital Twins. The framework includes a layered architecture comprising the Physical Layer, Connection Layer (with sublayers for Proxy, Connectors, and Data Concentrator), and the Digital Twins Layer. The study demonstrates the implementation through a use case involving an SDN network managed by the mBandit framework.

Tools and Frameworks:

The paper mentions the use of several tools and frameworks:

- Docker and Kubernetes: For container and cluster management.
- Apache Kafka: For managing data streams from multiple sources.
- ONOS (Open Network Operating System): As the network operating system.
- Mininet: For emulating the network topology.
- AI models: For optimizing traffic and managing network configurations.

Findings:

The study finds that the mBandit framework effectively enhances the capabilities of network management by automating routine tasks and optimizing network performance. The implementation demonstrates that the framework can dynamically adjust network configurations to improve quality of experience for users. The use of microservices ensures scalability and adaptability to various application contexts and execution environments.

Metrics:

The paper does not specify performance metrics used to evaluate the quality and reliability of the microservices tests.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for using Digital Twins and microservices technology for network management and control, including various tools and methodologies for implementation and testing.

21 Efficient Software Test Management System Based on Microservice Architecture

Publication Details:

- **Authors:** Peng Jiang, Yanhua Shen, Yonghui Dai
- **Title:** Efficient Software Test Management System Based on Microservice Architecture
- **Publication Year:** 2022
- **Venue:** IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)

Research Focus:

The paper proposes a method for establishing an efficient software test management system using microservice architecture. It focuses on overcoming the challenges associated with high module coupling, limited data processing capability, and the complexity of traditional single application architectures in software test management.

Methodology:

The methodology involves designing a software test management system framework based on microservice architecture. The system includes several key technologies such as Redis cluster, load balancing of service gateway, distributed lock, and master-slave structure of Redis. The system is designed to handle various testing phases including requirements testing, design testing, unit testing, functional testing, integration testing, performance testing, and maintenance testing.

Tools and Frameworks:

The paper mentions the use of several tools and frameworks:

- **Redis Cluster**: For high-performance key-value in-memory database with master-slave structure.
- **Docker and Kubernetes**: For container and cluster management.
- **Netflix Zuul**: For load balancing and service gateway management.
- **Zookeeper**: For distributed locking.
- **SpringBoot**: For microservice deployment.

Findings:

The study finds that the proposed microservice-based software test management system significantly reduces module coupling and improves data processing capabilities. The system demonstrates enhanced flexibility and scalability, making it suitable for complex software test management scenarios. The implementation of Redis for distributed caching and Netflix Zuul for load balancing ensures efficient resource management and high system performance.

Metrics:

None.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for establishing an efficient software test management system using various tools and methodologies related to microservice architecture.

22 End-to-End Test Coverage Metrics in Microservice Systems: An Automated Approach

Publication Details:

- **Authors:** Amr S. Abdelfattah, Tomas Cerny, Jorge Yero Salazar, Austin Lehman, Joshua Hunter, Ashley Bickham, Davide Taibi
- **Title:** End-to-End Test Coverage Metrics in Microservice Systems: An Automated Approach
- **Publication Year:** 2023
- **Venue:** arXiv

Research Focus:

The paper introduces test coverage metrics for evaluating the extent of end-to-end (E2E) test suite coverage for microservice endpoints. It presents an automated approach to compute these metrics to provide feedback on the completeness of E2E test suites. The focus is on ensuring that all middleware endpoints scattered across microservices are tested comprehensively.

Methodology:

The methodology involves proposing three metrics to assess endpoint coverage in E2E testing: microservice endpoint coverage, test case endpoint coverage, and complete test suite endpoint coverage. The approach includes static and dynamic analysis methods to collect data for calculating these metrics. The process comprises four stages: endpoint extraction from source code, endpoint extraction from log traces, coverage calculation, and coverage visualization. A proof-of-concept tool is implemented and a case study is conducted on a well-established system benchmark to demonstrate the approach.

Tools and Frameworks:

The paper mentions the use of the following tools and frameworks:

- **JavaParser**: For static analysis of Java source code.
- **JUnit and Selenium**: For executing test suites.
- **Elasticsearch**: For storing and analyzing log traces.
- **Prophet library**: For service dependency graph visualization.
- **Apache Maven**: For build automation.

Findings:

The study demonstrates that the proposed metrics provide valuable insights into the extent of test coverage achieved by E2E test suites. The case study results show that the automated approach can effectively capture all endpoints and measure their coverage. The metrics help identify gaps in test suites and ensure comprehensive testing of microservice systems. The tool implementation and visualization techniques offer practical benefits for continuous quality assurance.

Metrics:

The paper proposes three metrics for evaluating test coverage:

- **Microservice Endpoint Coverage**: The percentage of endpoints tested within each microservice.
- **Test Case Endpoint Coverage**: The percentage of endpoints covered by each test case.
- **Complete Test Suite Endpoint Coverage**: The overall percentage of endpoints covered by the entire test suite.

Categorization:

- **Metrics**: The paper introduces new performance metrics for assessing the effectiveness of E2E test suites in microservice systems, focusing on endpoint coverage.

23 An Overview of Fault Diagnosis for Test Alarms in Microservices through Multi-source Data

Publication Details:

- **Authors:** Shenglin Zhang, Jun Zhu, Bowen Hao, Yongqian Sun, Xiaohui Nie, Jingwen Zhu, Xilin Liu, Xiaoqian Li, Yuchi Ma, Dan Pei
- **Title:** Fault Diagnosis for Test Alarms in Microservices through Multi-source Data
- **Publication Year:** 2024
- **Venue:** ACM International Conference on the Foundations of Software Engineering (FSE Companion '24)

Research Focus:

The paper addresses the issue of diagnosing test alarms in microservices by utilizing multi-source data, including execution logs, trace logs, and test case information, to automatically classify and localize faults.

Methodology:

The research utilizes a novel framework called SynthoDiag, which integrates a knowledge graph-based approach to analyze multi-source logs. The methodology includes the Entity Fault Association and Position Value (EFA-PV) algorithm for localizing fault-indicative log entries and a block-based differentiation approach for filtering out irrelevant logs.

Tools and Frameworks:

SynthoDiag framework, including the EFA-PV algorithm and block-based differentiation strategy.

Findings:

SynthoDiag significantly improves fault diagnosis accuracy, achieving a 21% improvement in Micro-F1 and a 30% improvement in Macro-F1 scores for fault classification compared to baseline methods. The framework also shows an 81.9% top-5 accuracy in fault localization, surpassing existing methods. Key strengths include its ability to handle complex and frequently updated microservices and efficient filtering of irrelevant logs. Weaknesses involve challenges in dealing with new types of logs due to frequent software updates.

Metrics:

Micro-F1 score, Macro-F1 score, and top-5 accuracy are used to evaluate the performance of fault classification and localization.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a comprehensive framework (SynthoDiag) and algorithms (EFA-PV) for fault diagnosis in microservices.

24 Fitness-guided Resilience Testing of Microservice-based Applications

Publication Details:

- **Authors:** Zhenyue Long, Guoquan Wu, Xiaojiang Chen, Chengxu Cui, Wei Chen, Jun Wei
- **Title:** Fitness-guided Resilience Testing of Microservice-based Applications
- **Publication Year:** 2020
- **Venue:** IEEE International Conference on Web Services (ICWS)

Research Focus:

The paper presents IntelliFT, a guided resilience testing technique designed for microservice-based applications. It aims to expose defects in the fault-handling logic efficiently within a fixed time limit by leveraging existing integration tests and a fitness-guided search technique.

Methodology:

The methodology involves using IntelliFT, which leverages the integration tests of applications under test to explore the fault space and uses a fitness-guided search technique to decide whether injected faults can lead to severe failures. The process includes the following steps:

1. Execute fault-free integration tests to compute initial injection points.
2. Randomly generate a batch of fault tests, execute them, and evaluate their impacts.
3. Choose a previously executed fault test with a high impact value.
4. Mutate the chosen fault test to obtain a new test, execute it, and evaluate its impact.
5. Repeat the process iteratively.

Tools and Frameworks:

The paper mentions the use of several tools and frameworks:

- Istio: For injecting basic fault types (Abort, Delay) and implementing other fault types (Overload, Hang, Disconnect, Crash).
- Jaeger: For distributed tracing.
- Z3 Solver: For computing injection points.
- Selenium WebDriver and Vista: For visual test execution.
- OpenCV: For using computer vision algorithms.

Findings:

The study finds that IntelliFT significantly improves the fault diagnosis process by effectively exposing fault-handling bugs within limited testing time. The experimental results show that the proposed technique outperforms existing methods by quickly identifying severe bugs in the recovery logic of microservices.

Metrics:

The paper evaluates the effectiveness of the proposed technique using metrics such as the number of exposed unique test failures, the number of explored fault tests, and the number of backup paths triggered.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for resilience testing using IntelliFT, a fitness-guided fault injection and testing tool.

25 Fault Diagnosis for Test Alarms in Microservices through Multi-source Data

Publication Details:

- **Authors:** Xing Chen, Xiapu Luo, Han Li, Yong Zhang, Guangjie Du, Zhen Wang
- **Title:** Fault Diagnosis for Test Alarms in Microservices through Multi-source Data
- **Publication Year:** 2021
- **Venue:** IEEE International Conference on Service-Oriented System Engineering (SOSE)

Research Focus:

The paper addresses the issue of diagnosing faults in microservices through analyzing test alarms generated from various sources of data. The focus is on improving the efficiency and accuracy of fault diagnosis in complex microservices environments.

Methodology:

The study employs a combination of symbolic execution and automated reasoning to infer relations and dependencies among test programs. It uses a rule-based approach to analyze invocations of local and remote APIs during test executions, aiming to identify similarities and dependencies that help streamline the fault diagnosis process.

Tools and Frameworks:

None.

Findings:

The paper presents a methodology that effectively infers dependencies and similarities among test programs, thereby aiding in the fault diagnosis process. The approach is validated through a case study on a real-world microservice application, demonstrating its viability in identifying test program similarities and dependencies.

Metrics:

None.

Categorization:

- **Impact on Quality and Reliability**: The paper evaluates the effects of inferred similarities and dependencies on improving the fault diagnosis process in microservices, contributing to the overall reliability of the system.

26 JoT: A Jolie Framework for Testing Microservices

Publication Details:

- **Authors:** Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti, Florian Rademacher, Narongrit Unwerawattana
- **Title:** JoT: A Jolie Framework for Testing Microservices
- **Publication Year:** 2023
- **Venue:** COORDINATION 2023 - 25th International Conference on Coordination Models and Languages

Research Focus:

The paper introduces JoT, a testing framework for Microservice Architectures (MSAs) that focuses on technology agnosticism. The framework aims to reduce the complexity of testing MSAs that use different technology stacks and to facilitate writing and reusing tests involving multiple services and deployment configurations.

Methodology:

JoT utilizes the Jolie programming language to define tests as orchestrators that can consume or offer operations to the MSA under test. The methodology includes defining interfaces, access points, and test logic using Jolie. The framework supports flexible test deployment through configuration parameters that allow running the same tests under different settings.

Tools and Frameworks:

The framework relies on Jolie for writing tests, using its constructs for defining interfaces, ports, and test logic. JoT also employs technology-agnostic interfaces and configuration parameters to adapt tests to different deployment environments.

Findings:

JoT simplifies the testing of MSAs by supporting technology-agnostic test definitions, reducing the effort required to test services with different technology stacks. The framework enables efficient test reuse across different deployment configurations and improves the testing process for complex microservice interactions.

Metrics:

None.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed methodology for testing MSAs using the JoT framework, focusing on technology-agnostic test definitions and flexible deployment configurations.

27 Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection

Publication Details:

- **Authors:** Karl Meinke, Peter Nycander
- **Title:** Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection
- **Publication Year:** 2015
- **Venue:** International Conference on Testing Software and Systems (ICTSS)

Research Focus:

The paper explores the application of Learning-Based Testing (LBT) to evaluate the functional correctness and robustness of distributed microservice architectures, specifically focusing on fault injection and requirements testing.

Methodology:

The study employs Learning-Based Testing (LBT), which integrates machine learning with model checking in a feedback loop with the system under test (SUT). The methodology involves iterative refinement and extension of an initial model of the SUT using test cases as learner queries. The framework uses a case study of a commercial product for counter-party credit risk implemented as a distributed microservice architecture.

Tools and Frameworks:

The paper mentions the use of several tools and frameworks:

- **LBTest version 1.3.2**: For learning-based testing.
- **NuSMV**: For model checking.
- **RabbitMQ**: For event-based network communication in the microservice architecture.
- **IKL and L*Mealy**: Automata learning algorithms used in the testing sessions.

Findings:

The study finds that Learning-Based Testing (LBT) is feasible for both requirements and robustness testing of distributed microservice architectures. However, the high latency and non-determinism of the SUT necessitate new approaches to improve LBT coverage. The experimental results show that LBT can effectively identify faults in the SUT, but improvements are needed for handling distributed systems more efficiently.

Metrics:

The paper evaluates the effectiveness of the proposed technique using metrics such as the number of queries made by LBTest, the number of warnings generated, and the state model learned during the testing process.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper provides a detailed approach and framework for learning-based testing using LBTest and other tools for evaluating the functional correctness and robustness of distributed microservice architectures.

28 Microservice-Tailored Generation of Session-Based Workload Models for Representative Load Testing

Publication Details:

- **Authors:** Henning Schulz, Tobias Angerstein, Dušan Okanović, André van Hoorn
- **Title:** Microservice-Tailored Generation of Session-Based Workload Models for Representative Load Testing
- **Publication Year:** 2019
- **Venue:** 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)

Research Focus:

The paper focuses on developing and evaluating approaches for tailoring representative load tests to target specific microservices instead of the entire system. The goal is to create workload models that better represent the production workload for individual microservices, addressing the limitations of system-level testing in microservice architectures.

Methodology:

The study proposes two algorithms: log-based and model-based tailoring. The log-based algorithm transforms production request logs to generate tailored workloads for specific services. The model-based algorithm performs workload tailoring on the level of the workload model itself. The evaluation involves an experimental study using a representative microservice application, comparing both algorithms with system-level and request-based workload models in terms of test duration, infrastructure capacity, and representativeness.

Tools and Frameworks:

The study uses the WESSBAS approach for workload model generation, and tools such as Zipkin for trace collection, a Java service for converting traces into OPEN.xtrace, and Prometheus for collecting performance metrics.

Findings:

The tailored workload models were found to significantly reduce the required test execution time and the number of microservices to be deployed for the test. The model-based approach performed better than the log-based approach in terms of representativeness and qualitative characteristics of the generated models. However, both approaches slightly reduced representativeness compared to an untailored system-level test but were more representative than simple request-based tests.

Metrics:

The study used several performance metrics, including response time, CPU utilization, and memory consumption, to evaluate the quality and reliability of the microservices tests. Additionally, a distance metric based on the Kolmogorov-Smirnov statistic was used to assess the representativeness of the generated load tests.

Categorization:

- **Impact on Quality and Reliability:** The paper evaluates the effects of tailored workload models on the performance and reliability of microservice-based systems. It provides empirical evidence of the effectiveness of the proposed algorithms in generating representative workloads and reducing test execution time and resource requirements.

29 Microservices Integrated Performance and Reliability Testing

Publication Details:

- **Authors:** Matteo Camilli, Antonio Guerriero, Andrea Janes, Barbara Russo, Stefano Russo
- **Title:** Microservices Integrated Performance and Reliability Testing
- **Publication Year:** 2022
- **Venue:** IEEE/ACM 3rd International Conference on Automation of Software Test (AST '22)

Research Focus:

The paper focuses on proposing MIPaRT, a methodology and platform for integrated performance and reliability testing of microservice systems. The specific aspects addressed include the automatic generation and execution of performance-reliability ex-vivo testing sessions, collection of monitoring data, computation of performance and reliability metrics, and integrated visualization of the results.

Methodology:

The methodology follows three stages:

1. Definition of the operating conditions based on usage data collected from Ops, including workload specification, behavioral models, workload intensity, and behavior mix.
2. Execution of ex-vivo testing sessions with synthetic users replicating the defined operating conditions.
3. Integrated analysis to compute and visualize performance and reliability estimates.

The study employs controlled experiments using the Train Ticket benchmark microservices system to evaluate MIPaRT.

Tools and Frameworks:

The platform integrates multiple modules, including:

- Docker for deploying microservices.
- Locust for generating workload intensity.
- EvoMaster for generating request classes.
- Apache Zeppelin for interactive visualization.
- BenchFlow domain-specific language for defining operational conditions.

Findings:

Key results include:

- Integrated performance-reliability testing provides additional insights compared to verifying the two qualities in isolation.
- MIPaRT can detect existing relationships between performance and reliability issues.
- Evolutionary changes and operational changes have distinct impacts on performance and reliability, which can be quantified and visualized using MIPaRT.

Strengths:

- Comprehensive methodology integrating performance and reliability testing.
- Automated generation of testing scenarios and metrics computation.
- Effective visualization for QA engineers to identify critical issues.

Weaknesses:

- The methodology relies on the availability of microservice specifications and usage data.
- The controlled experiments are limited to a specific benchmark system.

Metrics:

The performance metrics used include:

- Performance estimator (normalized distance from average response time to a performance threshold).
- Performance Degradation (PD).
- Reliability estimator (ratio of non-failing requests).
- Failed Requests (FR).
- Connection Errors ratio (CE).
- Server Errors ratio (SE).

Categorization:

- **Category:** Testing Techniques, Tools, and Frameworks
- **Reason:** The paper presents a novel methodology and platform (MIPaRT) for integrated performance and reliability testing, along with specific tools and frameworks used for test generation and analysis in microservices.

30 Microservices: A Performance Tester's Dream or Nightmare?

Publication Details:

- **Authors:** Simon Eismann, Cor-Paul Bezemer, Weiyi Shang, Dušan Okanović, André van Hoorn
- **Title:** Microservices: A Performance Tester's Dream or Nightmare?
- **Publication Year:** 2020
- **Venue:** Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE '20), April 20–24, 2020, Edmonton, AB, Canada

Research Focus:

This paper discusses the benefits and challenges of microservices from a performance tester's point of view. It demonstrates how microservices affect the performance testing process through experiments on the TeaStore application, focusing on stability of execution environments, performance testing results across repeated runs, and detection of performance regressions.

Methodology:

The study employs a series of experiments on the TeaStore application deployed on Google Kubernetes Engine (GKE). Different scenarios were tested, including Default, Balanced, LargeVMs, and Autoscaling setups, to observe stability and performance metrics under varied load conditions. Performance regressions were injected to analyze detection capabilities.

Tools and Frameworks:

Google Kubernetes Engine (GKE), TeaStore application, Terraform for cluster provisioning, Docker for container deployment, load driver by von Kistowski et al.

Findings:

Key findings include:

- Execution environments of microservices are not stable, even with a constant number of provisioned instances.
- Significant differences in performance test results were observed due to environmental variability.
- Performance regressions can be detected, but require multiple test repetitions to account for variance.

The paper also discusses the benefits of microservices for performance testing, such as containerization and easy access to metrics, alongside challenges like environment stability and reproducibility of test results.

Metrics:

Performance metrics used include response time, CPU busy time, and resource utilization.

Categorization:

- **Impact on Quality and Reliability**: The study evaluates the effects of performance testing techniques on the overall quality and reliability of microservice-based systems, focusing on challenges in ensuring stable and reproducible test results.

31 Microusity: A Testing Tool for Backends for Frontends (BFF) Microservice Systems

Publication Details:

- **Authors:** Pattarakrit Rattanukul, Chansida Makaranond, Pumipat Watanakulcharus, Chaiyong Ragkhitwetsagul, Tanapol Nearunchorn, Vasaka Visoottiviseth, Morakot Choetkiertikul, Thanwadee Sunetnanta
- **Title:** Microusity: A Testing Tool for Backends for Frontends (BFF) Microservice Systems
- **Publication Year:** 2023
- **Venue:** arXiv preprint arXiv:2302.11150

Research Focus:

The paper presents Microusity, a novel tool designed for RESTful API testing in Backends for Frontends (BFF) microservice systems. The tool aims to address challenges in tracing and identifying errors in BFF microservices, providing detailed error reports and graph-based visualizations to help developers debug issues.

Methodology:

The methodology involves the design and implementation of Microusity, which combines RESTful API fuzzing with request tracking and visualization. The tool uses RESTler for fuzzing and Zeek for network monitoring. The approach includes:

1. BFF API fuzzing and request tracking: Mapping main requests to sub-requests.
2. Reporting mechanisms: Generating error reports and graph visualizations.

The tool was evaluated through user studies with software practitioners to assess its usability and effectiveness.

Tools and Frameworks:

The study mentions the use of several tools:

- **RESTler:** For RESTful API fuzzing.
- **Zeek:** For network monitoring and request tracking.
- **Cytoscape.js:** For graph visualization.
- **HTML, CSS, Bootstrap, and EJS:** For the front-end implementation.

Findings:

Key findings include:

- Microusity effectively traces errors from BFF to backend microservices, providing detailed error and graph reports.
- The tool is useful for identifying and debugging security issues and unhandled errors in BFF systems.
- User evaluations indicated that the tool's reports are clear and helpful for debugging, with high usability scores.

Strengths:

- Automated tracing and error reporting for BFF microservices.
- Clear visualization of request mappings and errors.

Weaknesses:

- Initial setup and learning curve for new users.
- Requires familiarity with RESTler and network monitoring concepts.

Metrics:

The paper did not explicitly mention specific performance metrics used to evaluate the quality and reliability of microservices tests. The focus was on qualitative user feedback regarding the tool's effectiveness and usability.

Categorization:

- **Testing Techniques, Tools, and Frameworks**: The paper presents a novel tool (Microusity) for testing BFF microservices, combining RESTful API fuzzing and network monitoring to trace and visualize request mappings and errors.

32 RAML-Based Mock Service Generator for Microservice Applications Testing

Publication Details:

- **Authors:** Nikita Ashikhmin, Gleb Radchenko, Andrei Tchernykh
- **Title:** RAML-Based Mock Service Generator for Microservice Applications Testing
- **Publication Year:** 2017
- **Venue:** Russian Supercomputing Days 2017

Research Focus:

This paper focuses on the development of a system that automatically generates mock services for RESTful microservices based on RAML (RESTful API Modeling Language) specifications. The generated mock services are packaged as Docker containers to facilitate deployment and testing in isolated environments.

Methodology:

The methodology involves creating a mock service generator that takes a RAML specification as input and produces a mock service deployed as a Docker container. The process includes:

1. Parsing the RAML specification to understand service endpoints and expected behaviors.
2. Generating the mock service code to handle HTTP methods such as GET, POST, PUT, and DELETE.
3. Building the mock service into a Docker container for easy deployment.

The implementation uses Python scripts and the Flask framework to handle HTTP requests, with additional validation and response generation based on the RAML file.

Tools and Frameworks:

The study mentions the use of several tools and frameworks:

- **RAML:** Used for specifying the interface of the RESTful services.
- **Docker:** Used for packaging the mock services as containers.
- **Flask:** A Python web framework used for implementing the mock service.
- **Elizabeth:** A library used for generating dummy data in responses.

Findings:

Key findings include:

- The proposed system can automatically generate deployable mock services based on RAML specifications, significantly reducing the time and effort required for testing microservice interactions.
- The mock services provide valid responses based on RAML specifications, including error handling for invalid requests.
- Integration with continuous integration systems showed a reduction in testing time by approximately 35

Strengths:

- Automated generation of mock services reduces manual effort.
- Use of Docker containers ensures isolated and consistent testing environments.

Weaknesses:

- The system currently supports only RAML v0.8.
- Initial setup and learning curve for new users may be high.

Metrics:

The paper did not explicitly mention specific performance metrics used to evaluate the quality and reliability of microservices tests. The focus was on qualitative improvements in the testing process and integration with continuous integration systems.

Categorization:

- **Testing Techniques, Tools, and Frameworks**: The paper presents a novel tool (RAML-Based Mock Service Generator) for testing microservices, combining RAML specification parsing and Docker containerization to generate and deploy mock services.

33 Research on Microservice Application Testing Based on Mock Technology

Publication Details:

- **Authors:** Du Lin, Fang Liping, Han Jiajia, Tang Qingzhao, Sun Changhua, Zhang Xiaohui
- **Title:** Research on Microservice Application Testing Based on Mock Technology
- **Publication Year:** 2020
- **Venue:** 2020 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)

Research Focus:

The paper proposes a novel construction method for microservice application systems based on Mock technology to improve the accuracy and efficiency of testing microservice systems. It addresses the challenges of dependency management and error isolation in microservice applications by using mock and stub technologies.

Methodology:

The methodology involves the design and implementation of a mock test platform that supports microservice interface management, mock data generation, and test environment setup. The process includes:

1. Analyzing and studying mock and stub technologies.
2. Developing a test scheme for unit and integration testing based on microservice requirements.
3. Implementing supporting test tools for the developed test scheme.
4. Deploying the test tools and integrating them into the DevOps process for continuous feedback and optimization.

The architecture of the mock test platform is detailed, providing efficient and powerful service interface management and test case management functionalities.

Tools and Frameworks:

The paper mentions the use of several tools:

- **Mock.js, EasyMock, Mockito:** For generating mock objects and services.
- **WireMock:** For simulating APIs in microservice architectures.

Findings:

Key findings include:

The mock test platform effectively removes external dependencies in unit and integration testing, enhancing test accuracy and stability.

The use of mock technology improves the portability and stability of test cases, allowing them to run independently of external services.

The platform supports test interface configuration, mock data management, and provides a flexible mock gateway for routing interface calls. Strengths:

Improved test accuracy and efficiency by isolating external dependencies.

Enhanced flexibility and performance in developing and testing microservice applications. Weaknesses:

Initial setup and configuration complexity.

Dependency on the accuracy of mock data and configurations.

Metrics:

The paper did not explicitly mention specific performance metrics used to evaluate the quality and reliability of microservices tests. The focus was on qualitative improvements in testing processes and the integration with DevOps for continuous feedback.

Categorization:

Impact on Quality and Reliability: The paper's focus on improving the testing process's accuracy, stability, and efficiency through mock technology directly impacts the overall quality and reliability of microservice applications.

34 Research on Microservice Application Testing System

Publication Details:

- **Authors:** Hongwei Li, Junsheng Wang, Hua Dai, Bang Lv
- **Title:** Research on Microservice Application Testing System
- **Publication Year:** 2020
- **Venue:** 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)

Research Focus:

The paper addresses the need for a comprehensive testing system for microservice applications within the State Grid Corporation of China. It proposes a microservice application testing system that includes six stages: unit test, integration test, single microservice test, contract test, end-to-end test, and cloud test. The focus is on improving test accuracy, stability, and efficiency through the use of mock technology and automation strategies.

Methodology:

The methodology involves the design and implementation of a testing system framework divided into six stages: unit test, integration test, single microservice test, contract test, end-to-end test, and cloud test. Each stage uses specific testing tools and strategies to ensure comprehensive coverage and reliability of microservice applications. The testing system is designed to be integrated into the DevOps process for continuous feedback and optimization.

Tools and Frameworks:

The paper mentions the use of various tools and frameworks for different stages of testing:

- **Unit Test:** Mock.js, EasyMock, Mockito, and Jenkins for process automation.
- **Integration Test:** Clumsy tools for simulating network failures.
- **Single Microservice Test:** Dependency Injection Frameworks.
- **Contract Test:** Pact for contract testing.
- **End-to-End Test:** Protractor for end-to-end testing.
- **Cloud Test:** Various cloud service platforms for test-as-a-service methods.

Findings:

Key findings include:

- The proposed testing system effectively addresses the complexities and dependencies in microservice applications, improving test accuracy and stability.
- The use of mock technology and automation strategies significantly enhances the efficiency and comprehensiveness of the testing process.
- The integration of the testing system into the DevOps process ensures continuous feedback and optimization, supporting the rapid development and deployment of microservice applications.

Strengths:

- Comprehensive coverage of microservice application testing through a multi-stage approach.
- Enhanced test accuracy and stability by isolating external dependencies.
- Improved efficiency through automation and continuous integration.

Weaknesses:

- Initial setup and configuration complexity.
- Dependency on the accuracy of mock data and configurations.

Metrics:

The paper did not explicitly mention specific performance metrics used to evaluate the quality and reliability of microservices tests. The focus was on qualitative improvements in testing processes and the integration with DevOps for continuous feedback.

Categorization:

- **Impact on Quality and Reliability:** The paper's focus on improving the testing process's accuracy, stability, and efficiency through mock technology directly impacts the overall quality and reliability of microservice applications.

35 An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study

Publication Details:

- **Authors:** Zhijie Lu
- **Title:** Research on Performance Optimization Method of Test Management System Based on Microservices
- **Publication Year:** 2023
- **Venue:** 2023 4th International Conference on Computer Engineering and Application (ICCEA)

Research Focus:

The paper focuses on addressing the challenges of optimizing the performance of microservice-based exam management systems. It proposes a method that utilizes system information grid optimization to enhance microservice performance in a closed-loop manner through microservice layout and information grid paradigm decomposition.

Methodology:

The methodology includes the establishment of a system information grid based on the deployment resources of the exam management system, deriving the microservice system's information map from the grid, and identifying performance blocking points through stress testing. Intelligent agents are set up at these blocking points to obtain the maximum entropy value of the information grid, which is then compared with the entropy value calculated through the grayscale table of the system information grid. The results are fed back to the microservice APM management control center for adjustments, forming a closed-loop optimization system.

Tools and Frameworks:

None.

Findings:

The study demonstrates that the proposed optimization method significantly improves the performance of the microservice-based exam management system. The system service response time and other performance indicators are reduced by an average of 36.58

Metrics:

The primary metric used in the study is the system service response time.

Categorization:

- **Impact on Quality and Reliability:** The paper evaluates the effects of performance optimization techniques on the overall quality and reliability of microservice-based systems. It provides empirical evidence of the effectiveness of these techniques in improving system performance.

36 Search-Based Performance Testing and Analysis for Microservice-Based Digital Power Applications

Publication Details:

- **Authors:** Qiugen Pei, Zheheng Liang, Zeling Wang, Lei Cui, Zhenyue Long, Guoquan Wu
- **Title:** Search-Based Performance Testing and Analysis for Microservice-Based Digital Power Applications
- **Publication Year:** 2023
- **Venue:** 6th International Conference on Energy, Electrical and Power Engineering (CEEPE)

Research Focus:

The paper focuses on developing a novel performance testing and analysis approach for microservice-based applications. It aims to identify performance issues and bottlenecks in microservice architectures, specifically within the context of digital power applications.

Methodology:

The study employs a multi-objective search-based profiling technique using a genetic algorithm. It actively generates diverse sequences of user requests to expose performance anomalies and diagnoses the root causes based on trace analysis. The methodology includes:

- Initial population creation from existing functional tests.
- Evolutionary search using crossover and mutation operators.
- Performance anomaly detection using the 3-sigma rule.
- Performance bottleneck identification through trace analysis and categorization.

Tools and Frameworks:

The paper mentions the use of distributed tracing tools such as Jaeger and Zipkin for capturing traces. It also uses Locust for load generation and Robot Framework for developing test cases.

Findings:

The proposed approach, PerfMS, effectively identifies performance issues and bottlenecks in microservice-based applications. The experimental results on an open-source benchmark (TrainTicket) and an industrial application (eSMM) demonstrate:

- Significant increase in execution time for evolved input combinations.
- Accurate identification of APIs with injected delays as performance bottlenecks.

Strengths:

- Effective in exposing performance issues and identifying bottlenecks.
- Utilizes existing functional tests to generate meaningful input sequences.

Weaknesses:

- Limited evaluation on only two applications.
- Dependence on the accuracy of initial functional tests.

Metrics:

The paper uses the following performance metrics:

- Execution time: Total elapsed time for request sequences.
- Response time: Latency of individual requests.
- Trace coverage: Number of exposed service operations in traces.
- Z-score: Used to measure the abnormality of an operation's self time.

Categorization:

- **Impact on Quality and Reliability:** The paper evaluates the effects of test generation techniques on the overall quality and reliability of microservices-based systems by demonstrating the identification of performance bottlenecks and anomalies.

37 Testing Microservices Architecture-Based Applications: A Systematic Mapping Study

Publication Details:

- **Authors:** Muhammad Waseem, Peng Liang, Gastón Márquez, Amleto Di Salle
- **Title:** Testing Microservices Architecture-Based Applications: A Systematic Mapping Study
- **Publication Year:** 2020
- **Venue:** 27th Asia-Pacific Software Engineering Conference (APSEC)

Research Focus:

The paper focuses on systematically identifying, analyzing, and classifying publication trends, research themes, approaches, tools, and challenges in the context of testing Microservices Architecture (MSA)-based applications.

Methodology:

The research used a Systematic Mapping Study (SMS) approach. The study followed guidelines proposed by Petersen et al. and Kitchenham et al. for conducting SMS and Systematic Literature Reviews (SLR). It involved a detailed search and selection process from seven major electronic databases, resulting in 33 primary studies.

Tools and Frameworks:

Several general-purpose tools are mentioned, such as JUnit for unit testing, Mockito for integration testing, Pact for contract testing, Selenium for end-to-end testing, and Nightmare for high-level browser automation. However, dedicated tools specifically for testing MSA-based applications were noted as lacking.

Findings:

Key findings include the identification of five research themes: automated testing, architecture, DevOps and CI, performance, and model-based testing. Integration and unit testing are the most mentioned approaches. Challenges include automated testing, inter-communication testing, faster test feedback, and integration testing. The study also highlights a lack of dedicated tools for testing MSA-based applications.

Metrics:

The paper does not specify particular performance metrics used to evaluate the quality and reliability of microservices tests.

Categorization:

- **Systematic Study Contributions:** The paper provides a systematic review of testing approaches, tools, and challenges in the context of MSA-based applications, identifying broader trends and aggregating knowledge from multiple studies.

38 An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study

Publication Details:

- **Authors:** Chu-Fei Wu, Shang-Pin Ma, An-Chi Shau, Hang-Wei Yeh
- **Title:** Testing for Event-Driven Microservices Based on Consumer-Driven Contracts and State Models
- **Publication Year:** 2022
- **Venue:** 2022 29th Asia-Pacific Software Engineering Conference (APSEC)

Research Focus:

This paper focuses on the testing of event-driven microservice systems using a software testing tool called CCTS (Composite Contract Testing Service). The tool combines consumer-driven contract testing with event-driven state models to verify the correctness of messaging and state transitions in microservices.

Methodology:

The research uses a combination of consumer-driven contract testing and state model-based verification. CCTS records state transitions of event exchanges between services, retrieves possible transition paths, and analyzes event logs to ensure they conform with specified transitions and paths. The methodology includes functional testing of CCTS using a real-world microservice system.

Tools and Frameworks:

CCTS (Composite Contract Testing Service) is the main tool introduced in the paper. It utilizes RabbitMQ as the message broker to manage events.

Findings:

The study found that CCTS can effectively detect potential defects in event-driven microservice systems, including isolated states, cyclic states, incomplete contract tests, and unqualified event sequences. The results of the functional testing of CCTS on a real-world microservice system demonstrated its effectiveness in identifying these issues.

Metrics:

The paper does not specify particular performance metrics used to evaluate the quality and reliability of microservices tests.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper introduces CCTS, a tool combining consumer-driven contract testing with state model verification to enhance the testing of event-driven microservices.

39 UTEMS: A Unit Testing Scheme for Event-driven Microservices

Publication Details:

- **Authors:** Seung-Yeob Lee, Chang-Hwan Lee, Kyeong-Woo Kim, Seong-Hwan Kim, Sungwon Kang
- **Title:** UTEMS: A Unit Testing Scheme for Event-driven Microservices
- **Publication Year:** 2023
- **Venue:** 2023 International Conference on Software Engineering and Knowledge Engineering (SEKE)

Research Focus:

The paper focuses on developing a unit testing scheme for event-driven microservices to improve test efficiency and reliability. It proposes UTEMS (Unit Testing Scheme for Event-driven Microservices) that facilitates the automated generation of unit tests for event-driven microservice systems.

Methodology:

The research utilizes a method for generating unit tests based on the behavior of event-driven microservices. It involves:

- Analyzing the event-driven architecture to identify key components and interactions.
- Generating unit test cases automatically using a predefined schema.
- Validating the generated tests through an empirical study.

Tools and Frameworks:

The paper mentions the use of UTEMS, the proposed tool for automated unit test generation. Specific details about other tools and frameworks are not provided.

Findings:

The study finds that UTEMS significantly improves the efficiency and coverage of unit testing in event-driven microservice systems. Key findings include:

- Automated test generation reduces the time required for creating unit tests manually.
- The proposed scheme improves the fault detection rate in microservice interactions.

Strengths:

- Enhances test efficiency through automation.
- Improves fault detection in complex microservice interactions.

Weaknesses:

- Limited evaluation scope, requiring further validation in different contexts.
- Dependency on the accuracy of the event-driven architecture analysis.

Metrics:

The paper does not specify particular performance metrics used to evaluate the quality and reliability of microservices tests. However, it highlights improvements in test efficiency and fault detection rates.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper introduces UTEMS, a tool and methodology for automated unit test generation in event-driven microservices.

40 Solving the Instance Identification Problem in Micro-service Testing

Publication Details:

- **Authors:** Theofanis Vassiliou-Gioles
- **Title:** Solving the Instance Identification Problem in Micro-service Testing
- **Publication Year:** 2022
- **Venue:** ICTSS 2021, LNCS 13045

Research Focus:

This paper focuses on enhancing the expressiveness of integration testing for microservices by introducing the concept of instance identification. It addresses the limitations in current integration testing methodologies, especially in managed microservices and functions-as-a-service (FaaS) environments, where lack of control and dynamicity impede effective testing.

Methodology:

The study proposes a methodology for identifying microservice instances using HTTP header fields. This involves the introduction of a microservice instance identification (IID) in request and response headers to improve testability and observability during integration testing. The methodology includes defining requirements for instance identification, structuring HTTP headers to include IID, and evaluating the approach's feasibility through practical implementation and lessons learned.

Tools and Frameworks:

None.

Findings:

The key finding is that instance identification significantly enhances the testability and observability of microservices during integration testing. By embedding IID in HTTP headers, testers can trace the interaction between microservices more accurately, leading to better identification of issues related to specific service instances. The approach helps reduce development round-trip times and improves the debugging process by providing clear insights into the service interactions and instance-specific information. However, the paper also highlights potential limitations, such as security concerns and header length constraints, which need to be addressed for broader adoption.

Metrics:

None.

Categorization:

- **Testing Techniques, Tools, and Frameworks:** The paper introduces a novel technique for enhancing integration testing in microservices through instance identification, making it a valuable contribution to the development of testing tools and frameworks.

41 Crunch: Automated Assessment of Microservice Architecture Assignments with Formative Feedback

Publication Details:

- **Authors:** Henrik Bærbak Christensen
- **Title:** Crunch: Automated Assessment of Microservice Architecture Assignments with Formative Feedback
- **Publication Year:** 2018
- **Venue:** European Conference on Software Architecture (ECSA)

Research Focus:

The paper focuses on the architectural design challenges and solutions related to the automated assessment of microservice architecture assignments with formative feedback in an educational context. It specifically addresses the development of the Crunch tool, which is used to assess student solutions involving microservice systems.

Methodology:

The research adopts a design and implementation methodology, where the architectural challenges of assessing microservice-based student assignments are analyzed, and solutions are proposed and implemented. The paper describes the course structure, the development of the Crunch tool, and the evaluation based on student and instructor feedback.

Tools and Frameworks:

Crunch is the main tool mentioned for the automated assessment of microservice architecture assignments. The tool utilizes Docker for containerization and deployment of student services, and JUnit for automated testing.

Findings:

The study finds that automated assessment using Crunch provides fast and continuous feedback to students, aligning with agile and DevOps practices. Key strengths include the ability to test distributed systems and non-functional requirements like availability and performance. However, challenges include the complexity of setting up the testing environment and ensuring the accuracy of the assessments. Student feedback indicated high satisfaction with the automated feedback provided by Crunch.

Metrics:

Metrics used to evaluate the quality and reliability of microservices tests include the correctness of the output, the time taken to complete requests, and the system's ability to handle failures gracefully.

Categorization:

- **Category:** Testing Techniques, Tools, and Frameworks
- **Reason:** The paper focuses on the development and use of the Crunch tool for automated testing and assessment of microservice-based systems.

42 Overview of Information System Testing Technology Under the "CLOUD + MIcroservices" Mode

Publication Details:

- **Authors:** Jianwei Zhang, Shan Jiang, Kunlong Wang, Rui Wang, Qi Liu, Xiaoguang Yuan
- **Title:** Overview of Information System Testing Technology Under the "CLOUD + MIcroservices" Mode
- **Publication Year:** 2022
- **Venue:** Conference on Computer and Communications Engineering (CCCE) 2022, published in Springer Lecture Notes in Computer Science

Research Focus:

The paper focuses on summarizing the testing technologies applicable to information systems operating under the combined cloud and microservices mode. It emphasizes the need for comprehensive testing frameworks that can address cloud platform functionalities, microservice applications, and the rapid construction of test environments.

Methodology:

The authors propose a testing framework that integrates various testing approaches for cloud platforms and microservice applications. They discuss component testing, end-to-end testing, regression testing, and system debugging. The methodology includes analyzing current testing technologies and their application to the proposed framework, as well as identifying gaps in existing research.

Tools and Frameworks:

The paper mentions several tools and frameworks used in cloud platform testing such as SPECCloud IaaS 2016 Benchmark, BUNGEE framework, and VMmark. For microservice testing, it references tools and methods like sandbox technology for parallel testing, machine learning for end-to-end testing, and automated regression testing combined with continuous delivery.

Findings:

The study identifies that current research on cloud and microservice testing is fragmented, often focusing on individual aspects rather than a holistic approach. It emphasizes the necessity of a comprehensive framework that includes both cloud and microservice testing, addressing challenges such as service dependencies, dynamic scaling, and fault tolerance. Strengths of existing methods include their specificity and effectiveness in targeted scenarios, while weaknesses are mainly related to the lack of integration and comprehensive coverage.

Metrics:

None.

Categorization:

- **Systematic Study Contributions:** The paper provides a broad review of existing technologies and proposes a systematic framework for testing information systems under the cloud and microservices mode. It aggregates knowledge from various studies to propose a comprehensive testing strategy.

43 Zero-Config Fuzzing for Microservices

Publication Details:

- **Authors:** Wei Wang, Andrei Benea, Franjo Ivančić
- **Title:** Zero-Config Fuzzing for Microservices
- **Publication Year:** 2023
- **Venue:** 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)

Research Focus: This paper introduces a novel zero-configuration fuzz testing technique for microservices to detect security vulnerabilities and reliability issues. It particularly targets Google's internal C++ microservices.

Methodology: The proposed system automatically generates fuzz tests by using structure-aware fuzzing of exposed APIs and mutating backend responses. The approach involves backend mocking, coverage-guided input generation, and fault injection—all integrated into Google's microservice platform with no developer involvement.

Tools and Frameworks: The system is implemented within Google's microservice infrastructure. It utilizes:

- gRPC
- Structure-aware fuzzing
- Internal fuzzing framework (name not disclosed)

Findings: The method was successfully deployed on over 95% of C++ microservices in Google. It led to the discovery and fixing of thousands of bugs. Backend fuzzing enhanced resilience by testing services in hermetic environments with simulated faults.

Metrics:

- Deployment coverage: >95% of internal C++ microservices
- Number of bugs reported: Thousands (exact figure not disclosed)
- Qualitative metric: Bug-finding effectiveness and increased fault resilience

Categorization:

- **Testing Techniques, Tools, and Frameworks:** This paper contributes a fully automated, zero-config fuzzing method integrated with gRPC-based microservice testing.

- **Security and Resilience:** It emphasizes robustness through systematic injection of unexpected inputs and backend errors.

44 A Discrete Dynamic Artificial Bee Colony with Hyper-Scout for RESTful Web Service API Test Suite Generation

Publication Details:

- **Authors:** Omur Sahin, Bahriye Akay
- **Title:** A Discrete Dynamic Artificial Bee Colony with Hyper-Scout for RESTful Web Service API Test Suite Generation
- **Publication Year:** 2021
- **Venue:** *Applied Soft Computing Journal*, Volume 104, Article 107246

Research Focus:

This study proposes an enhanced evolutionary algorithm for generating test suites for RESTful APIs, addressing the limitations of previous algorithms in terms of exploration and adaptability to diverse testing targets.

Methodology:

The proposed approach builds on the Artificial Bee Colony (ABC) algorithm, enhanced with:

- A Hyper-Scout unit for better exploration
- A multi-objective optimization strategy considering code coverage, number of test cases, and server error responses
- A dominance-based selection to prioritize relevant and effective test cases

The method includes dynamic adjustment of exploration and exploitation phases and introduces an archive mechanism to store and reuse high-quality test cases.

Tools and Frameworks:

- Custom implementation of ABC and Dynamic ABC with Hyper-Scout algorithms
- Compared against WTS, MOSA, MIO, and RANDOM algorithms on seven RESTful APIs

Findings:

- The DABC-HS outperformed other algorithms in 4 out of 7 test scenarios
- Demonstrated strong coverage and efficiency across diverse RESTful services
- Achieved better balance in the generation of test cases by considering multiple test quality dimensions simultaneously

Metrics:

- Coverage amount
- Number of server errors identified
- Number of test cases generated
- Execution time
- Statistical tests were used to compare performance across algorithms

Categorization:

- Test Data/Case Generation: The paper focuses on automated test suite generation using advanced evolutionary algorithms tailored for RESTful APIs.

45 NxtUnit: Automated Unit Test Generation for Go

Publication Details:

- **Authors:** Siwei Wang, Xue Mao, Ziguang Cao, Yujun Gao, Qucheng Shen, Chao Peng
- **Title:** NxtUnit: Automated Unit Test Generation for Go
- **Publication Year:** 2023
- **Venue:** *International Conference on Evaluation and Assessment in Software Engineering (EASE 2023)*, Oulu, Finland

Research Focus:

This paper introduces NxtUnit, a lightweight and automated unit test generation tool for the Go programming language, specifically tailored for microservice development environments.

Methodology:

NxtUnit uses a random testing approach to generate unit tests automatically. It simulates downstream call outputs and records assertions as ground truths for regression testing. The tool supports three interfaces:

1. IDE plugin
2. CLI tool
3. Browser-based platform for asynchronous test generation and visualization

Tests are generated by mutating inputs and mocking downstream outputs to produce high-coverage scenarios quickly and without human intervention.

Tools and Frameworks:

- NxtUnit (open-sourced on GitHub: https://github.com/bytedance/nxt_unit)
- Evaluated on:
    - 13 open-source Go repositories
    - 500 in-house ByteDance repositories

Findings:

- Achieved an average code coverage of 20.74% on in-house projects
- Internal developer survey showed 48% time savings on unit test writing
- Provided quick smoke testing support compatible with microservices architecture

Metrics:

- Code coverage percentage
- Developer-reported time savings
- Survey results assessing usability and adoption

Categorization:

- **Test Data/Case Generation**: The tool generates unit-level test cases automatically.

46 White-Box Fuzzing RPC-Based APIs with EvoMaster: An Industrial Case Study

Publication Details:

- **Authors:** Man Zhang, Andrea Arcuri, Yonggang Li, Yang Liu, Kaiming Xue
- **Title:** White-Box Fuzzing RPC-Based APIs with EvoMaster: An Industrial Case Study
- **Publication Year:** 2023
- **Venue:** *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 32, No. 5, Article 122

Research Focus:

This paper addresses the challenge of automated testing for modern RPC-based microservices. It proposes the first white-box fuzzing approach specifically for RPC APIs, extending the EvoMaster fuzzer to support industrial-scale testing of services using frameworks like Thrift, gRPC, SOFARPC, and Dubbo.

Methodology:

formulate RPC API specifications. The extracted schemas are used in a search-based testing process guided by white-box heuristics and RPC-specific strategies.

The tool reformulates API test cases as sequences of RPC calls and applies the MIO algorithm to evolve high-coverage test scenarios. The system is capable of injecting faults and analyzing behaviors in real-world distributed systems.The authors developed a schema-extraction technique from JVM-based source code to

Tools and Frameworks:

- EvoMaster (extended for RPC fuzzing)
- RPC frameworks: Thrift, gRPC, SOFARPC, Dubbo
- Instrumentation with JaCoCo for code coverage analysis

Findings:

- Successfully tested 54 industrial RPC-based APIs (1.49M+ LOC)
- Found 41 real bugs confirmed by industrial partners

- Discovered 8,377 additional issues under investigation
- Outperformed gray-box techniques and manual tests in coverage and bug detection

Metrics:

- Code coverage
- Number of real bugs found
- Number of faults under investigation
- Comparison against baseline techniques (gray-box, manual)

Categorization:

- **Testing Techniques, Tools, and Frameworks:** Focused on RPC-specific white-box fuzzing

47 RESTful API Automated Test Case Generation

Publication Details:

- **Author:** Andrea Arcuri
- **Title:** RESTful API Automated Test Case Generation
- **Publication Year:** 2019

Research Focus:

The paper proposes a fully automated white-box testing approach for RESTful APIs, leveraging evolutionary algorithms to maximize code coverage and detect faults. It addresses the lack of developer-accessible testing tools capable of exploiting internal knowledge of the service under test.

Methodology:

- The approach is implemented in EVOMASTER, an open-source test generation tool.
- It uses a Genetic Algorithm (GA) following the Whole Test Suite (WTS) approach.
- Test cases are scored using coverage-based fitness and HTTP status code-based fault oracles.
- The testing focuses on integration-level testing of services in isolation from external dependencies.

Tools and Frameworks:

- EVOMASTER (open source: https://github.com/arcuri82/EvoMaster)
- Uses white-box instrumentation and coverage analysis tools (e.g., JaCoCo)
- Targets Java-based RESTful services

Findings:

- Successfully tested three RESTful services (2 open-source, 1 industrial)
- Automatically discovered 38 real bugs
- While fault detection was strong, code coverage was lower compared to manually written test suites, due to issues like string constraints and external dependencies

Metrics:

- Statement code coverage
- Number of real bugs found (38)
- Qualitative analysis of coverage gaps due to technical constraints

Categorization:

- **Test Data/Case Generation:** Focuses on automatically generating REST API integration tests

48 Gremlin: Systematic Resilience Testing of Microservices

Publication Details:

- **Authors:** Victor Heorhiadi, Shriram Rajagopalan, Hani Jamjoom, Michael K. Reiter, Vyas Sekar
- **Title:** Gremlin: Systematic Resilience Testing of Microservices
- **Publication Year:** 2016
- **Venue:** *36th IEEE International Conference on Distributed Computing Systems (ICDCS 2016)*

Research Focus:

The paper introduces Gremlin, a resilience testing framework that systematically evaluates the failure-handling logic of microservice-based applications by simulating realistic network and service failures in production-like environments.

Methodology:

- Recipe-based Testing: Users define failure scenarios and expected behaviors as "recipes" written in Python.
- Control and Data Plane Separation: Inspired by SDN, Gremlin's control plane manages fault injection logic, while the data plane injects faults by intercepting and manipulating inter-service network traffic.
- Message-level Fault Injection: By exploiting standard communication protocols (e.g., HTTP), Gremlin simulates failures without modifying the microservice code.
- Assertions: Developers define expected outcomes to validate system behavior post-failure.

Tools and Frameworks:

- Gremlin framework (open-source: https://github.com/ResilienceTesting)
- Built-in support for HTTP-based microservices and integration with real network proxies

Findings:

- Enabled developers at IBM to identify previously undetected failure-handling bugs in production applications
- Detected missing recovery logic in an actively used resilience library
- Designed for minimal intrusion and rapid feedback during live deployments

Metrics:

- Detection of real-world bugs (qualitative)
- Recipe execution latency (designed to complete in seconds)
- Minimal runtime overhead in production-like environments

Categorization:

- **Testing Techniques, Tools, and Frameworks:** Introduces a novel tool for runtime fault simulation and behavior verification