Non-functional Regression Testing on Serverless Applications Using Controlled Online Experiments

Simon Frey
B.Sc. Computer Science

Supervisor: Prof. Dr.-Ing. Stefan Tai
Secondary Supervisor: TBD
Advisor: Jörn Kuhlenkamp

Technische Universität Berlin

March 4, 2020

1 Motivation

The serverless computing [10] or function as a service (FaaS) execution model gains more traction[1]. This growth is facilitated by two differences compared to the infrastructure as a service (IaaS) model, the first being the Fully-managed elastic scalability where "[...] the [cloud] provider dynamically adapts resource allocations according to the customer's demand [workload] and the customer pays for the actually consumed resources [...]"[16, p.1]. This allows to focus more resources on business logic than DevOps[9]. Additionally FaaS offers a potential cost benefit[2] empowered by the fine-granular pay-per-execution cost model[2]. Because of this model FaaS does not have cost at rest. Thus, volatile workloads are a scenario where FaaS is most beneficial.

Both characteristics are most beneficial if the application code has a good quality. In order to ensure a high code quality, the software needs to be tested on a regular bases. One test schema is regression testing which "[...] is to ensure that changes made to software, such as adding new features or modifying existing features, have not adversely affected features of the software that should not change. [...]"[17, p.1]. Regression testing is of interest for web applications as they "[...] must undergo rapid adjustments, since the businesses they support are frequently changing [...]"[11, p.1]. The e-commerce website layed out in the evaluation section is such a web application profiting off regression testing. Regression testing consists of functional and non-functional test cases. Functional cases test the actual behavior against the specification and non-functional cases test "[...] the way a system operates, rather than specific behaviors of that system [...]"[3]. Non-functional regression testing is an important mechanism to ensure good application behavior. Manual regression testing is resource intensive as time consuming processes are often repeated. Automatic regression testing solves this resource problem by automating the test execution and evaluation. These tests are also referred to as Experiments. Automatic regression testing can be implemented in two different ways: Offline or Online[6]. Offline tests run within a dedicated experimentation infrastructure isolated from the live application. This brings two problems: The Infrastructure overhead increases with the complexity of the application as all resources need to be copied and continuously updated to stay in sync with the live application. The other problem is the difficulty to ensure a good workload quality as the workload is artificial and can differ a lot from the actual live application. Online tests try to tackle these problems, by being executed in the live application. Thereby the workload quality is high because it is the live workload. The

[1] https://cacm.acm.org/magazines/2019/12/241054-the-rise-of-serverless-computing/fulltext
[2] https://aws.amazon.com/lambda/pricing/
[3] https://en.wikipedia.org/wiki/Non-functional_testing

infrastructure overhead is smaller as no isolated environment needs to be created and kept up to date. These benefits come with the downside of user facing side effects, which depend on the chosen deployment method. With the usage of the live workload it is difficult to run certain experiments relying on certain traffic patterns. The different deployment methods and the problems with traffic depended experiments are layed out in the Problem section.

With the rise of FaaS and its usage in more complex applications non-functional automatic regression testing is required to ensure the same application quality for FaaS as for conventional execution models. The distributed nature of serverless applications increases the difficulty to keep a offline testing environment up to date as of why online experiments are beneficial. Online experiments on serverless applications are not supported out of the box by the cloud providers and research does not yet cover this topic.

In conclusion non-functional regression testing on serverless applications using controlled online experiments is relevant but yet unexplored.

2 Problem

Online experiments require the integration of an experiment setup into the production environment. There are three different deployment methods known from continuous deployment[12] (CD) which are also utilized for this integration. The first is *Percentage based routing* a group combining different strategies found in the literature(Canary deployment[5], A/B deployment[7] & Gradual deployment[5]). All are based on the same technic: Routing a certain percentage of the workload trough the experiment. Sometimes the terms are used interchangeable as there is no clear cut. The second one also depending on traffic routing is *Shadow/Dark deployment*[12]. It works with duplicating a certain amount of traffic to the experiment. Compared to the percentage based deployment a dedicated second deployment of the function is utilized which results are not returned into the live application. The last one, not depending on traffic routing, is called *Feature toggles*[1]. Utilizing special code in the live codebase the activation is done within the application logic.

Depending on which deployment method is used the amount of user-facing side effects differ. All deployment methods work on the live workload resulting in a realistic workload, but the ability to use specific workloads is low, as only workload pattern occurring during the time of the experiment can be tested. This is a problem for capacity experiments[5, p. 76]. This kind of experiments test the application behavior regarding special workload scenarios. E.g. The e-commerce website used for evaluation may have a lot more traffic during the christmas time than the rest of the year. In order to be able to ensure the application can handle that load capacity experiments are required but difficult to implement in an online experiment setup.
This problem result in the following research question:

How can developers use specialized workloads in online experiments on serverless applications on demand?

To tackle that question the thesis *proposes a system to shape the experiment traffic* in order to run online capacity experiments and *evaluates the system quality* within the e-commerce case study application.

3 Related Work

In their Bifrost paper Schermann et.al.[14] lay out the architecture of a tool enabling percentage based routing & shadow deployment as deployment methods for experiments in container based application. Their insights are used to explore comparison criteria and base experiment designs on.

Ernst et.al. propose the usage of ephemeral proxy in their canary paper[3]. It is to be evaluated if this approach is superior to the permanent proxy used within Bifrost for percentage based routing & shadow deployment.

Apart form Bifrost Scherman et.al. have published other relevant papers regarding the thesis covering research on continuous experimentation[15],[13].

Online experiments rely on zero downtime deployment and the research in this field[5][4][3] helps as base for the system design layed out in the next section.

The percentage based release strategy is already implemented by AWS with AWS Codeploy[4]. It is to be evaluated if this implementation is usable for controlled online experiments.

4 Approach

The thesis focuses on the biggest (by market share[5]) provider of cloud infrastructure Amazon Web Services (AWS). The overhead of supporting multiple cloud providers is out of scope. To further narrow the scope only AWS API Gateway[6] events as triggers are evaluated. By its usage of the HTTP protocol it allows to use standard tools like JMeter[7] for reproducable evaluation-traffic generation.

The experimentation control is done via a proxy function shaping the traffic, e.g. duplicating events to simulate a higher traffic scenario. The experiment may run on the live function or on a test function, depending on the use-case for the developer. The proxy injects a marker into the events in order to enable the functions to recognize experiment traffic and prevent side effects. Side effects are prevented within the experiment function and are thereby out of scope of the system proposed in the thesis.
For a single experiment deployed with the strategy to 1:1 duplicate the traffic this would look like following:

[4]https://aws.amazon.com/de/blogs/compute/implementing-safe-aws-lambda-deployments-with-aws-codedeploy/

[5]https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

[6]https://aws.amazon.com/api-gateway/

[7]https://jmeter.apache.org/

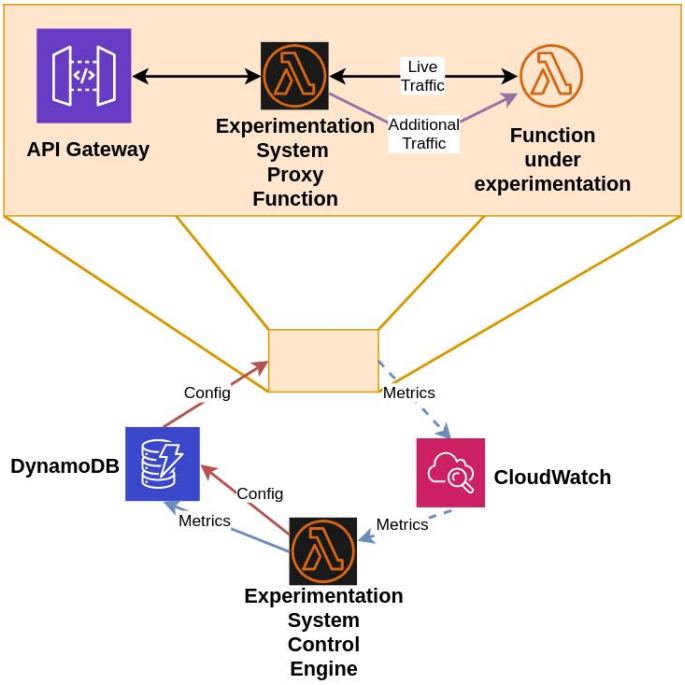

Figure 1: Experiment duplicating traffic 1:1

The control engine deploys and removes the proxy with the AWS SDK. The proxy is configured trough an entry in DynamoDB[8] which it checks on startup and in regular intervals. Metrics are publish to the control engine via AWS CloudWatch[9]. The control engine aggregates and persists the metrics into DynamoDB.

5 Evaluation

The case study application from Bifrost[10] will be used as base application as "Unfortunately, few suitable open source microservice-based applications exist [...] [and that] application simulates a generic e-commerce website selling consumer electronics. It was kept simple in order to provide a testbed for the performance evaluation" [14, p.7]. As mentioned in the motivation web applications benefit from non-functional regression testing because of their rapid changing nature which makes the application a valid use-case for online experiments. The application is adapted to fit into the AWS Lambda infrastructure.

Following experiments are implemented with the layed out system on the case study application:

- *Elasticity experiment* as introduced by Kuhlenkamp et.al. in their article [8] testing the ability of the application to adapt to changing workloads
- *Edge-case workload injection experiment* injecting workloads that where problematic for the application in the past in order to test if new application versions improved for that workloads

The system is evaluated against the following criteria:

The most important criteria are the *Client-visible Side-effects (CvSEs)* which should be minimized during the execution of the experiments to keep the live application functional. If the CvSEs are to high a rollback of the experiment might be required to keep the application available. The time from recognizing the problem until the rollback is finished is called *Mean time to repair (MttR)*.

[8] https://aws.amazon.com/dynamodb/
[9] https://aws.amazon.com/cloudwatch/
[10] https://github.com/sealuzh/bifrost-microservices-sample-application

Different methods offer different *Workload accuracy (WA)*. If that accuracy is to low the experiments will generate wrong data which results in wrong decisions. On the other hand there is *Workload Expressiveness (WE)*, which is the ability to express different workloads. If this is low, only very few possible workload scenarios can be tested.

In case the deployment consumes lot of time the *Time to experiment start (TtES)* increases and a continuous execution of the experiments might not be feasible. This characteristic involves the time it takes a developer to add an experiment to the experimentation application. This time increases with more complex applications involved. Apart from this time overhead the other overhead criteria is the *Infrastructure cost overhead (ICO)*. It depends on the amount of additional infrastructure required. Too high costs could prevent continuous execution of the experiments as of economical reasons.

The softest criteria is *Usability (U)* of the experimentation system. Though not influencing the actually experiment execution it is still important because a difficult system might not be used widely enough to bring value.

References

[1] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect's Perspective*. SEI Series in Software Engineering. New York: Addison-Wesley, 2015. ISBN: 978-0-13-404984-7. URL: http://my.safaribooksonline.com/9780134049847.

[2] Adam Eivy. "Be Wary of the Economics of "Serverless" Cloud Computing". In: *IEEE Cloud Computing* 4.2 (Mar. 2017), pp. 6–12. ISSN: 2372-2568. DOI: 10.1109/MCC.2017.32.

[3] Dominik Ernst, Alexander Becker, and Stefan Tai. "Rapid Canary Assessment Through Proxying and Two-Stage Load Balancing". In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2019, pp. 116–122. DOI: 10.1109/ICSA-C.2019.00028.

[4] Dror Feitelson, Eitan Frachtenberg, and Kent Beck. "Development and Deployment at Facebook". In: *IEEE Internet Computing* 17.4 (July 2013), pp. 8–17. ISSN: 1941-0131. DOI: 10.1109/MIC.2013.25.

[5] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 1st. Addison-Wesley Signature Series (Fowler). Addison-Wesley Professional, 2010. ISBN: 9780321670229. URL: https://books.google.de/books?id=6ADDuzere-YC.

[6] Richard Karp. "On-Line Algorithms Versus Off-Line Algorithms: How Much is it Worth to Know the Future?" In: July 1992. URL: http://www.icsi.berkeley.edu/pubs/techreports/TR-92-044.pdf.

[7] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. "Practical Guide to Controlled Experiments on the Web: Listen to Your Customers Not to the Hippo". In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '07. San Jose, California, USA: Association for Computing Machinery, 2007, pp. 959–967. ISBN: 9781595936097. DOI: 10.1145/1281192.1281295. URL: https://doi.org/10.1145/1281192.1281295.

[8] Jörn Kuhlenkamp et al. "Benchmarking Elasticity of FaaS Platforms as a Foundation for Objective-driven Design of Serverless Applications". In: *35th ACM/SIGAPP Symposiumon Applied Computing (SAC '20)*. Apr. 2020. DOI: 10.1145/3341105.3373948.

[9] Zhu Liming, Bass Len, and Champlin-Scharff Georg. "DevOps and Its Practices". In: *IEEE Software* 33.3 (May 2016), pp. 32–34. ISSN: 1937-4194. DOI: 10.1109/MS.2016.81.

[10] Mike Roberts. *Serverless Architectures*. https://martinfowler.com/articles/serverless.html. [Online; accessed 27.02.2019]. May 2017.

[11] Michael Ruth and Shengru Tu. "Towards Automatic Regression Test Selection for Web Services". In: *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*. Vol. 2. July 2007, pp. 729–736. DOI: 10.1109/COMPSAC.2007.219.

[12] Tony Savor et al. "Continuous Deployment at Facebook and OANDA". In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. May 2016, pp. 21–30.

[13] Gerald Schermann, Jürgen Cito, and Philipp Leitner. "Continuous Experimentation: Challenges, Implementation Techniques, and Current Research". In: *IEEE Software* 35.2 (Mar. 2018), pp. 26–31. ISSN: 1937-4194. DOI: 10.1109/MS.2018.111094748.

[14] Gerald Schermann et al. "Bifrost: Supporting Continuous Deployment with Automated Enactment of Multi-Phase Live Testing Strategies". In: *Proceedings of the 17th International Middleware Conference*. Middleware '16. Trento, Italy: Association for Computing Machinery, 2016. ISBN: 9781450343008. DOI: 10.1145/2988336.2988348. URL: https://doi.org/10.1145/2988336.2988348.

[15] Gerald Schermann et al. "We're doing it live: A multi-method empirical study on continuous experimentation". In: *Information and Software Technology* 99 (2018), pp. 41–57. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.02.010. URL: http://www.sciencedirect.com/science/article/pii/S0950584917302136.

[16] Andreas Weber et al. “Towards a Resource Elasticity Benchmark for Cloud Environments”. In: *Proceedings of the 2nd International Workshop on Hot Topics in Cloud Service Scalability*. HotTopiCS ’14. Dublin, Ireland: Association for Computing Machinery, 2014. ISBN: 9781450330596. DOI: 10.1145/2649563.2649571. URL: https://doi.org/10.1145/2649563.2649571.

[17] W. Eric Wong et al. “A study of effective regression testing in practice”. In: *Proceedings The Eighth International Symposium on Software Reliability Engineering*. Nov. 1997, pp. 264–274. DOI: 10.1109/ISSRE.1997.630875.