Abstract

BACKGROUND: The number of users of web services is increasing annually and, therefore, all developers aim to develop a high reliable software. However, no system can withstand any load, so site reliability engineers prepare for potential service failures by implementing various reliability practices to minimize associated risks. Feedback control system is a system that can give feedback about its current status so that engineers can react to it in advance the risks of a fall.

OBJECTIVE: However, for feedback control systems developers face a problem that each reliability practice is configured in its own way and combinations of these settings can give different results in the reliability of the system. This paper outlines a web-service for reproducible load testing, visualization, and configuration of feedback control systems

METHODS: The web-services was developed by using Kotlin and Spring Boot framework, and Kafka broker message was used for communication. Yandex tank was chosen as a high load generator for the ability to use several cores for load generation and a convenient API. Additionally, by using Angular framework I developed a UI interface for a more convenient and visual use of the service.

RESULTS: The proposed service can handled services for configuration, executed load testing scenarios and provided real-time results as graphs.

CONTRIBUTION AND APPLICABILITY: This service can be used in testing systems under various load scenarios with different configuration variants. These practices help to find the most effective combination of parameters that ensures optimal system performance.

Chapter 1

Introduction

Chapter 2

Literature Review

To further implementation of the service it is required to choose appropriate tools and approaches for development. This chapter is organized in such way:

- Section 1 reviews modern load generation tools
- Section 2 gives a brief overview of reliability patterns configuration
- Section 3 outlines the tools and technologies for implementing service

I Load Generation Tool

The main criteria of load generation tool choice are ability to generate more than 100000 requests per seconds and to execute load test by code. [1] present a comprehensive comparisons of the modern load generation tools. They use different criteria, such as different protocol supporting, cost and easy to learn. Similar comparison proposed by [2], but, additionally, the authors compare tools by efficiency. Based on comparisons the most usable tool is

Jmeter [2] because of multiple scripting language, friendly graphical user interface and open source. Additionally, all authors pointed out the Gatling tool due to its flexible load generation scenarios. But I chose to use Yandex.Tank because all studies do not consider the fact that Yandex tank, unlike Jmeter or Gatling, can use different engines to generate loads, such as Phantom, BFG, Pandora [yandex tank doc]. For example, with the Phantom engine [2], it can generate a load of more than 100,000 requests per second, or it can use Jmeter engine and use all its features. This tool is open-source and can be configured using a config file. Additionally, Yandex Tank has a ready-made web server [3], which can execute load test by HTTP request. It will be taken as the basis for the load generation microservice.

II Reliability patterns

Reliability patterns ensure the stability and resilience of high-load systems. But each pattern have its own tune parameters and in case of multiple patterns implementation it is challenging to handle all configurations.

In order to properly withstand with high load [4] examines corresponding strategies. They present several approaches, such as rate-limit, retry strategy, circuit-breaker and load-balancer. The authors offers many ideas how to properly implement them. Similar ideas are presented in [12] about circuit-breaker and retry patterns. However, the authors do not highlight best practices for configuring these approaches. For example, which parameters are better to set in certain load scenarios.

Circuit-breaker have several tune parameters [4], such as sleep window, error percentage threshold, etc. At the same time, load balancer can have multiple strategies, such as round-robin, ratio, etc. Retry strategy can be configured by maximum number and interval of retries [11]. The parameters of these patterns produce different throughput in different combination, and it is crucial to understand how to find the most performable.

III Orchestrator services

Orchestrator services is needed for automated configuring, coordinating and managing software. The examples of orchestrator services are Consul, Kubernetes, etc. Consul is configuration management system that provides service discovery and health checking [15]. At the same time Kubernetes can orchestrate containers, handle internal networks, etc. However, none of these services can configure concrete reliability patterns. They are not able to run load test and find the most performable configuration.

IV Service implementation

First, Spring framework is the main choice of the web services implementation. [8] explains that in comparison to Django, Rails, Spring has high scalability and high-quality documentation. Additionally, the authors points out that Spring support multiple language, such as Java, Kotlin and Groovy.

Second, service is divided into microservices due to machine workload during load test [9]. It can affect the perfomance of configuration orchestration logic. In microservice architecture the configuration service will be stable in case of load generation system throttling

Third, service based on event-driven architecture [6] because of the constant configuration changes and the need for real-time awareness. According to this architecture, it is required to use stream of event and the authors suggests to use message broker for its implementation. Kafka tool is used to implement it due to high throughput and fault tolerance [7].

Fourth, the service infrastructure is constructed by combination of Docker and Kubernetes. Docker is used for containerization that allows easier deployment and scalability [13]. Kubernetes is used for orchestrating and managing these containers. It provides reliablity features, such as self-healing, automatic scaling, etc. [14]