# Evaluation of Microservice Architecture Designs in an IoT-Context

Abramov Sviatoslav
svyatabram@gmail.com

Supervisor: Prof. Dr.-Ing Günter Schäfer

Supervisor: Associate Prof. Dr. Igor Anikin

# Content

- Introduction
- Requirements
- State of the Art
- The system overview
- Results
- Conclusion
- Future Work

# Introduction

# The goals

The main goal is to find the best MSA architecture design in an IoT context.

We compared:

► Interconnection methods

► Database management systems

► Load balancing strategies

# What we have done

1. Analyzed the most demanded technologies in MSA context.
2. Built the MSA application, satisfying the most common requirements.
3. Developed a load generation, simulating IoT devices.
4. Implemented a monitoring system
5. Processed measurements.

# Requirements

# Functional requirements

1. Provide connectivity for IoT devices.
2. Transform IoT device data model to the system data model.

# Non-functional Requirements

## Qualitative

- Testable
- Reproducible
- Deployable

## Quantitative

- Response time
- Scalable

# State of the art

# Basic articles

M. S. Hatem Hamad and R. Abed, “Performance evaluation of restful web services for mobile devices,” *Computer Engineering Department, Islamic University of Gaza, Palestine, International Arab Journal of e-Technology*, 2010.

In the article advantages of RESTful web services before SOAP web services are shown:

- RESTful web services provide less message size.
- RESTful web services provide less response time.

# Basic articles

P. J. Amaral M. and C. D., “Performance evaluation of microservices architectures using containers.,” *IEEE 14th International Symposium on Network Computing and Applications*, 2015.

In this article Server Virtualization provides performance improvement is shown.

- SV increases server throughput
- SV decreases server latency

# Basic articles

- J. F. Kunhua Zhu and Y. Li, *Research the performance testing and performance improvement strategy in web application", 2nd international Conference on Education Technology and Computer.* 2010.

This article provides a survey about overall MSA based application testing :

- Functional and load testing of web MSA based web applications basis.
- Prediction of an application response time changing with an increasing user load.
- User load impact on application throughput.
- Possible bottlenecks caused by system utilization.

# Not answered questions

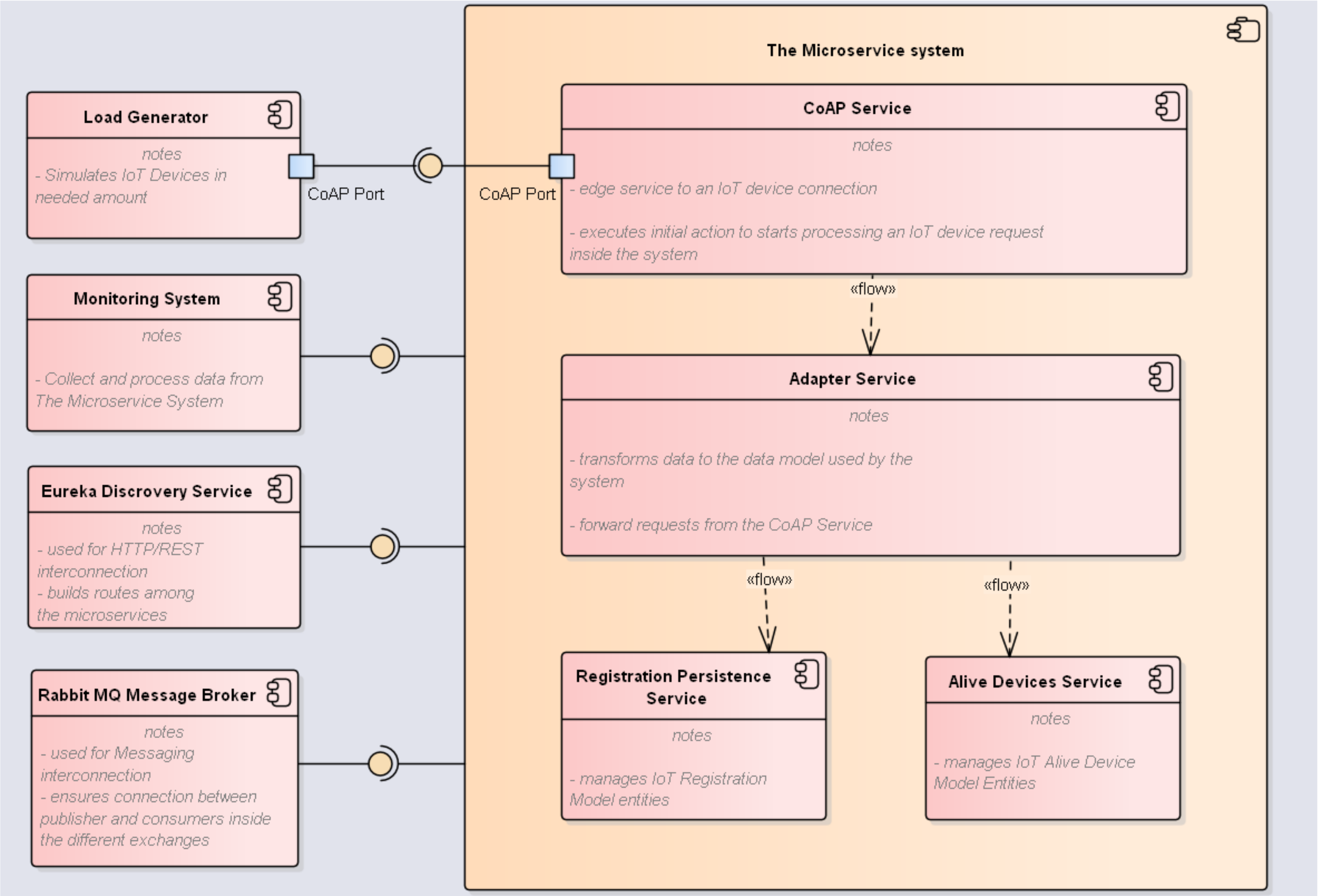
- We did not find in that surveys which design chooses we have to made to build MSA system in an IoT-device context.
- Which interconnection method fits a certain service functionality better?
- How to use load balancing according to services functionality?
- …?

# The system overview

# The system developing

- The MS system has to provide connectivity, registration, data model transformation.
- Assessment criteria is response time -> we have to measure response time of each service and of the whole MS system.
- We used the same hard/software version.
- Docker images?
- Load generator is able to generate different load?
- We tried to write more performable code?

# The system developing



The Microservice system

Load Generator
notes
- Simulates IoT Devices in needed amount

CoAP Port

CoAP Port

CoAP Service
notes
- edge service to an IoT device connection
- executes initial action to starts processing an IoT device request inside the system

«flow»

Monitoring System
notes
- Collect and process data from The Microservice System

Adapter Service
notes
- transforms data to the data model used by the system
- forward requests from the CoAP Service

Eureka Discrovery Service
notes
- used for HTTP/REST interconnection
- builds routes among the microservices

«flow»

«flow»

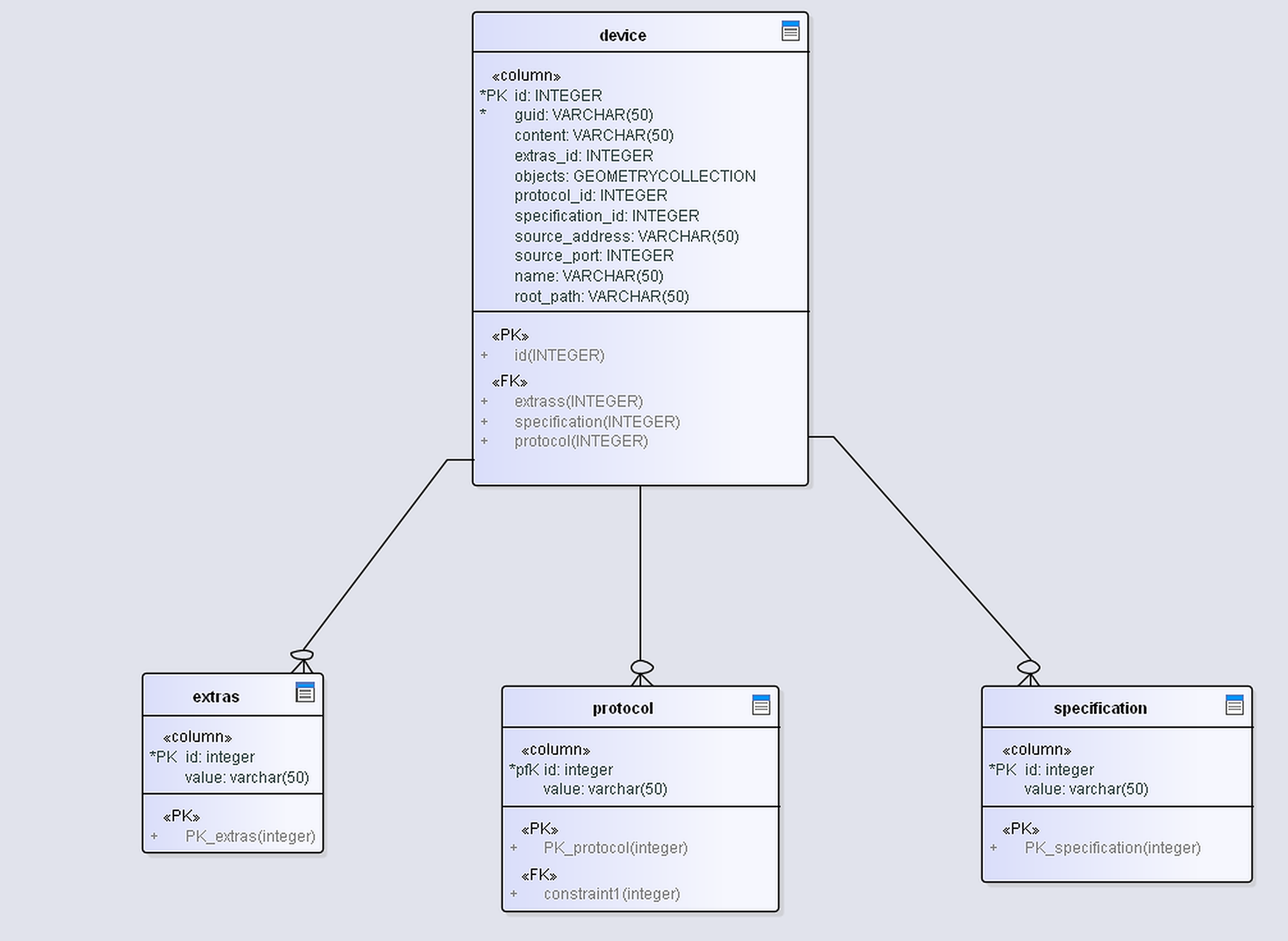
Rabbit MQ Message Broker
notes
- used for Messaging interconnection
- ensures connection between publisher and consumers inside the different exchanges

Registration Persistence Service
notes
- manages IoT Registration Model entities

Alive Devices Service
notes
- manages IoT Alive Device Model Entities

The system components diagram.

# Registration data model



**device**

«column»
*PK id: INTEGER
* guid: VARCHAR(50)
content: VARCHAR(50)
extras_id: INTEGER
objects: GEOMETRYCOLLECTION
protocol_id: INTEGER
specification_id: INTEGER
source_address: VARCHAR(50)
source_port: INTEGER
name: VARCHAR(50)
root_path: VARCHAR(50)

«PK»
+ id(INTEGER)
«FK»
+ extrass(INTEGER)
+ specification(INTEGER)
+ protocol(INTEGER)

**extras**

«column»
*PK id: integer
value: varchar(50)

«PK»
+ PK_extras(integer)

**protocol**

«column»
*pfK id: integer
value: varchar(50)

«PK»
+ PK_protocol(integer)
«FK»
+ constraint1(integer)

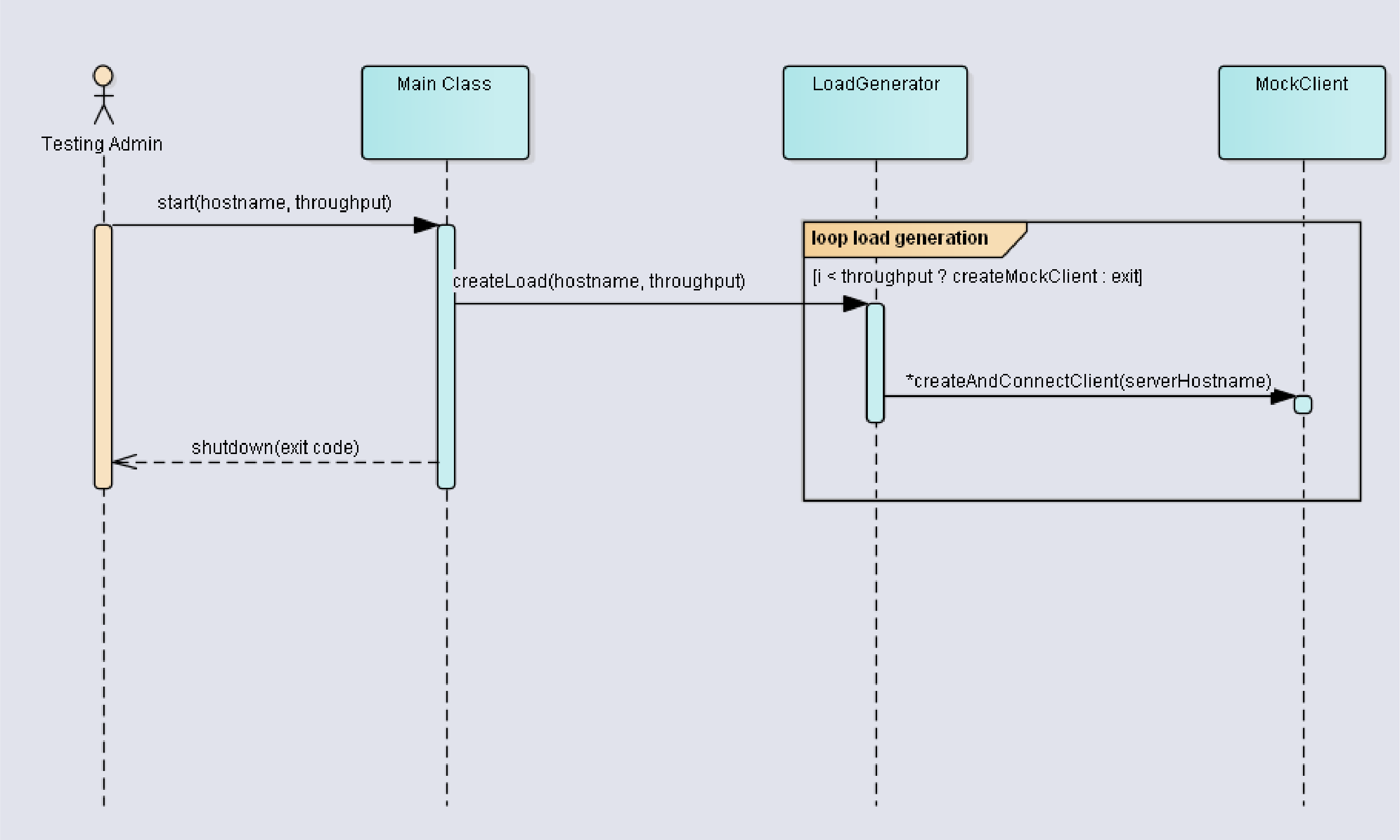
**specification**

«column»
*PK id: integer
value: varchar(50)

«PK»
+ PK_specification(integer)

# Load generation

- Simulates a given amount of IoT devices.

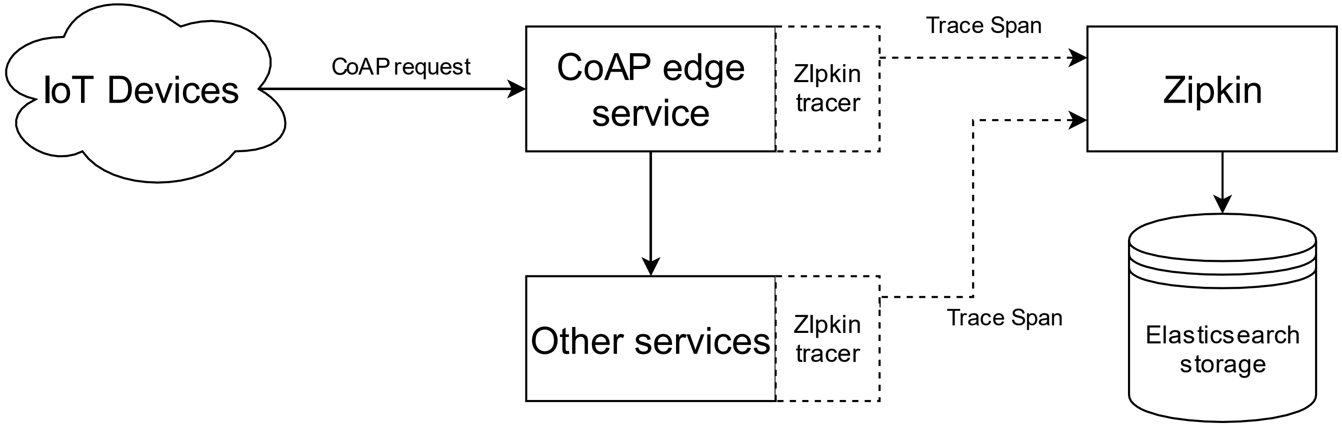

Testing Admin
Main Class
LoadGenerator
MockClient
start(hostname, throughput)
createLoad(hostname, throughput)
loop load generation
[i < throughput ? createMockClient : exit]
*createAndConnectClient(serverHostname)
shutdown(exit code)

Load generation sequence diagram.

# Measuring system

- To measure response time we used *distributed tracing* system.
- We stored trace logs in *document-oriented* DB.
- We used search engine to retrieve and analyze logs

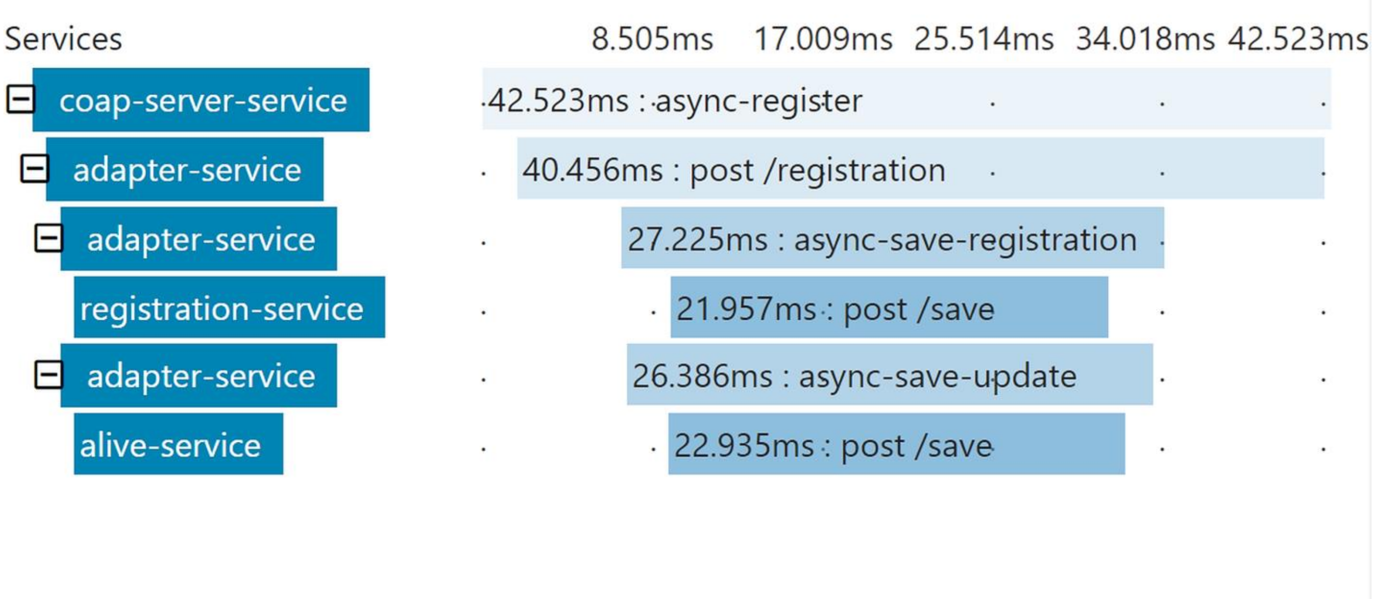
# Measuring system



Measuring system diagram.

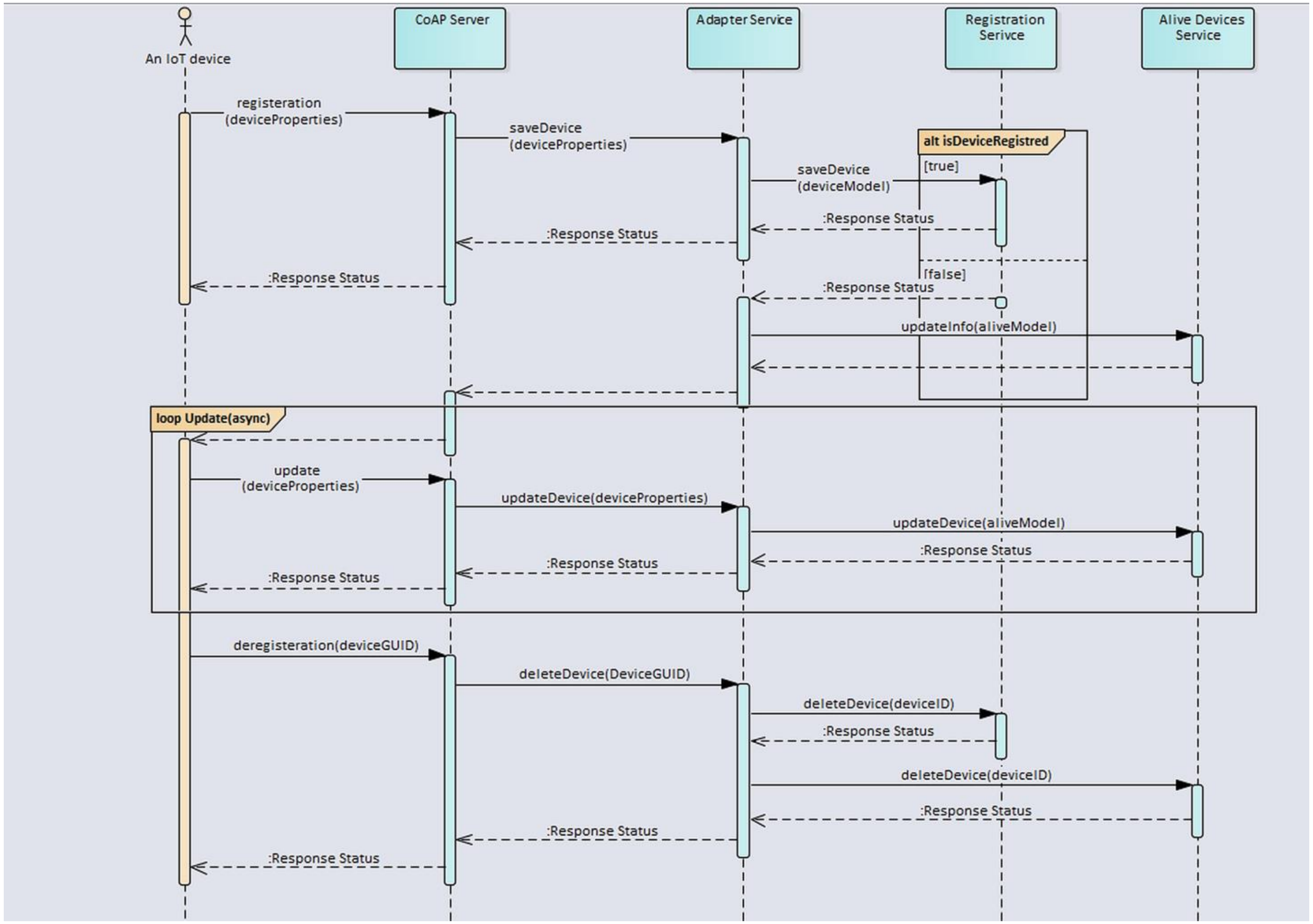
- Each service uses Zipkin API to send tracing spans to Zipkin server.
- All spans persist in ES database.
- Possibility to measure each service response time.
- Services traceroute.

# Measuring system



Request timeline diagram example.

# Test scenario



Test sequence diagram.

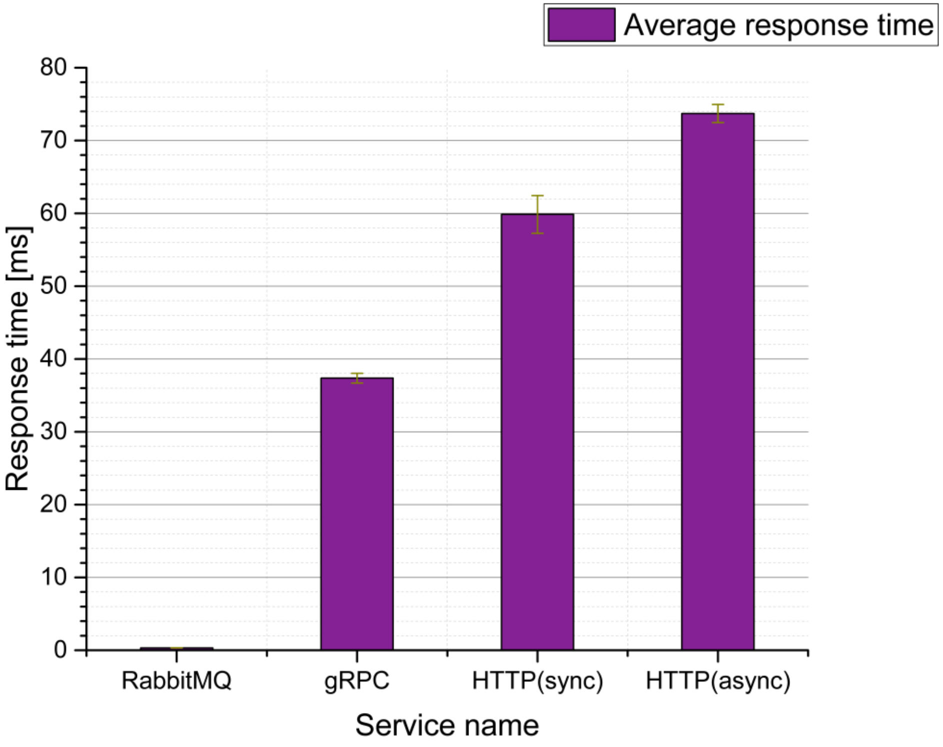
# Results

# Interconnection comparison

- RabbitMQ provides the lowest response time of less than 1 millisecond.

- Async. HTTP provides the highest response time,  a bit higher 70 milliseconds.

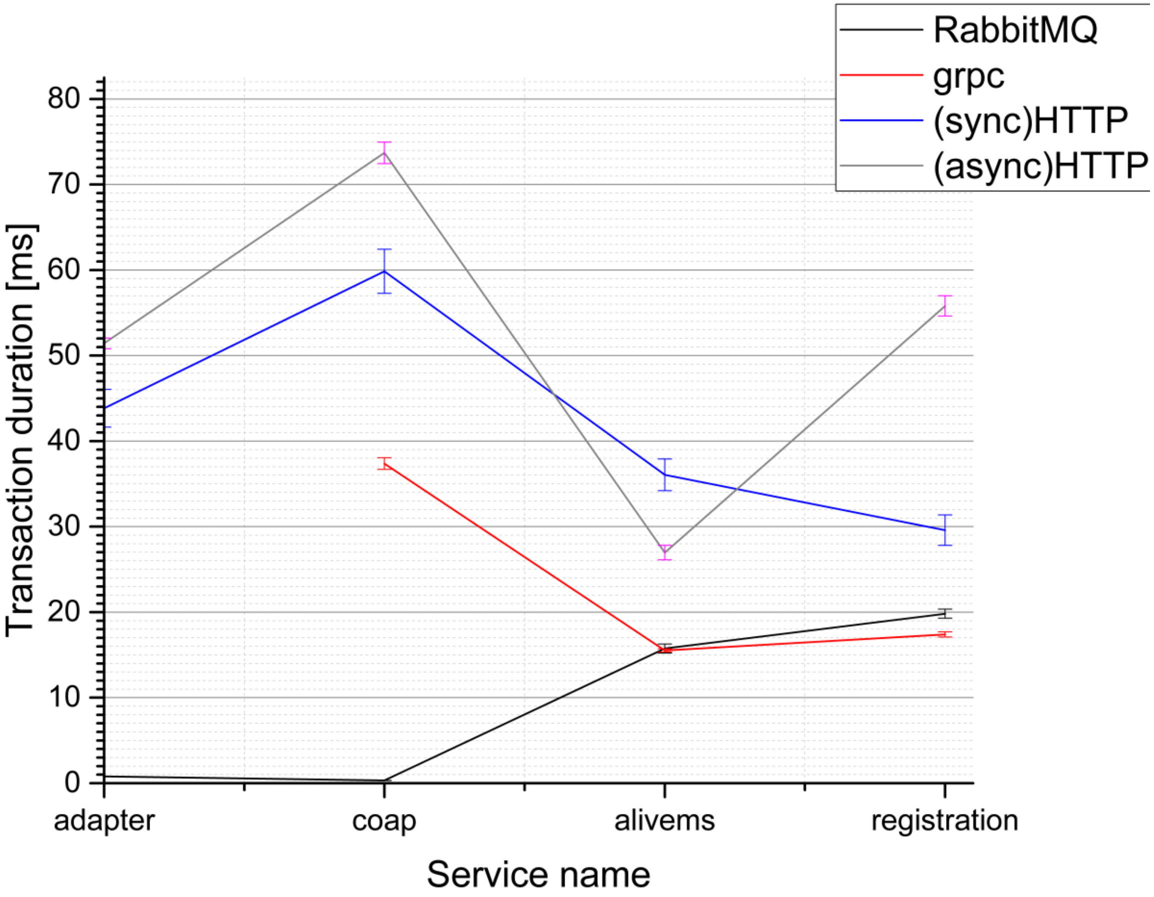
- I’m too stupid I can’t understand what I have to write about response time.



# Transaction duration by service

- gRPC interconnection provides lowest transaction duration time of persistence services.

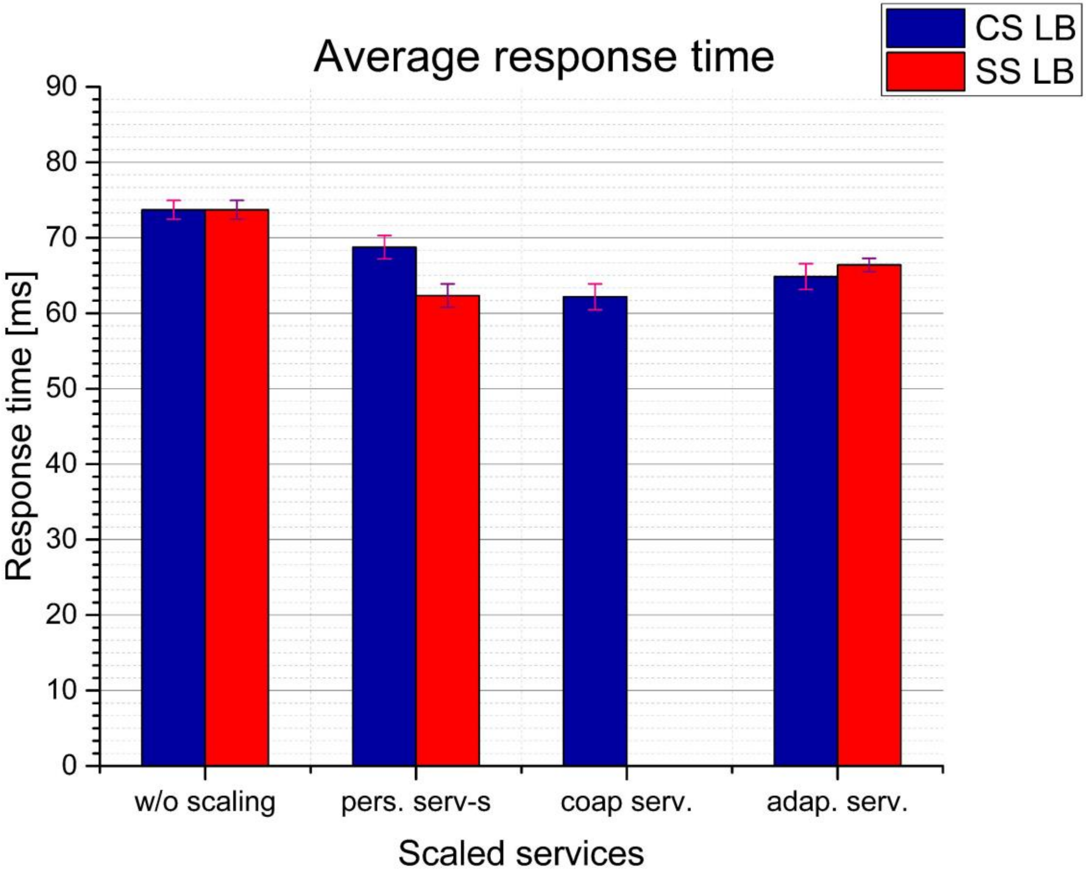
- RabbitMQ provides the lowest transaction duration time of non-persistence services.



# Load balancing strategies comparison

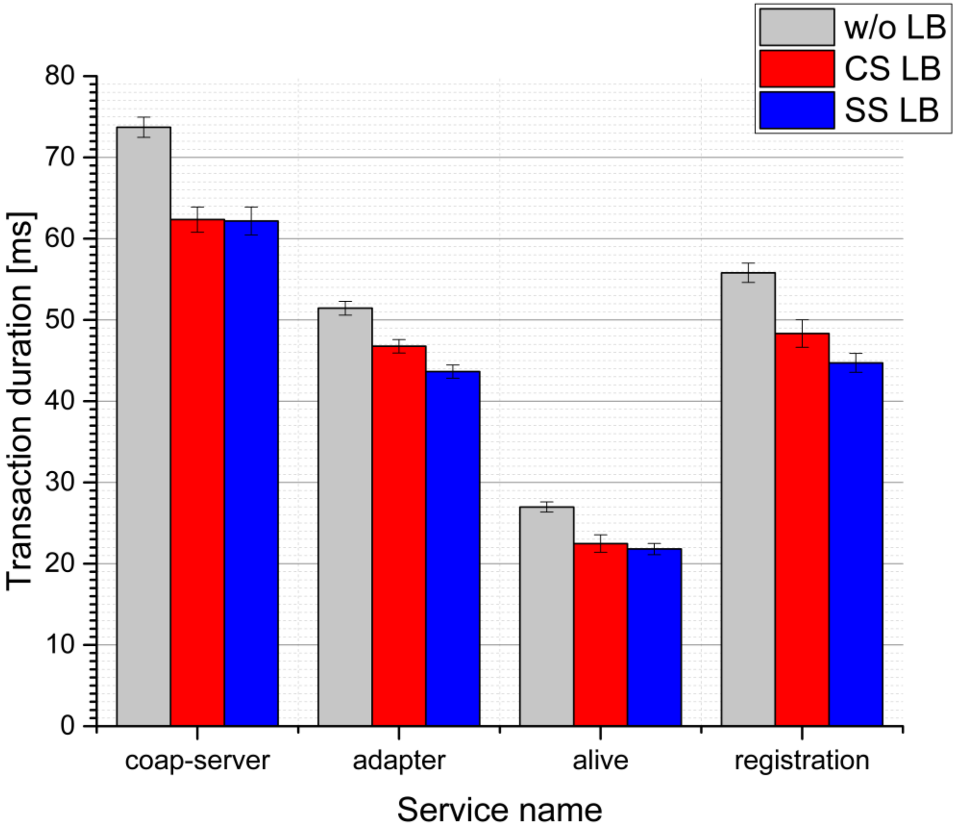
- Client-side LB strategy provides about 10% less a system response time, scaling persistency services.

- Server-side LB strategy improves the system response by almost the same amount but by scaling edge service.



# LB Impact on services

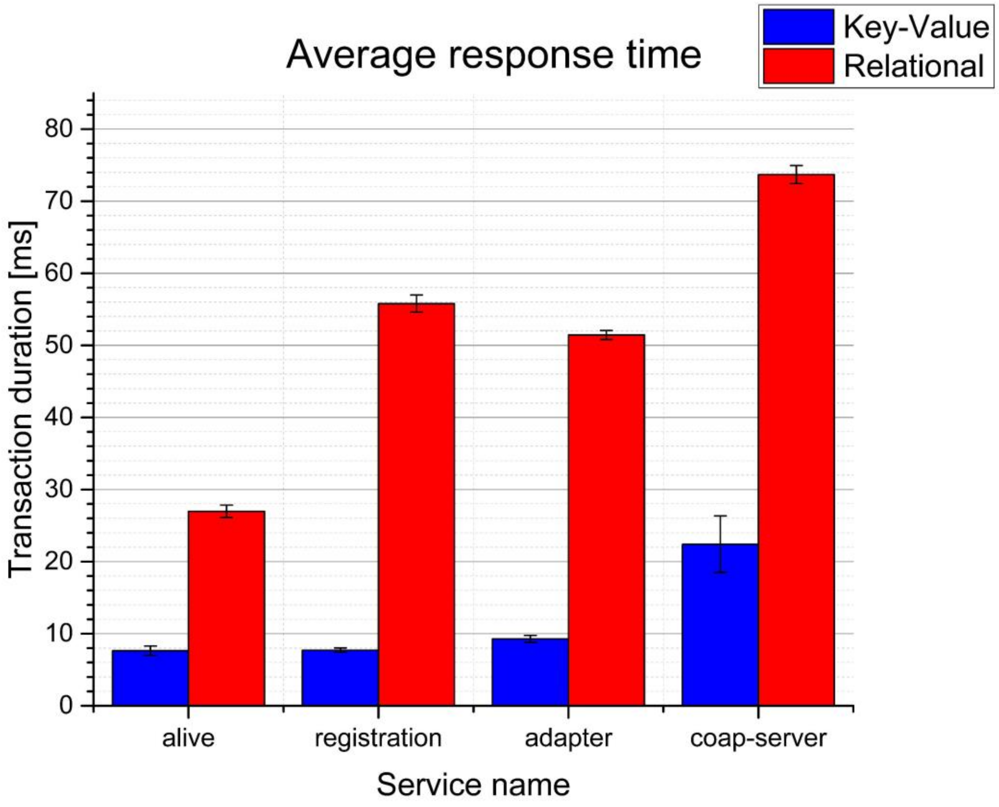
- Server-side LB provides slightly faster overall response time.



# DBMS comparison

- Key-Value DBMS requires significantly less time to execute database operations.

- To process a DB operation, it needs 8 milliseconds, approximately.



# Conclusion

# Conclusion

- Connect non-persistence services via RabbitMQ and persistence services via gRPC.
- Document-oriented key-value DBMS can ensure the lowest transaction duration of persistence services.
- Load balancing strategy choice depends on which service has to be scaled. Also, SS provides a little faster response time.

# Future work

- To test the microservice system with some additional services satisfying more realistic requirements.
- Improve the load generator.
- Survey about how caching might affect the system response time.
- Prove our research for production usage.

# Thank you for attention