SERVICE ORIENTED SOFTWARE ENGINEERING - SOFTWARE ENGINEERING FOR AUTONOMOUS SYSTEMS
A.A. 2018/2019

NETFLICS

ce**N**tralized, s**E**rvice orien**T**ed, sel**F** adaptive, app**LIC**ation for movie **S**treaming

VALENTINA CECCHINI, m. 255596
STEFANO VALENTINI, m. 254825

OVERVIEW

Video streaming application based on **web-services** and **self-adaptivity**.

ARCHITECTURE

Orchestration between **SOAP** and **REST** services.
MAPE-K based load balancer.

DEMO

Demonstration of **how** the system works and how the load balancer **organizes** the workload.

OVERVIEW

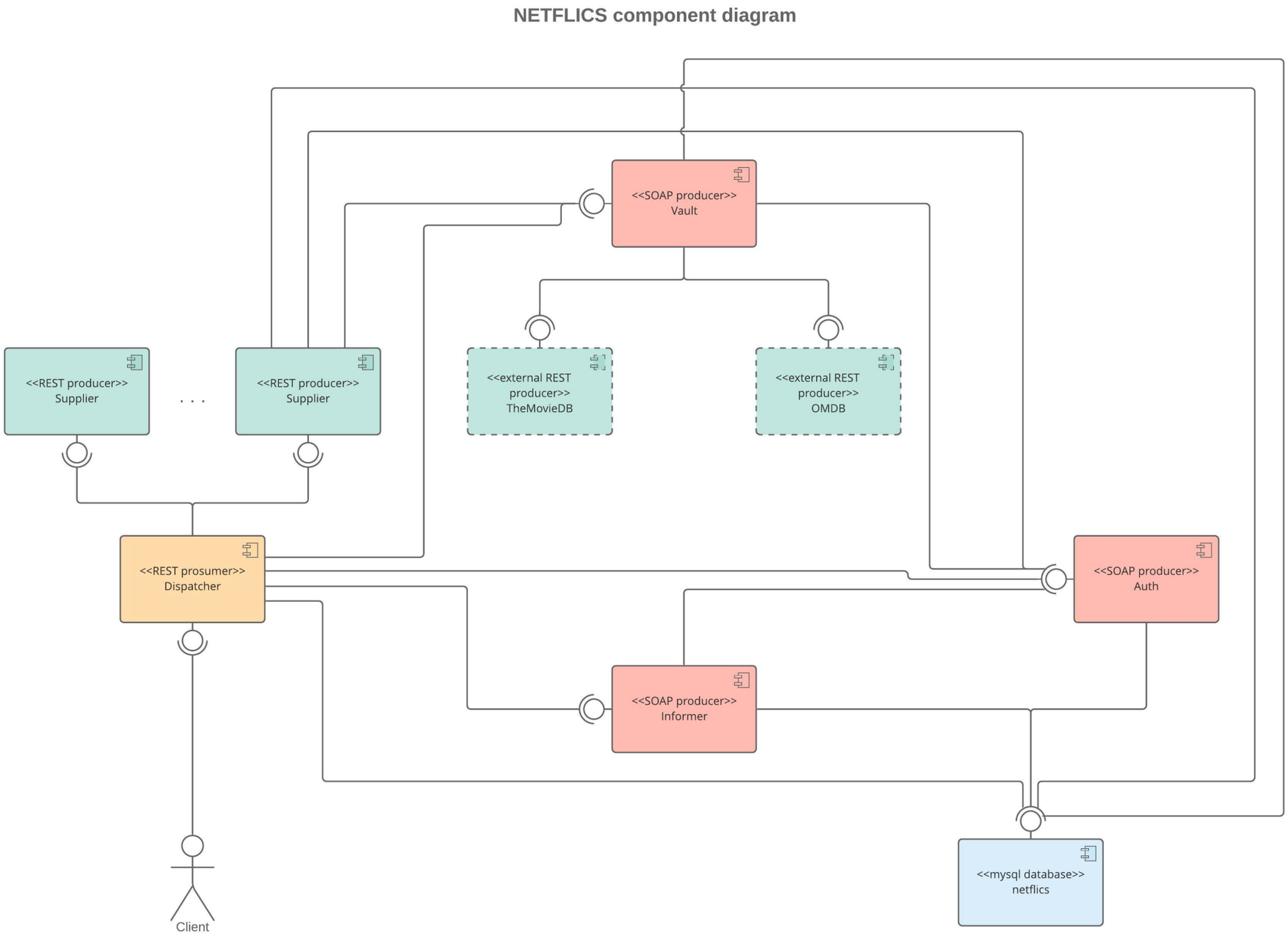
The aim of this project is to build a **video streaming** application based on **web services**, with particular focus on **self-adaptivity**.

The system exploits the **interactions** between services to achieve a great level of **availability** while avoiding waste of resources through the implementation of a **load-balancing**, a mechanism that is able to:

"**turn off**" certain components of the system when not needed

wake them **back up** in case of an high volume of incoming requests is detected

SYSTEM ARCHITECTURE



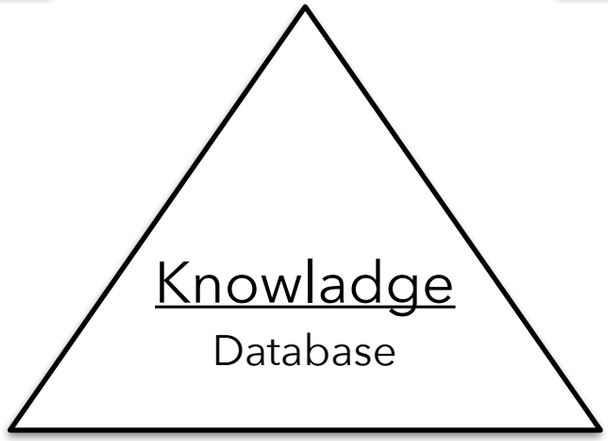
LOAD BALANCING

Analyze

The Dispatcher retrieves all the awake and free suppliers and check which of them has the requested movie and which has not.

Plan

This phase consists of filling certain lists that will be later processed by the execute phase, based on the results of previous phases



Monitor

The dispatcher checks sleeping and awake suppliers. Verifies the availability of all the awake suppliers to decide whether a supplier is “free” or not (based based on number of free slots).

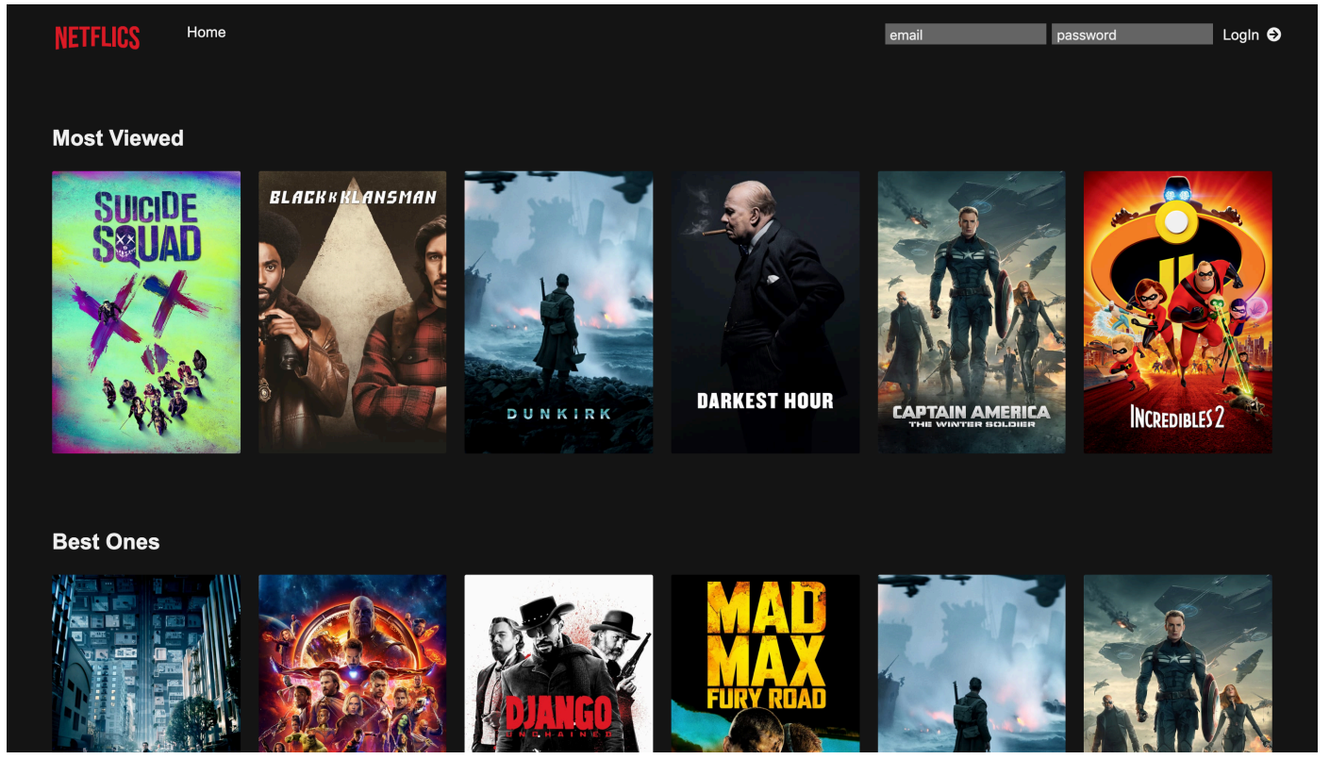
Execute

This phase consists on sending the respective commands to the right suppliers, by looking at the previously filled lists.

System

WEB CLIENT

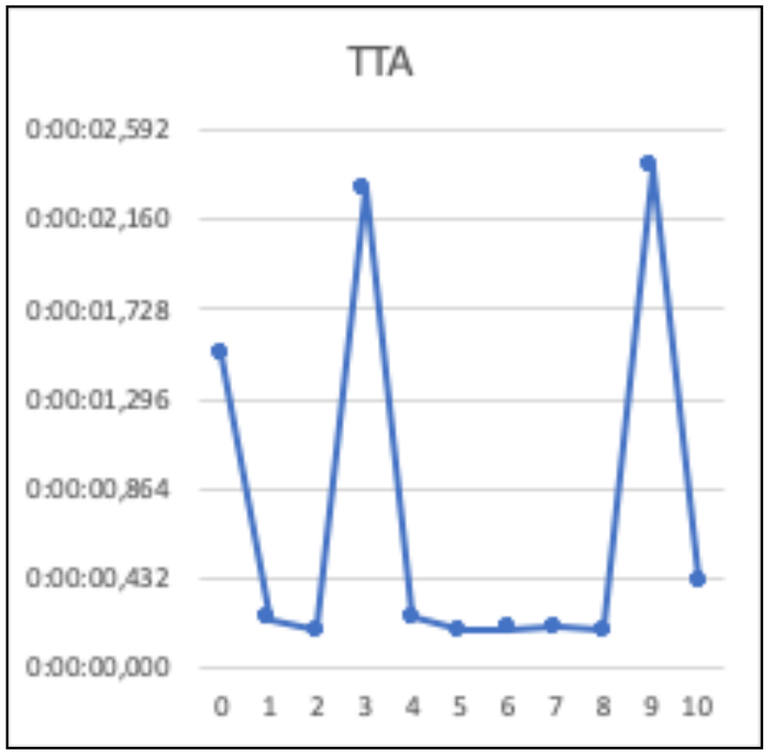
The webclient has been implemented with **VueJS**, this Javascript framework allows to manipulate the HTML view by **binding** the data **entities** defined in the **.vue file** with the HTML components; it also allows to **store data locally** (so to avoid to perform the login on each page refresh, by saving the token on local storage, for example).
REST calls to the dispatcher service are performed through the **axios** framework while the **video stream** is rendered using the **videojs** framework.



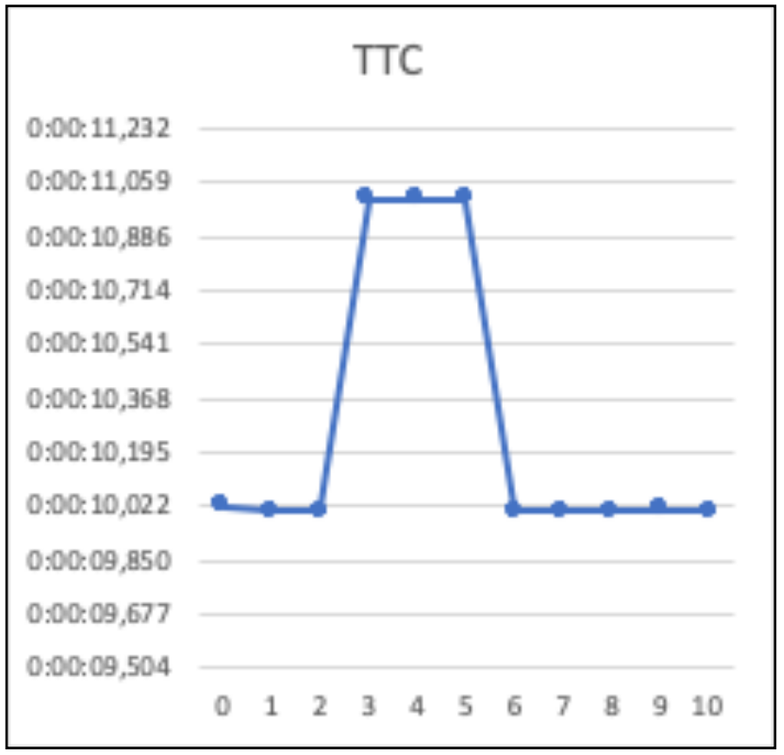
LOAD BALANCER - TEST 1

3 suppliers: all awake from the beginning

- Average time to accept the request (**TTA**): 0.734 s



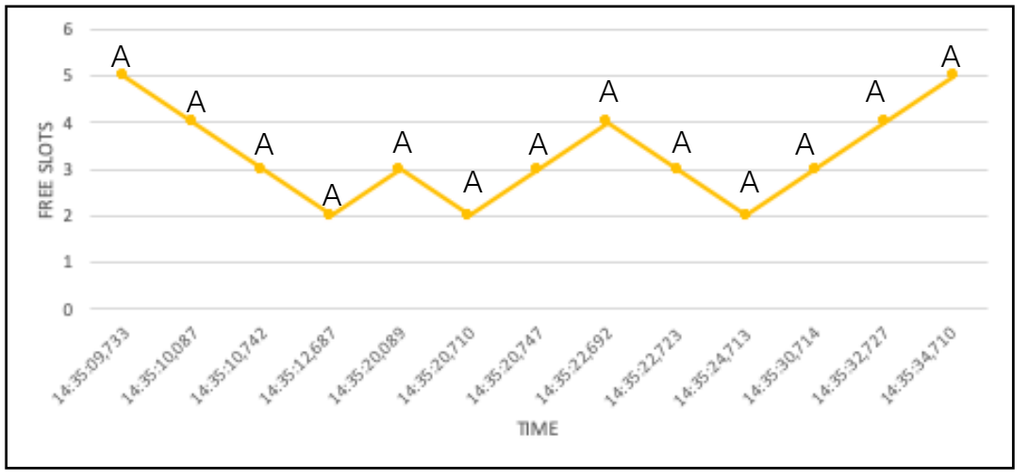
- Average time to complete the request (**TTC**): 0.734 s



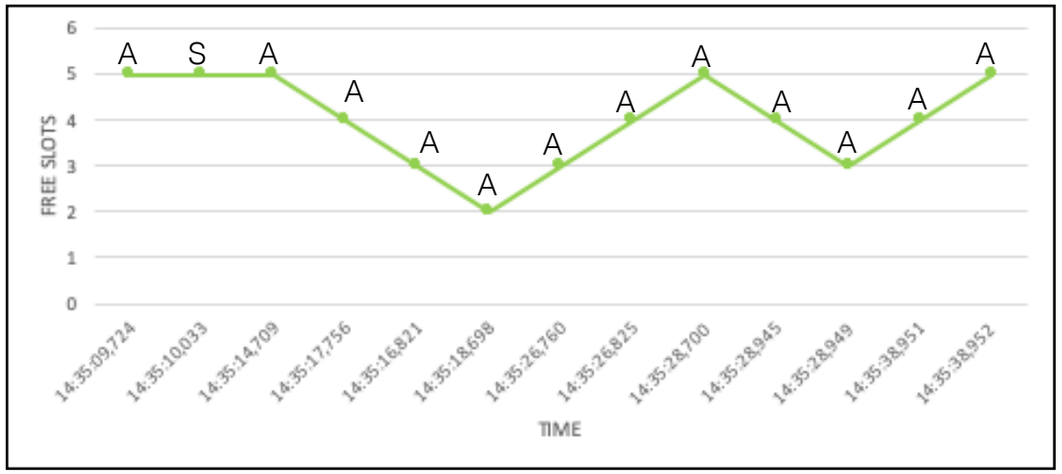
LOAD BALANCER - TEST 1

3 suppliers: all awake from the beginning

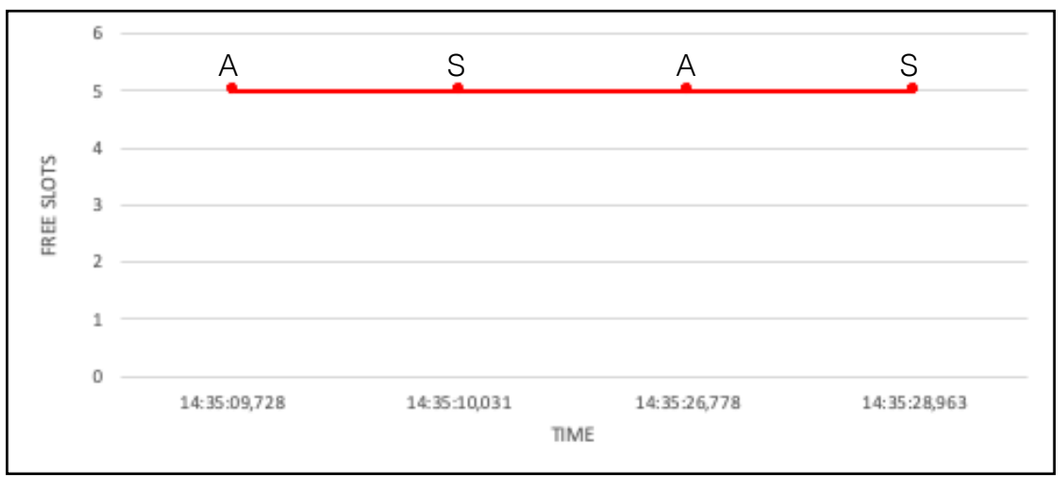
SUPPLIER 1:
- serves requests 0, 1, 2, 6 ,7 ,8
- average free slots: 3



SUPPLIER 2 :
- serves requests 3, 4, 5, 9, 10
- average free slots: 4



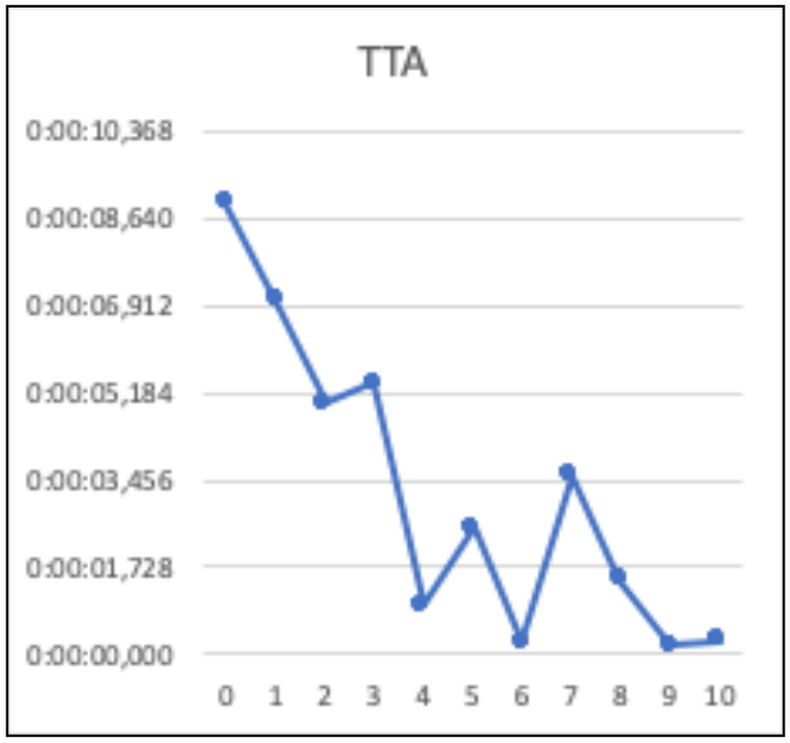
SUPPLIER 3 :
- serves no requests
- average free slots: 5



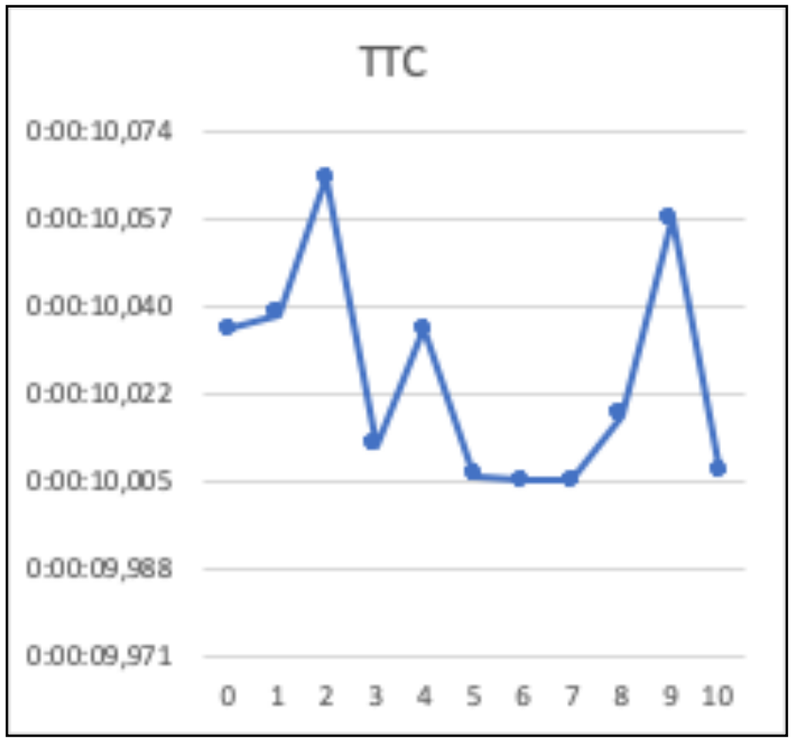
LOAD BALANCER - TEST 2

3 suppliers: all sleeping from the beginning

- Average time to accept the request (**TTA**): 3.225 s



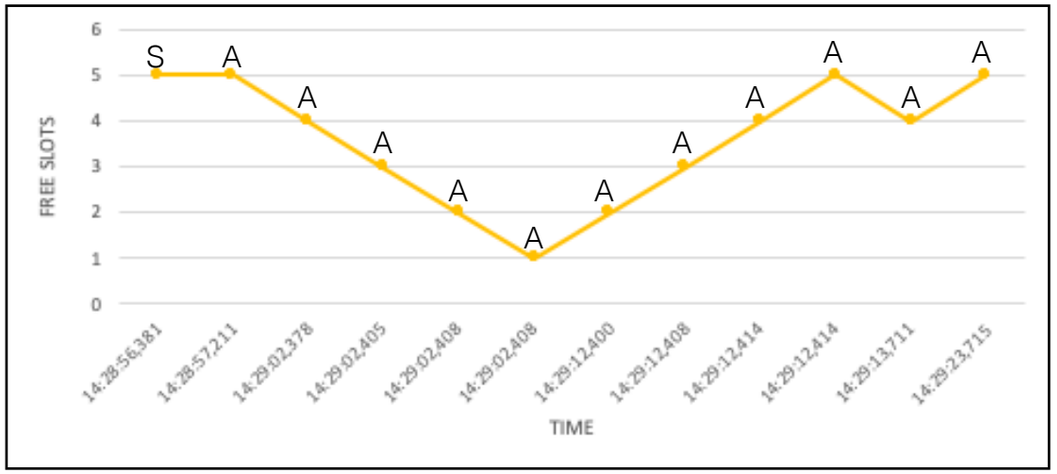
- Average time to complete the request (**TTC**): 10.026 s



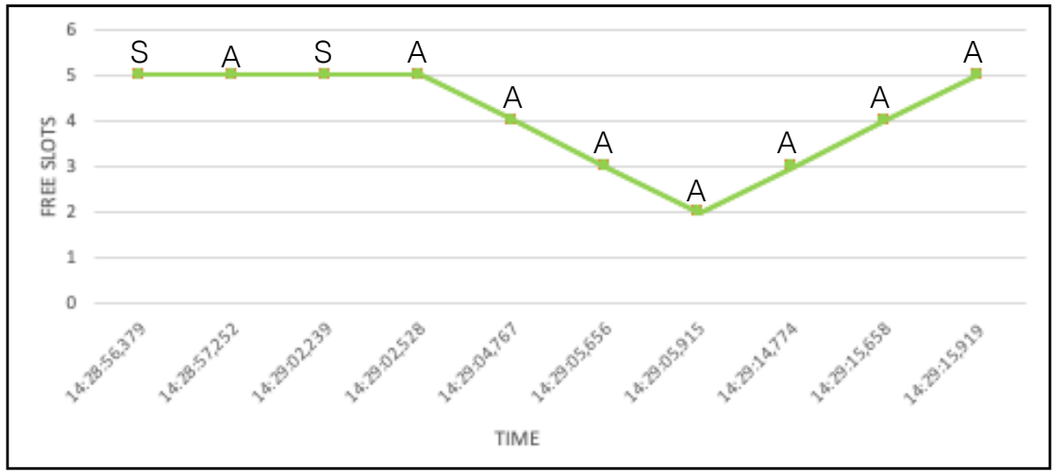
LOAD BALANCER - TEST 1

3 suppliers: all awake from the beginning

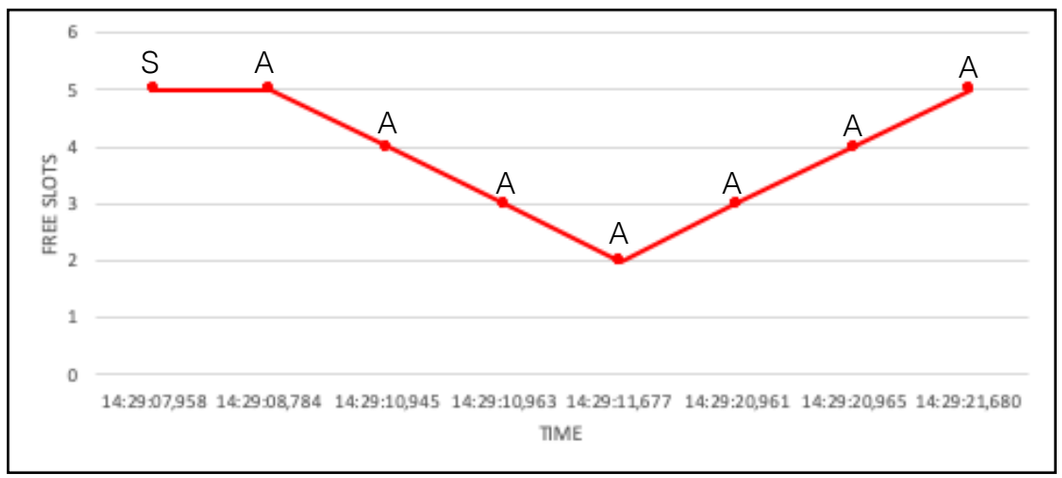
SUPPLIER 1:
- serves requests 0, 1, 2, 4, 10
- average free slots: 4



SUPPLIER 2 :
- serves requests 3, 5, 6
- average free slots: 4



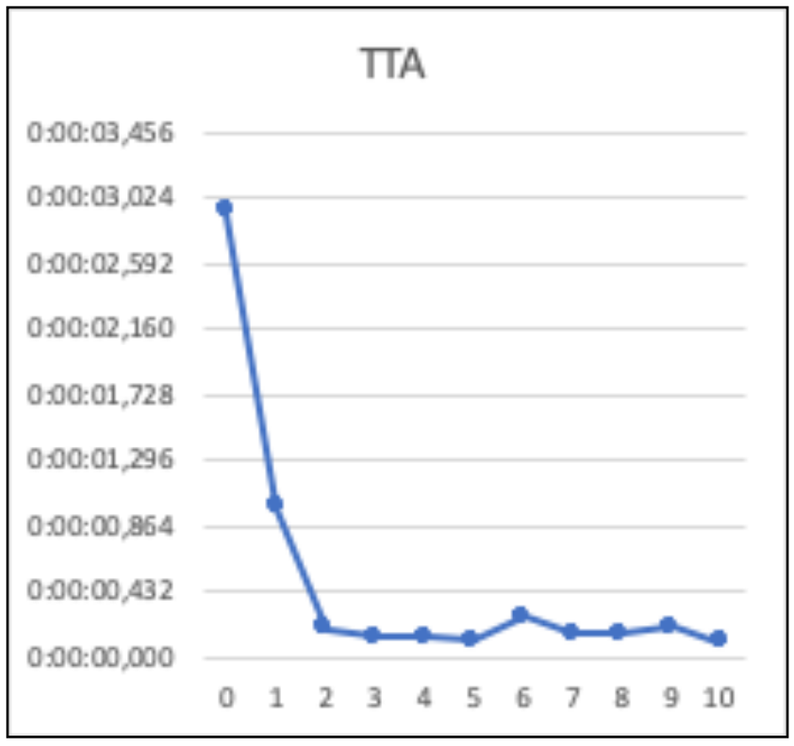
SUPPLIER 3 :
- server requests 7, 8, 9
- average free slots: 4



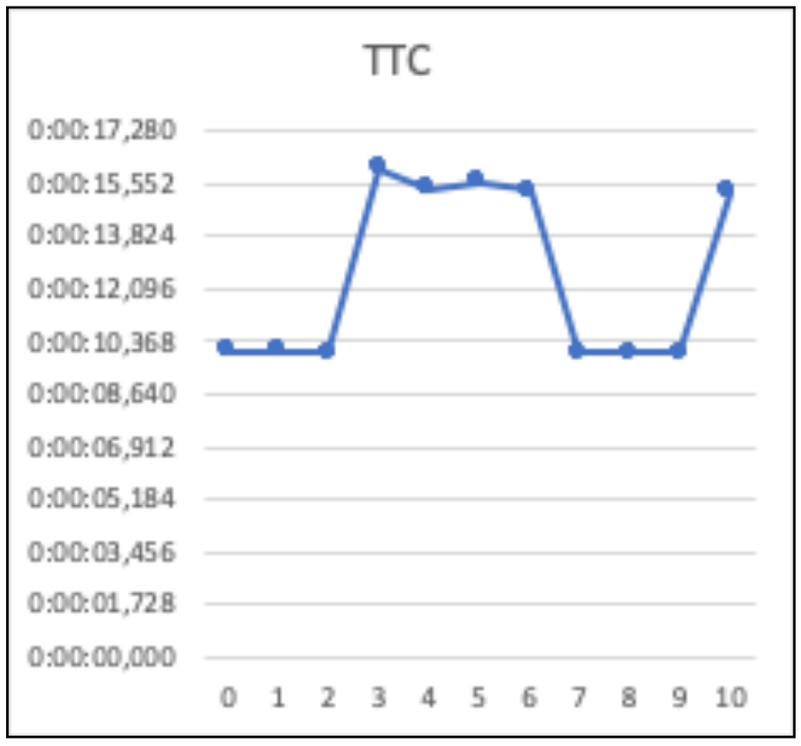
LOAD BALANCER - TEST 3

3 suppliers: supplier1 awake; supplier 2 and supplier 3 sleeping

- Average time to accept the request (**TTA**): 0.484 s



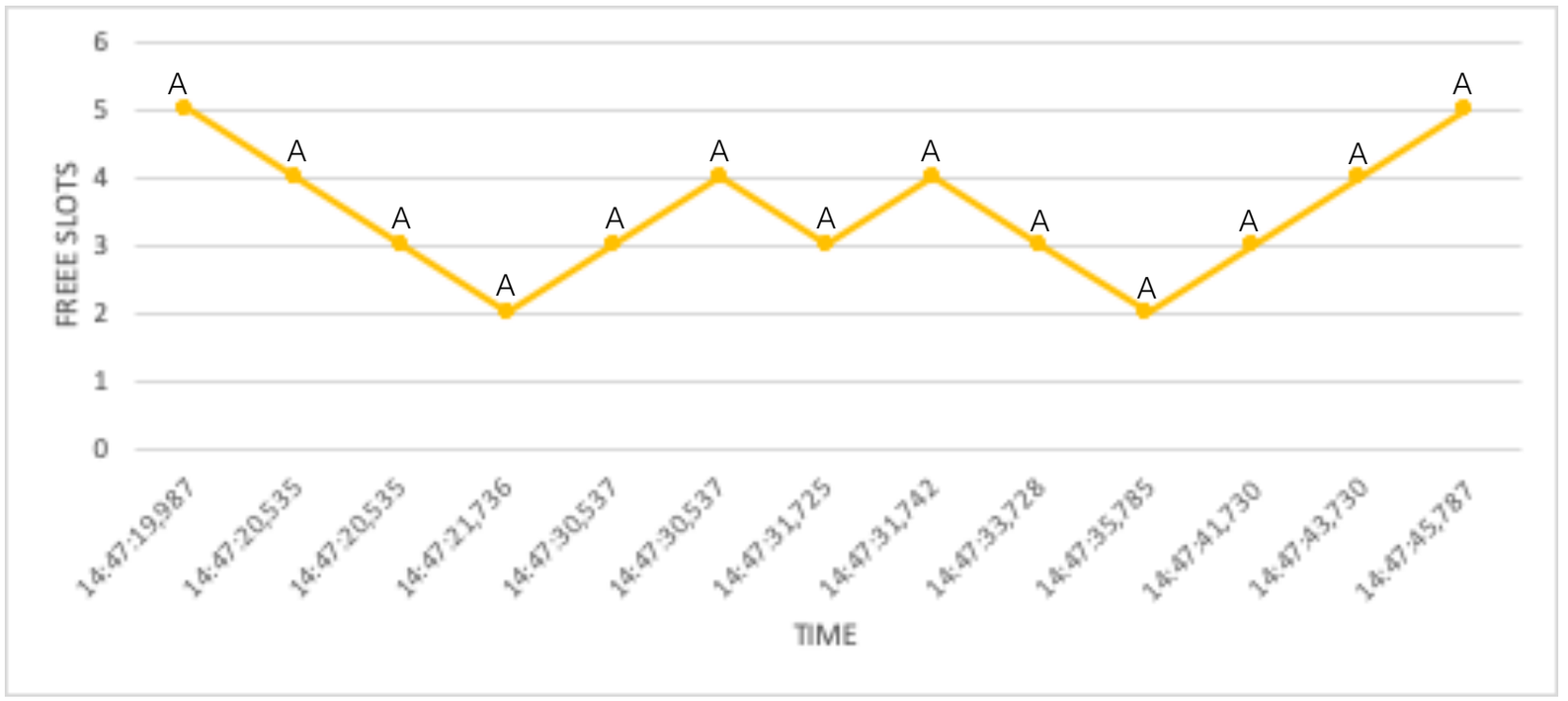
- Average time to complete the request (**TTC**): 12.515 s



LOAD BALANCER - TEST 3

3 suppliers: supplier1 awake; supplier 2 and supplier 3 are offline

SUPPLIER 1:
- serves requests 0, 1, 2, 7, 8, 9; the other requests have been sent to the *Vault*
- average free slots: 3



DEMO

https://youtu.be/fQPH8HF32iM

THANKS FOR YOUR ATTENTION