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

Software Engineering for Autonomous Systems/Service Oriented Software Engineering

Valentina Cecchini – 255596
Stefano Valentini – 254825

demo video: https://youtu.be/fQPH8HF32iM

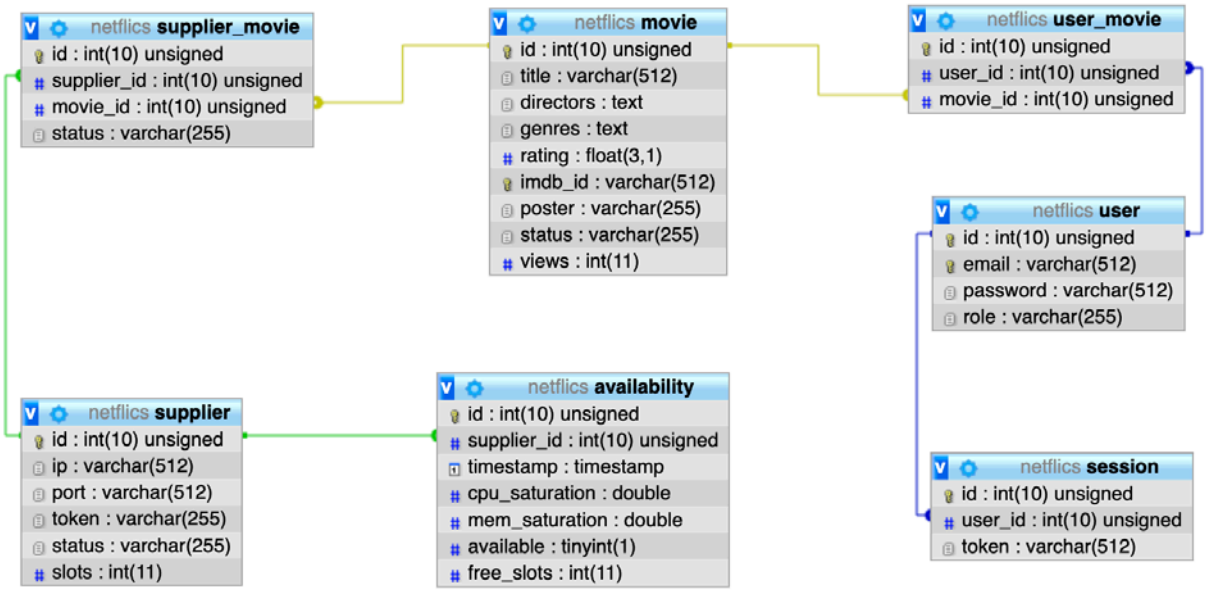
**full resolution images can be found along with this document, test logs can be found in the /test folder*

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 and to achieve a great level of availability while avoiding waste of resources through the implementation of a load-balancing mechanism that is able to “turn off” certain components of the system when not needed and to wake them back up in case of an high volume of incoming requests is detected.

E-R MODEL



The database underlying the system is organized in the following way:

There are three principal entities: *user*, *movie* and *supplier*; for the *user* we are interested in saving the e-mail and password and in distinguishing them by the *role* that can be "ADMIN" or "USER".

For each *user* we want to record each log-in through the table *session* and the set of movies that he has watched through the table *user_movie*.

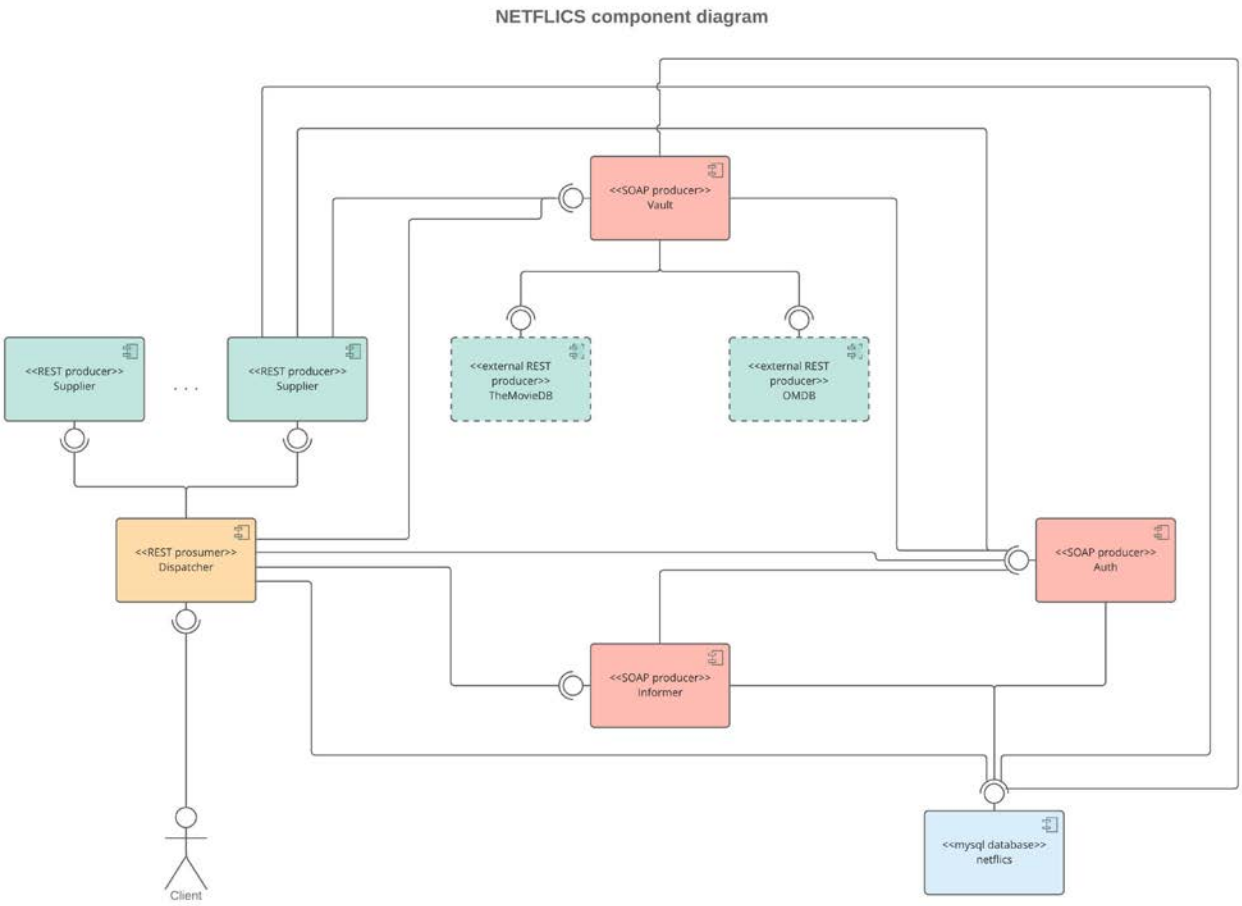
For the *supplier* entity, we want to keep data about where each single supplier can be found, a way to recognize it, it's status and how many slots (i.e., how much client connections it can holds) it has.

For reach *supplier* we desire to store the *availability*, its resource consumption and its number of free slots, at any given time.

For the *movie* entity we want save all the metadata and the number of times that the movie is seen.

The *supplier_movie* entity holds information about whether a given *supplier* has a certain movie.

SYSTEM'S ARCHITECTURE



SOAP services:

- **Auth**: is the service that is responsible for checking the user's credentials to allow it to log in/out and access the system's functionalities; endpoints**:**
  - **`logIn(email, password) -> {result, role, token}`**
    `checks the user's credentials and performs the log in action, returning the role and a new token for the user to use.`
  - **`logOut(token) -> {result}`**
    `deletes the session associated to the received token.`
  - **`checkToken(token) -> {result, role}`**
    `checks the token and returns the role of the user.`

- **Vault**: is the service that is responsible for keeping all the movie files on disk and to send them to the suppliers that request them**:**
  - **`getMovie(imdb_id, token) -> {movie, result}`**
    `returns the movie data associated to the imdb_id, if the user is authorized.`
  - **`addMovie(imdb_id, movie, token) -> {result}`**
    `saves to disk the movie data that it has received, retrieves the metadata of the movie from TheMovieDB and OMDB external services and stores them into the db.`

- **Informer:** is the service that is responsible for retrieving information from the database, this kind of information will be used for the webclient gui (list of the most viewed movies, etc.).

REST services

- **Supplier**: is the service that is responsible for retrieving the movie data from the *Vault* service and make it available for the *Dispatcher*; endpoints:
  - **GET /{token}/movie/{id}**
    returns the requested movie data as a StreamingOutput.
  - **GET /{token}/availability**
    returns the current system resource occupation.
  - **POST /{token}/movie/{id}**
    commands this supplier to fetch the movie data from the Vault. service.
  - **POST /{token}/wakeup**
    commands this supplier to wake up.
  - **POST /{token}/sleep**
    commands this supplier to go to sleep.

- **Dispatcher**: is the only component of the system that communicates with the client(s), it's main task is the one that consist on permitting the client to stream the movie by requesting it from the "best" *supplier* (the mechanism by which this is accomplished is detailed in the "Load Balancing" section of this document; endpoints:
  - **PUT /{token}/movie/{imdbId}**
    forwards the received movie data to the Vault.addMovie service (reserved to admins).
  - **POST /{token}/logout**
    calls Auth.logOut.
  - **GET /movie/mostviewed**
    returns the data that is retrieved by the Informer.mostViewed service.
  - **GET /movie/bestones**
    returns the data that is retrieved by the Informer.bestOnes service.
  - **GET /{token}/movie/lastviewed**
    returns the data that is retrieved by the Informer.lastViewed service.
  - **GET /{token}/movie/stream/{imdbId}**
    main functionality of this component, finds the "best" supplier and opens a bridge stream between such supplier and the client.

LOAD BALANCING

The load balancing procedure is based on the **MAPE-K** cycle; it is implemented in this system by the *LoadBalancer* class in the *Dispatcher* service and it starts any time the *Dispatcher* receives a stream request by a client.

MONITOR: the *Dispatcher* retrieves from the database the list of the sleeping suppliers (and puts them into the *sleepingSuppliers* list) and the suppliers that are awake; then the *Dispatcher* sends an availability request to all the awake suppliers (in parallel), if any; so to retrieve their availability data (in this case the number of free slots).

When the availability data is returned, the system decides whether a given supplier is considered "free" or not by looking at the number of its free slots, so to build the *freeSuppliers* list.

ANALYZE: the *Dispatcher* retrieves the list of all the suppliers that currently have the requested movie data, then it divides them in two groups: the suppliers that are "free" and currently **have** the movie data and the suppliers that are "free" and currently **do not have** the movie data, so to fill the *freeSupplierHavingMovie* and *freeSuppliersNotHavingMovie* lists.

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

If the list *freeSuppliers* has more than 0 entries then it means that there are free suppliers, so check whether there are free suppliers having the movie by looking at the size of the *freeSuppliersHavingMovie* list; if so, choose the one that has the highest amount of free slots, lets call this supplier *chosenOne*; if, instead, the list is empty, but the *freeSupplier* one is not. it must mean that the *freeSuppliersNotHavingMovie* is not empty: in this case choose the best one in terms of free slots and add it to the *suppliersToFetch* list (meaning that it will be commanded to retrieve the movie data from the *Vault* service).

Now, we are still in the case in which the *freeSuppliers* list is not empty, in this case there may be some supplier that is idle, i.e., the number of its free slots is equal to the number of its total slots, if so, add it to the *suppliersToSleep* list, paying attention to exclude the *chosenOne* and all the *suppliersToFetch* from this list.

While, if the *freeSuppliers* list is empty we do not perform any of the previously described actions and we check, instead, if there are some sleeping suppliers, by looking at the size of the *sleepingSuppliers* list: if so, we choose some random sleeping supplier(s) and we put it(them) into the *suppliersToWake* list, we also put them into the *suppliersToFetch* list, since after having been woken up they will not have any movie data available.

Finally, if there no *freeSuppliers* and there are no *sleepingSuppliers* it means that the only choice we have is to directly call the *Vault* service to stream back its data to the user, notice that this is an extreme situation that should not happen.

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

For each element of a given list the appropriate request is performed, e.g., a `POST /{token}/sleep` is sent to each member of the *suppliersToSleep* list.

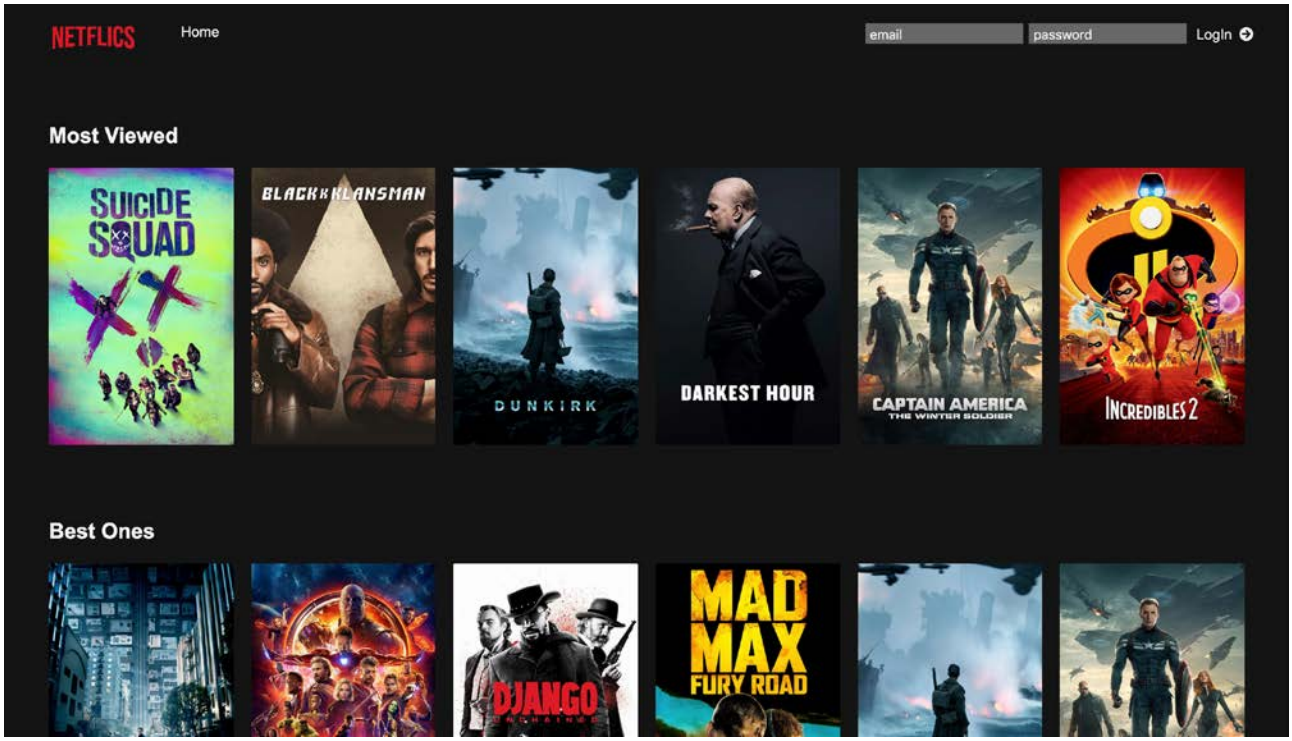
Now, if the *chosenOne* has been previously identified, a stream from that supplier to the client is opened, otherwise the stream will be opened from the *Vault.*

If, in the plan phase, neither the *chosenOne* has been identified, neither it has been decided to stream from the *Vault* (i.e., we are waiting for some supplier to wake up and fetch the movie) the procedure is restarted after a fixed amount of time, for at most *X* times.

WEBCLIENT

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.



TESTS

To test the application a *testclient* java application has been developed: this application sends a certain number of requestst to the `GET /{token}/movie/stream/{imdbId}` endpoint, one each *X* second(s); for each test the number of requests to perform is *10*, each one every *2* seconds.

Since the machine in which the services are deployed is able to handle the stream requests in milliseconds, an artificial delay of *10* seconds and *15* seconds has been added on the *supplier* request and *vault* request respectively.

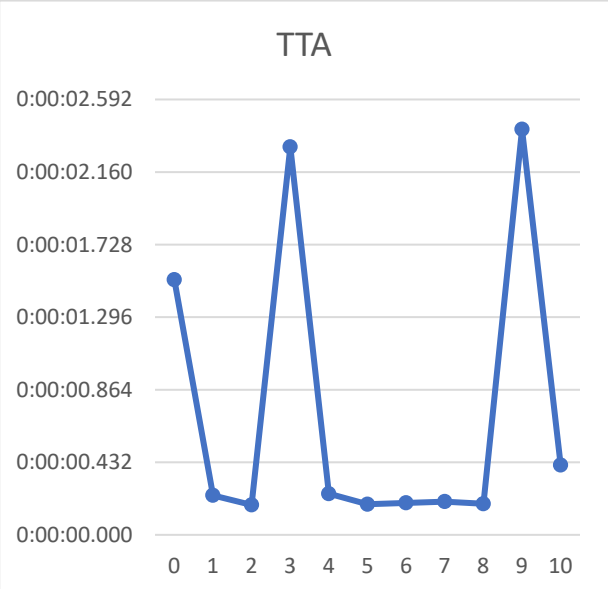
Suppliers with a number of free slots <= *2* are considered *not free*.

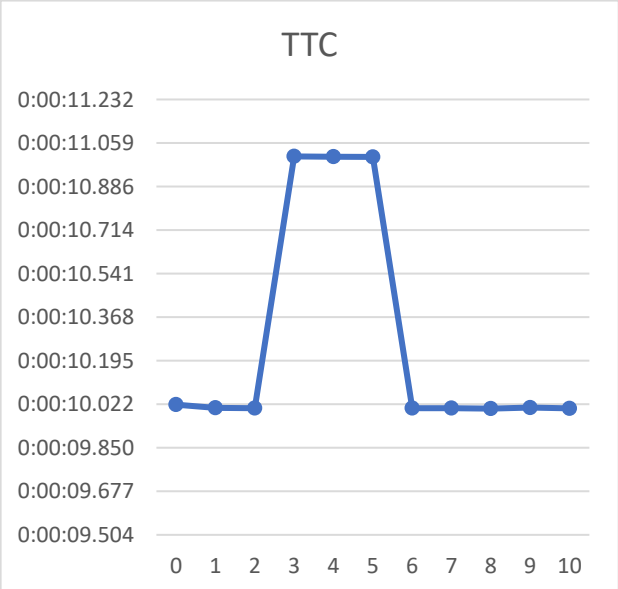
TEST 1

- supplier 1: *AWAKE*
- supplier 2: *AWAKE*
- supplier 3: *AWAKE*

REQUEST #	START TIME	ACCEPT TIME	END TIME	TTA	TTC
0	14:35:08,568	14:35:10,087	14:35:20,108	0:00:01,519	0:00:10,021
1	14:35:10,507	14:35:10,742	14:35:20,750	0:00:00,235	0:00:10,008
2	14:35:12,509	14:35:12,687	14:35:22,694	0:00:00,178	0:00:10,007
3	14:35:14,510	14:35:16,820	14:35:27,826	0:00:02,310	0:00:11,006
4	14:35:16,511	14:35:16,756	14:35:27,761	0:00:00,245	0:00:11,005
5	14:35:18,516	14:35:18,698	14:35:29,702	0:00:00,182	0:00:11,004
6	14:35:20,520	14:35:20,710	14:35:30,717	0:00:00,190	0:00:10,007
7	14:35:22,525	14:35:22,723	14:35:32,730	0:00:00,198	0:00:10,007
8	14:35:24,528	14:35:24,713	14:35:34,718	0:00:00,185	0:00:10,005
9	14:35:26,531	14:35:28,945	14:35:38,954	0:00:02,414	0:00:10,009
10	14:35:28,534	14:35:28,949	14:35:38,955	0:00:00,415	0:00:10,006

Average **TTA** (Time To Accept): **0.734s**
Average **TTC** (Time To Complete)**: 10.280s**

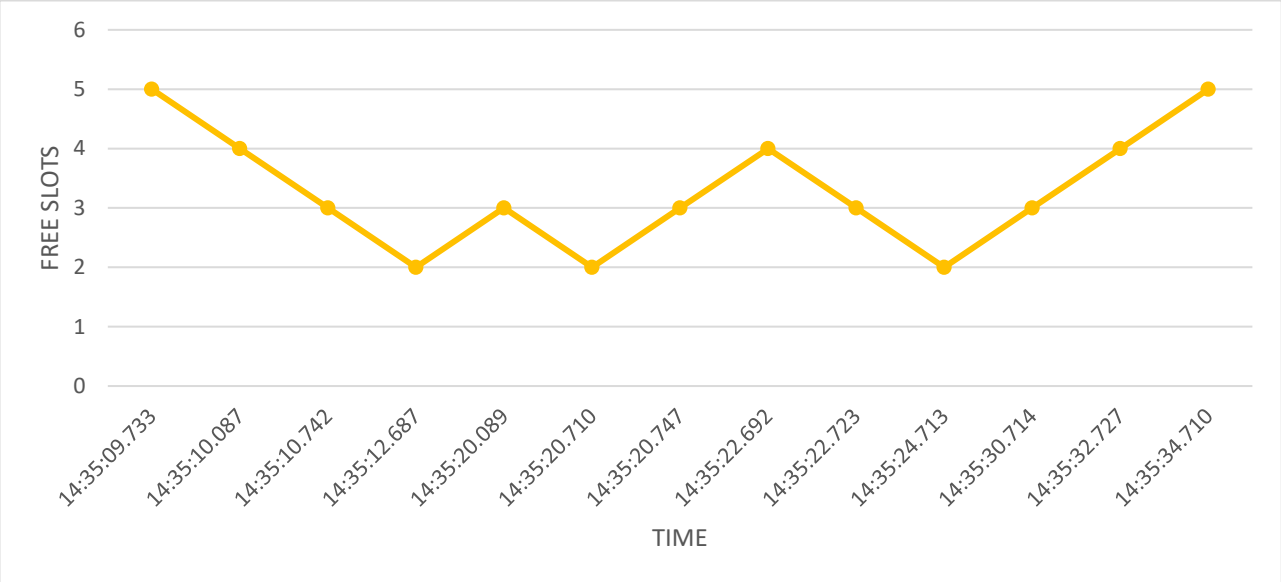




SUPPLIER 1 (serves requests 0, 1, 2, 6 ,7 ,8):

TIME	FREE SLOTS	STATUS
14:35:09,733	5	AWAKE
14:35:10,087	4	AWAKE
14:35:10,742	3	AWAKE
14:35:12,687	2	AWAKE
14:35:20,089	3	AWAKE
14:35:20,710	2	AWAKE
14:35:20,747	3	AWAKE
14:35:22,692	4	AWAKE
14:35:22,723	3	AWAKE
14:35:24,713	2	AWAKE
14:35:30,714	3	AWAKE
14:35:32,727	4	AWAKE
14:35:34,710	5	AWAKE

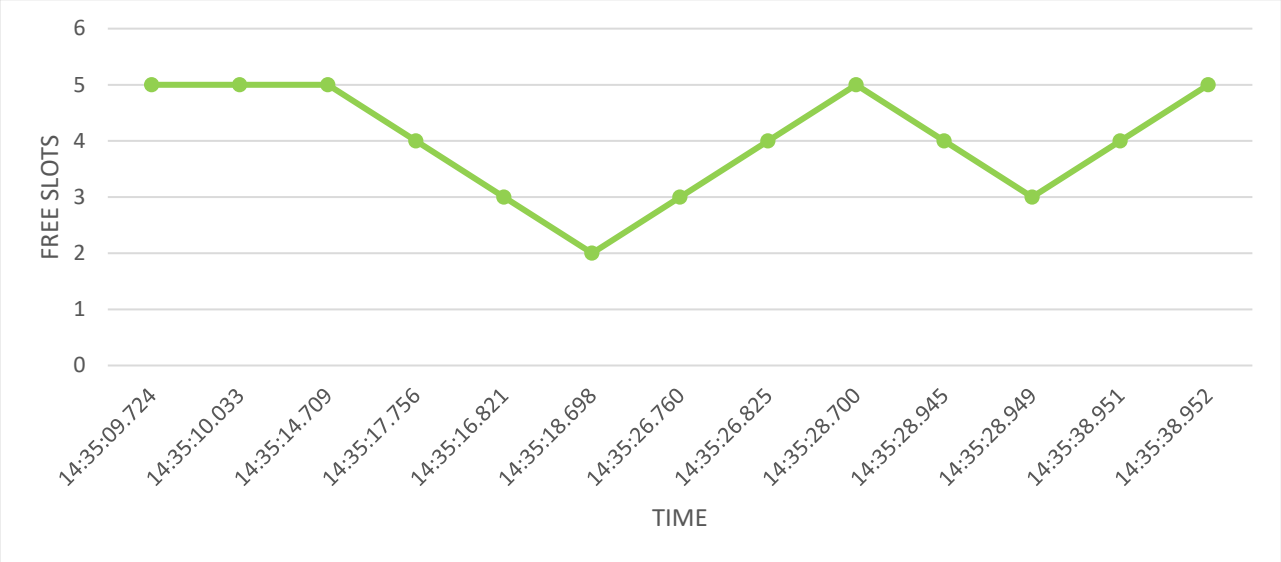
Average free slots: **3**



SUPPLIER 2 (serves requests 3, 4, 5, 9, 10):

TIME	FREE SLOTS	STATUS
14:35:09,724	5	AWAKE
14:35:10,033	5	SLEEP
14:35:14,709	5	AWAKE
14:35:17,756	4	AWAKE
14:35:16,821	3	AWAKE
14:35:18,698	2	AWAKE
14:35:26,760	3	AWAKE
14:35:26,825	4	AWAKE
14:35:28,700	5	AWAKE
14:35:28,945	4	AWAKE
14:35:28,949	3	AWAKE
14:35:38,951	4	AWAKE
14:35:38,952	5	AWAKE

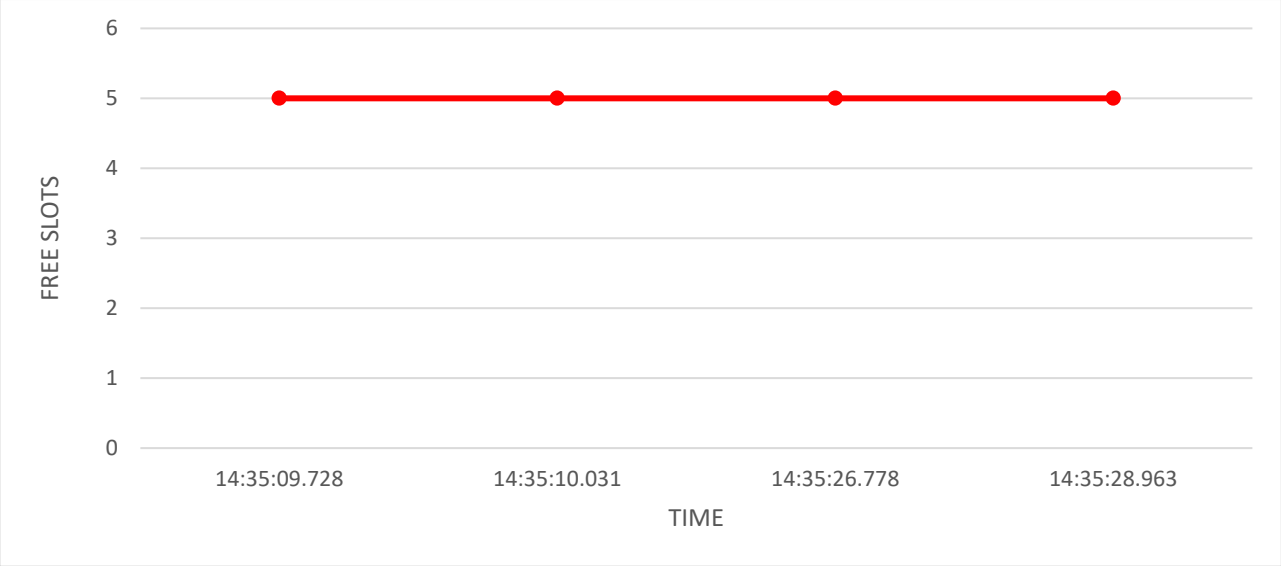
Average free slots: **4**



SUPPLIER 3 (serves no requests)

TIME	FREE SLOTS	STATUS
14:35:09,728	5	AWAKE
14:35:10,031	5	SLEEP
14:35:26,778	5	AWAKE
14:35:28,963	5	SLEEP

Average free slots: **5**



This supplier shows an anomaly dictated by the asynchronous behaviour of this system, it starts in an AWAKE state, since supplier 1 and 2 are chosen to handle the first requests it is put to sleep.

When request 9 is received, the system sees supplier 1 and supplier 2 as not free, so it awakes supplier 3 and restarts the procedure; but, in the next iteration supplier 2 becomes free and it is chosen to handle request; since supplier 1 is still not idle it is not put to sleep, but supplier 3 is (since all its slots are free as it has just been woken up); logs:

```
14:35:26,578 - MONITOR PHASE, token: 9
14:35:26,579 - there are 1 sleeping suppliers, token: 9
14:35:26,580 - there are 2 awake suppliers, token: 9
14:35:26,581 - sending getAvailability to supplier: 1 - localhost:8081, token: 9
14:35:26,581 - sending getAvailability to supplier: 2 - localhost:8082, token: 9
14:35:26,639 - supplier 1 is available
14:35:26,639 - supplier 2 is available
14:35:26,639 - supplier 1 is not free (2 slots are available), token: 9
14:35:26,639 - supplier 2 is not free (2 slots are available), token: 9
14:35:26,640 - ANALYZE PHASE, token: 9
14:35:26,643 - PLAN PHASE, token: 9
14:35:26,643 - there are no free suppliers available, token: 9
14:35:26,643 - there are sleeping suppliers, token: 9
14:35:26,643 - waiting for suppliers to wake up and fetch, token: 9
14:35:26,643 - EXECUTE PHASE, token: 9
14:35:26,643 - sending wake up..., token: 9
14:35:26,643 - sending fetchMovie..., token: 9
14:35:26,643 - sending wakeUp to supplier: 3 – localhost:8083, token: 9
14:35:26,643 - sending fetchMovie to supplier: 3 – localhost:8083, token: 9
14:35:26,646 - the requested movie is currently not available due to system overload,
try again later, token: 9
```

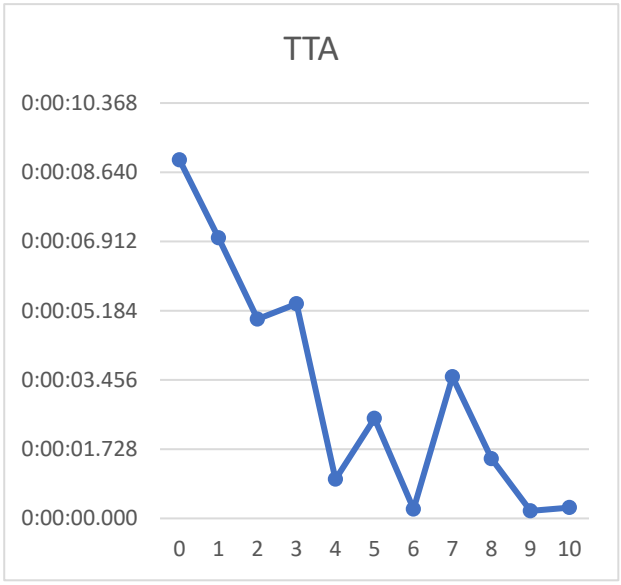
```
14:35:28,703 - MONITOR PHASE, token: 9
14:35:28,705 - there are 0 sleeping suppliers, token: 9
14:35:28,706 - there are 3 awake suppliers, token: 9
14:35:28,707 - sending getAvailability to supplier: 1 - localhost:8081, token: 9
14:35:28,707 - sending getAvailability to supplier: 2 - localhost:8082, token: 9
14:35:28,708 - sending getAvailability to supplier: 3 - localhost:8083, token: 9
14:35:28,811 - supplier 2 is available, token: 9
14:35:28,822 - supplier 2 is free (5 slots are available), token: 9
14:35:28,833 - supplier 3 is available, token: 9
14:35:28,833 - supplier 3 is free (5 slots are available), token: 9
14:35:28,837 - supplier 3 is available, token: 9
14:35:28,860 - supplier 1 is available, token: 9
14:35:28,864 - supplier 1 is not free (2 slots are available), token: 9
14:35:28,864 - ANALYZE PHASE, token: 9
14:35:28,867 - PLAN PHASE, token: 9
14:35:28,867 - there are free suppliers, token: 9
14:35:28,867 - there are free suppliers having movie, token: 9
14:35:28,867 - the best supplier has been chosen: 2, token: 9
14:35:28,867 - EXECUTE PHASE, token: 9
14:35:28,867 - sending sleep..., token: 9
14:35:28,868 - sending sleep to supplier: 3 – localhost:8083, token: 9
14:35:28,868 - starting stream from 2..., token: 9
```

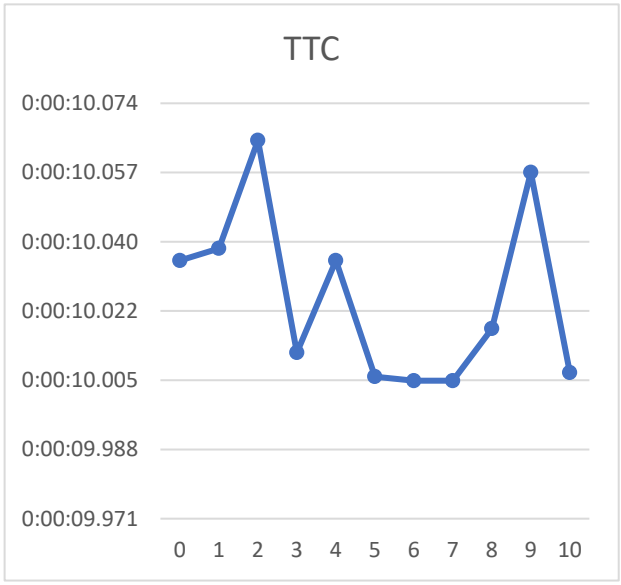
TEST 2

- supplier 1: *SLEEP*
- supplier 2: *SLEEP*
- supplier 3: *SLEEP*

ID	START TIME	ACCEPT TIME	END TIME	TTA	TTC
0	14:28:53,460	14:29:02,408	14:29:12,443	0:00:08,948	0:00:10,035
1	14:28:55,401	14:29:02,405	14:29:12,443	0:00:07,004	0:00:10,038
2	14:28:57,404	14:29:02,378	14:29:12,443	0:00:04,974	0:00:10,065
3	14:28:59,410	14:29:04,767	14:29:14,779	0:00:05,357	0:00:10,012
4	14:29:01,430	14:29:02,408	14:29:12,443	0:00:00,978	0:00:10,035
5	14:29:03,419	14:29:05,915	14:29:15,921	0:00:02,496	0:00:10,006
6	14:29:05,424	14:29:05,656	14:29:15,661	0:00:00,232	0:00:10,005
7	14:29:07,428	14:29:10,963	14:29:20,968	0:00:03,535	0:00:10,005
8	14:29:09,454	14:29:10,945	14:29:20,963	0:00:01,491	0:00:10,018
9	14:29:11,441	14:29:11,629	14:29:21,686	0:00:00,188	0:00:10,057
10	14:29:13,441	14:29:13,711	14:29:23,718	0:00:00,270	0:00:10,007

Average **TTA** (Time To Accept): **3.225s**
Average **TTC** (Time To Complete)**: 10.026s**

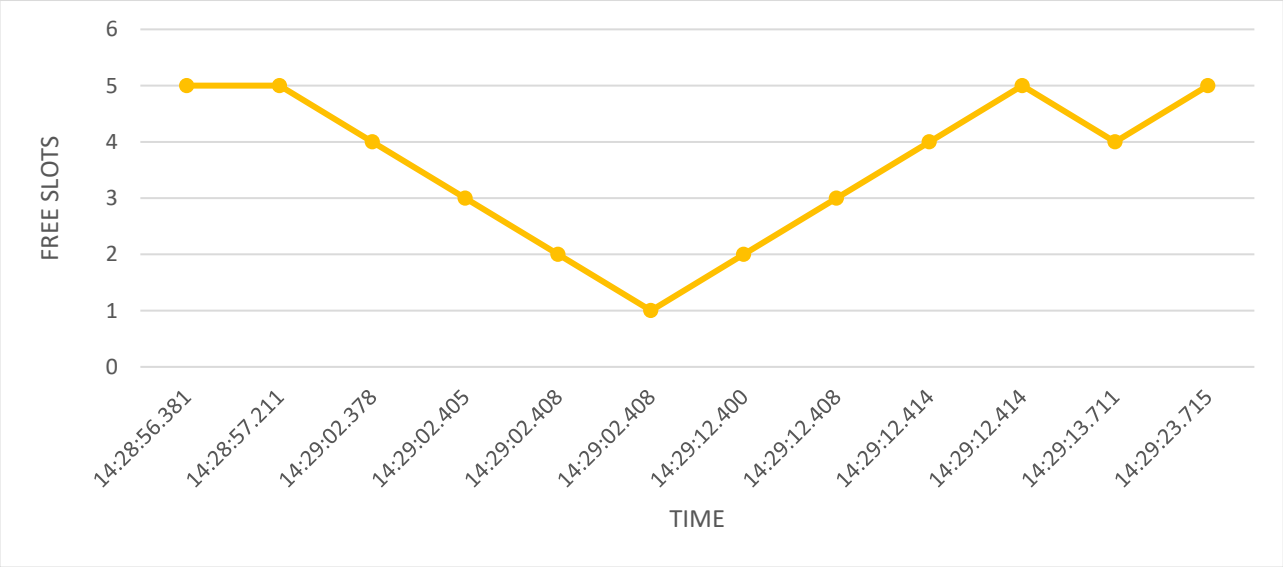




SUPPLIER 1 (server requests 0, 1, 2, 4, 10)

TIME	FREE SLOTS	STATUS
14:28:56,381	5	SLEEP
14:28:57,211	5	AWAKE
14:29:02,378	4	AWAKE
14:29:02,405	3	AWAKE
14:29:02,408	2	AWAKE
14:29:02,408	1	AWAKE
14:29:12,400	2	AWAKE
14:29:12,408	3	AWAKE
14:29:12,414	4	AWAKE
14:29:12,414	5	AWAKE
14:29:13,711	4	AWAKE
14:29:23,715	5	AWAKE

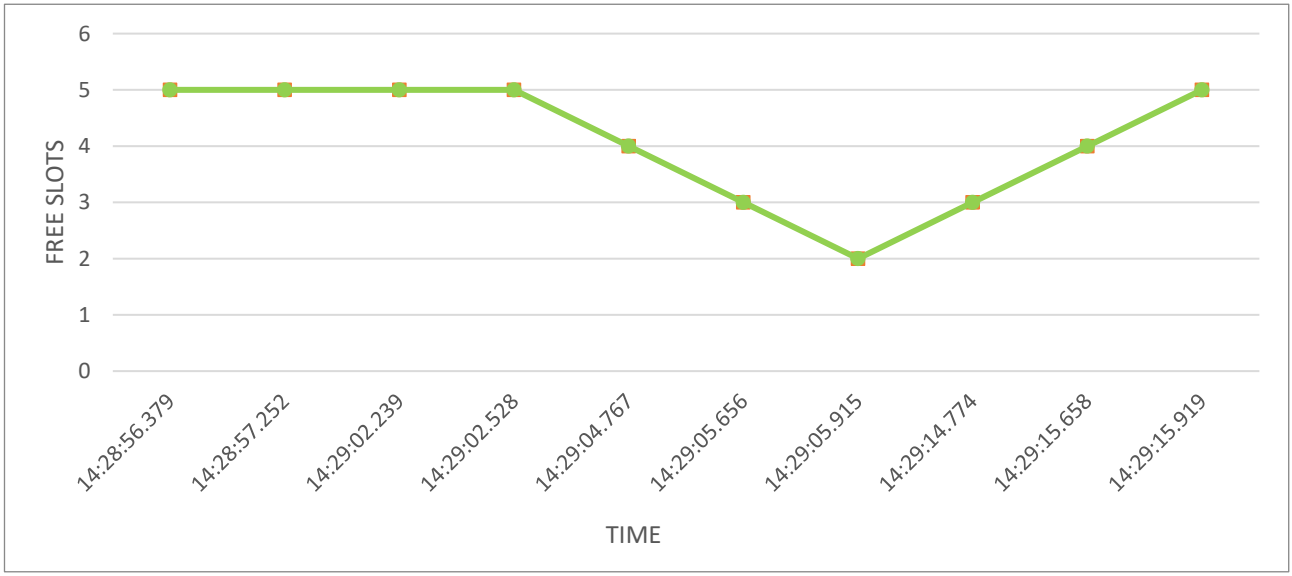
Average free slots: **4**



SUPPLIER 2 (server requests 3, 5, 6)

TIME	FREE SLOTS	STATUS
14:28:56,379	5	SLEEP
14:28:57,252	5	AWAKE
14:29:02,239	5	SLEEP
14:29:02,528	5	AWAKE
14:29:04,767	4	AWAKE
14:29:05,656	3	AWAKE
14:29:05,915	2	AWAKE
14:29:14,774	3	AWAKE
14:29:15,658	4	AWAKE
14:29:15,919	5	AWAKE

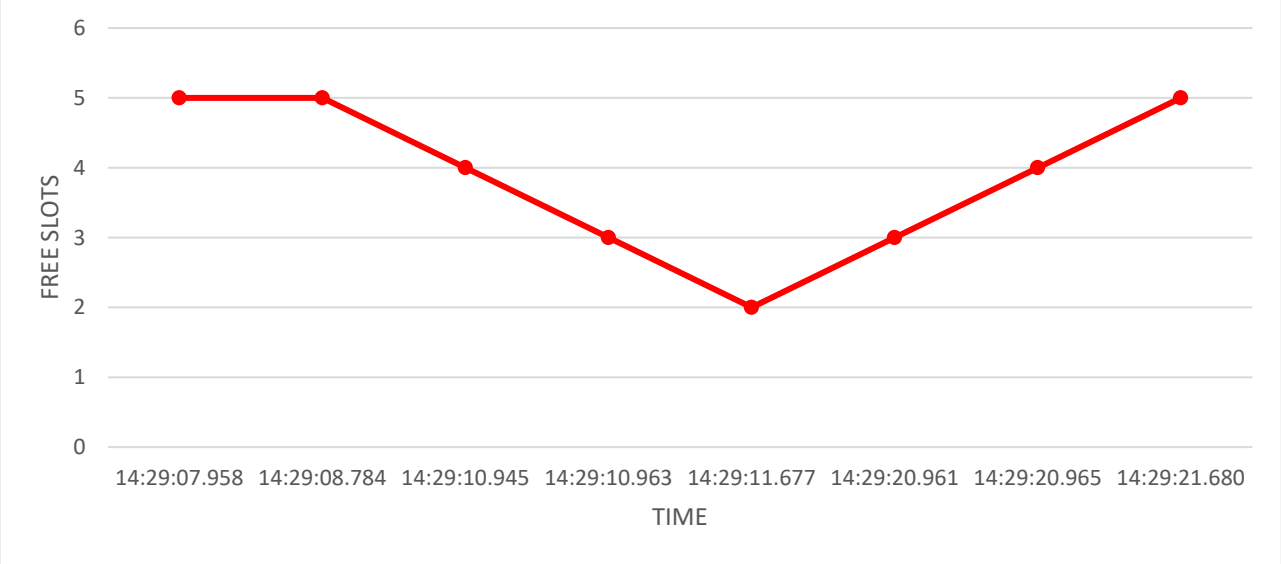
Average free slots: **4**



SUPPLIER 3 (server requests 7, 8, 9)

TIME	FREE SLOTS	STATUS
14:29:07,958	5	SLEEP
14:29:08,784	5	AWAKE
14:29:10,945	4	AWAKE
14:29:10,963	3	AWAKE
14:29:11,677	2	AWAKE
14:29:20,961	3	AWAKE
14:29:20,965	4	AWAKE
14:29:21,680	5	AWAKE

Average free slots: **4**

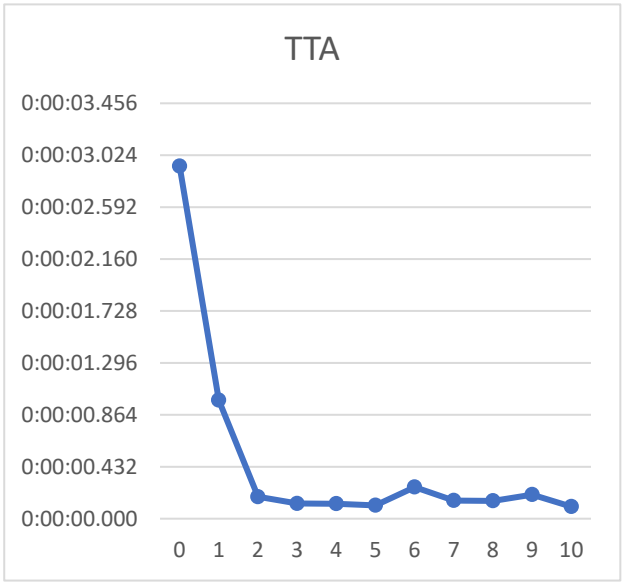


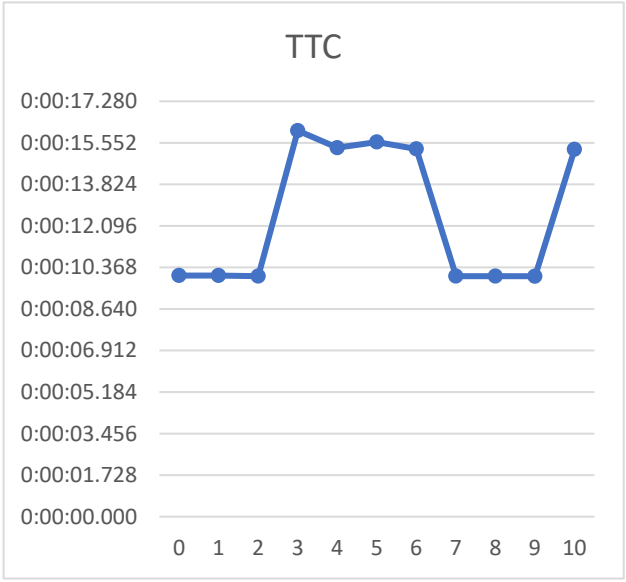
TEST 3

- supplier 1: *AWAKE*
- supplier 2: *OFFLINE*
- supplier 3: *OFFLINE*

ID	START TIME	ACCEPT TIME	END TIME	TTA	TTC
0	14:47:17,602	14:47:20,535	14:47:30,567	0:00:02,933	0:00:10,032
1	14:47:19,549	14:47:20,535	14:47:30,567	0:00:00,986	0:00:10,032
2	14:47:21,554	14:47:21,736	14:47:31,744	0:00:00,182	0:00:10,008
3	14:47:23,556	14:47:23,682	14:47:39,742	0:00:00,126	0:00:16,060
4	14:47:25,561	14:47:25,685	14:47:41,034	0:00:00,124	0:00:15,349
5	14:47:27,567	14:47:27,678	14:47:43,263	0:00:00,111	0:00:15,585
6	14:47:29,570	14:47:29,833	14:47:45,132	0:00:00,263	0:00:15,299
7	14:47:31,574	14:47:31,725	14:47:41,733	0:00:00,151	0:00:10,008
8	14:47:33,580	14:47:33,728	14:47:43,733	0:00:00,148	0:00:10,005
9	14:47:35,585	14:47:35,785	14:47:45,789	0:00:00,200	0:00:10,004
10	14:47:37,588	14:47:37,688	14:47:52,972	0:00:00,100	0:00:15,284

Average **TTA** (Time To Accept): **0.484s**
Average **TTC** (Time To Complete)**: 12.515s**





SUPPLIER 1 (serves requests 0, 1, 2, 7, 8, 9; the other requests have been sent to the *Vault)*

TIME	FREE SLOTS	STATUS
14:47:19,987	5	AWAKE
14:47:20,535	4	AWAKE
14:47:20,535	3	AWAKE
14:47:21,736	2	AWAKE
14:47:30,537	3	AWAKE
14:47:30,537	4	AWAKE
14:47:31,725	3	AWAKE
14:47:31,742	4	AWAKE
14:47:33,728	3	AWAKE
14:47:35,785	2	AWAKE
14:47:41,730	3	AWAKE
14:47:43,730	4	AWAKE
14:47:45,787	5	AWAKE

Average free slots: **3**