Performance Evaluation Report

Bagrin Radu
341C5

Parts of the assignment which were completed:

I. Prerequisites
II. Implementation
III. Evaluation
IV. Documentation

I consider my assignment should receive 90-95 points.

Testarea inițială

Pentru a testa câte requesturi pot fi primite de o singură mașină am scris următorul scriptul **test_initial_system.py**. Acesta face în parallel **nr_requests** cereri de GET, iar numărul maxim pentru care am reușit să primesc înapoi răspunsuri cu **status code = 200** a fost de **175**.

*Menționez că acesta este numărul pe care îl obțin în momentul redactării acestui document. Cu câteva zile înainte reușeam cu același cod să trimit uneori și 500 de requesturi simultane cu succes. Nu știu de ce în acest moment maximul este 175.

**Mai menționez și că dacă rulez scriptul de 2 ori consecutiv cu N = 175, a doua oară primesc următoarea excepție:
requests.exceptions.ConnectionError: ('Connection aborted.', RemoteDisconnected('Remote end closed connection without response'))
Pentru N=150 nu am această problemă, din acest motiv politicile mele vor fi testate pentru maxim 150 de requesturi trimise simultan.

	ASIA/0 – 175 requests	US/0 – 175 requests	EMEA – 175 requests
Total time	**4.583 seconds**	**4.330 seconds**	**4.103 seconds**
Average time / request	**0.026 seconds/request**	**0.024 seconds/request**	**0.023 seconds/request**

Latențele pentru fiecare regiune au fost extrase din răspunsurile requesturilor GET:
response time - work time și arată așa:

- **EMEA** – între 300 și 350 ms
- **US** – între 350 și 450 ms
- **ASIA** – între 500 și 600 ms

Computation time pentru un woker variază între **18** și **21 ms**

Response time pentru un worker variză între **300** și **600 ms**

Latența introdusă de **forwarding unit** este cuprinsă între 50 și 200 ms, în dependență de worker și am dedus-o aflând diferența de timp dintre requestul către *localhost:5000/worker* și adresa directă către acel worker.

Latența introdusă de **Heroku** am extras-o din logurile de pe aplicații și aceasta variază între **20** și **27 ms** atunci când nu există load și crește imediat cum crește și numărul de requesturi GET generate simultan. Pentru 1 request latența e de ~22ms, pentru 2 requesturi primul are latența de ~22 ms, iar al doilea de ~38ms, pentru 5 requesturi latența de la al doilea request variază între 35 și 80 de ms, pentru 10 requesturi latența depășește 100 de ms, pentru 20 de requesturi latența depășește și 200 de ms, iar de la 50 de requesturi în sus latența se stabilizează între 30 și 600 de ms.

Problema arhitecturii curente este că fiecare worker lucrează separat, utilizatorii pot chema direct workerul pe care vor să-l folosească și se poate întâmpla ca mai mulți utilizatori să utilizeze workerul X, în timp ce workerul Y să nu primească niciun request. Astfel există șanse ca workerul X să crape sau să răspundă foarte greu utilizatorilor.

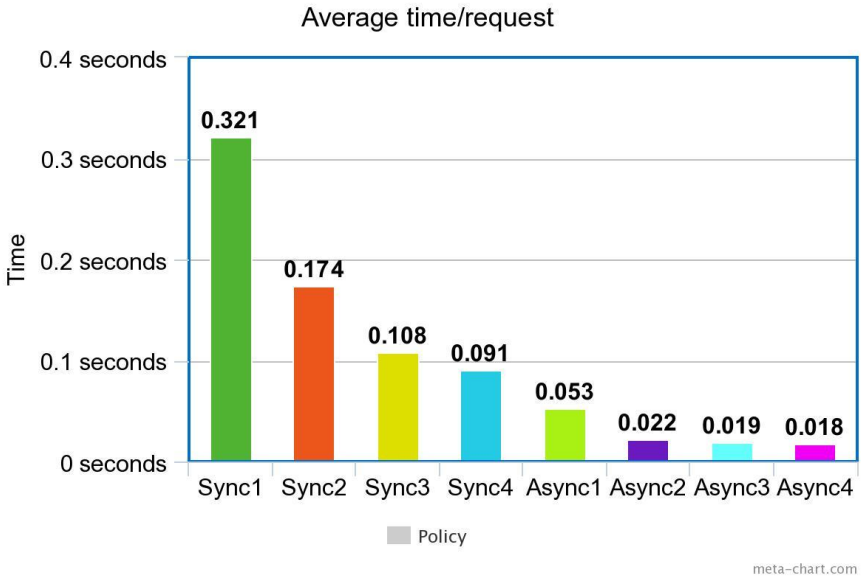
Implementare loadbalancer

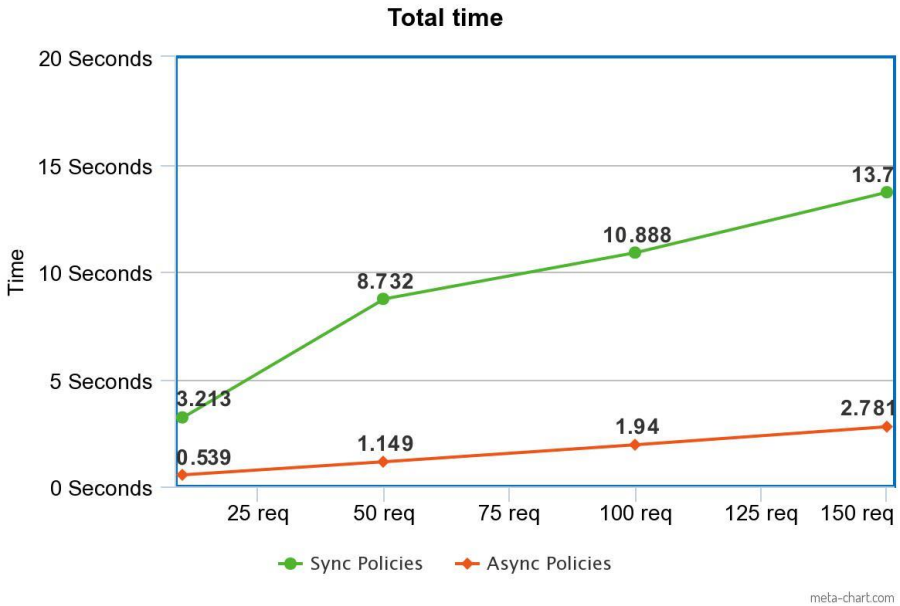
Am implemenat 2 tipuri de loadbalancere: sincron (**sync_loadbalancer.py**) și asincron (**async_loadbalancer.py**), fiecare utilizând câte 4 politici => în total 8.
O politică este aplicată în dependență de numărul primit ca input astfel:
dacă N <= 10 – aplică sync1/async1 altfel
dacă N <= 50 – aplică sync2/async2 altfel
dacă N <= 100 – aplică sync3/async3 altfel
aplică sync4/async4

Sync1/Async1 trimite sincron/asincron **toate** requesturile către workerul **emea**, acesta fiind cel mai rapid de accesat.
Sync2/Async2 trimite sincron/asincron **N/3** requesturi către **fiecare regiune**.
Sync3/Async3 trimite sincron/asincron **N/5** requesturi către **fiecare worker**.
Sync4/Async4 trimite sincron/asincron **N / 19 * X** requesturi către **fiecare worker**, unde pentru **us X=4**, pentru **asia X=3**, iar pentru **emea X=5**.

Evaluare performanță

Pentru a testa performanța am făcut o comparație a timpului mediu pentru un request pentru fiecare din cele 8 politici, iar rezultatul arată așa **=>**





De asemenea am făcut și o comparație sincron vs asincron folosindu-mă de timpul total necesar pentru trimiterea a 10/50/100/150 de requesturi.
<= Rezultatele finale arată astfel