FAKULTET INFORMATIKE
I DIGITALNIH TEHNOLOGIJA
Diplomski studij informatike

Seminarski rad iz kolegija

INFRASTRUKTURA ZA PODATKE VELIKOG OBUJMA

Horizontalno skaliranje pomoću posrednika za poruke: kašnjenje obrade ovisno o broju pretplatnika tijekom velikog opterećenja

Mentori: Dr. sc. Rok Piltaver
Tomislav Slaviček-Car, mag. inf.

Autor: Bojan Radulović

U Rijeci, Prosinac 2023.

Sadržaj

1. Uvod

U današnjem brzom i dinamičnom digitalnom okruženju, važnost skalabilnosti sustava postaje ključna za uspjeh organizacija. Skalabilnost označava sposobnost sustava da se prilagodi rastućem opterećenju ili promjenama u zahtjevima, čime se omogućuje gladak i učinkovit nastavak njegova rada. Ovaj koncept nije samo tehnički izazov, već i strateški imperativ koji utječe na konkurentske prednosti, inovaciju i dugoročnu održivost poduzeća. U ovom eksperimentalnom radu vidjet ćemo kako korištenje posrednika za poruke (eng. message broker) te dodavanje pretplatnika utječe na performanse sustava tijekom perioda velikog opterećenja te donijeti zaključak o učinkovitosti korištenja posrednika za poruke za horizontalno skaliranje sustava.

2. Posrednici za poruke

Posrednici za poruke su elementi u arhitekturi distribuiranih sustava koji olakšavaju komunikaciju između različitih dijelova sustava. Djeluju kao posrednici između različitih komponenti sustava, prenoseći poruke od jednog dijela sustava do drugog. Korištenje posrednika za poruke sa sobom donosi niz prednosti za skalabilnost i performanse sustava.

Posrednici za poruke omogućuju jednostavno dodavanje ili uklanjanje komponenti sustava bez narušavanja cjelokupne arhitekture te time pridonose horizontalnoj skalabilnosti sustava. Tijekom perioda velikog opterećenja na jednostavan način je moguće pokrenuti dodatne pretplatnike koji će preuzeti dio zahtjeva te samim time ublažiti opterećenje sustava. Također, posrednici poruka pridonose otpornosti sustava, jer mogu upravljati i redistribuirati opterećenje kako bi se izbjeglo preopterećenje ili kvarovi. Ukoliko svejedno dođe do kvara na nekom od pretplatnika, zahtjevi će biti distribuirani među preostalim pretplatnicima.

Međutim, potrebno je uzeti u obzir i potencijalne nedostatke, poput kompleksnosti upravljanja porukama i potrebe za dodatnom sigurnosnom zaštitom. Unatoč tim izazovima, posrednici za poruke pružaju mnoge prednosti u sustavima gdje je ključno postizanje učinkovite razmjene poruka i događaja između različitih komponenti. Primjene ovakvog sustava su raznovrsne, od mikroservisnih arhitektura, IoT sustava do velikih distribuiranih aplikacija poput sustava za razmjenu podataka u stvarnom vremenu ili praćenje prometa.

3. Arhitektura sustava

U svrhu mjerenja učinkovitosti korištenja posrednika za poruke te dodavanja pretplatnika za horizontalno skaliranje, izgrađen je jednostavan sustav koji simulira zapisivanje i čitanje podataka o događajima (npr. kao dio aplikacije za kalendare). Sustav se sastoji od četiri glavne komponente:

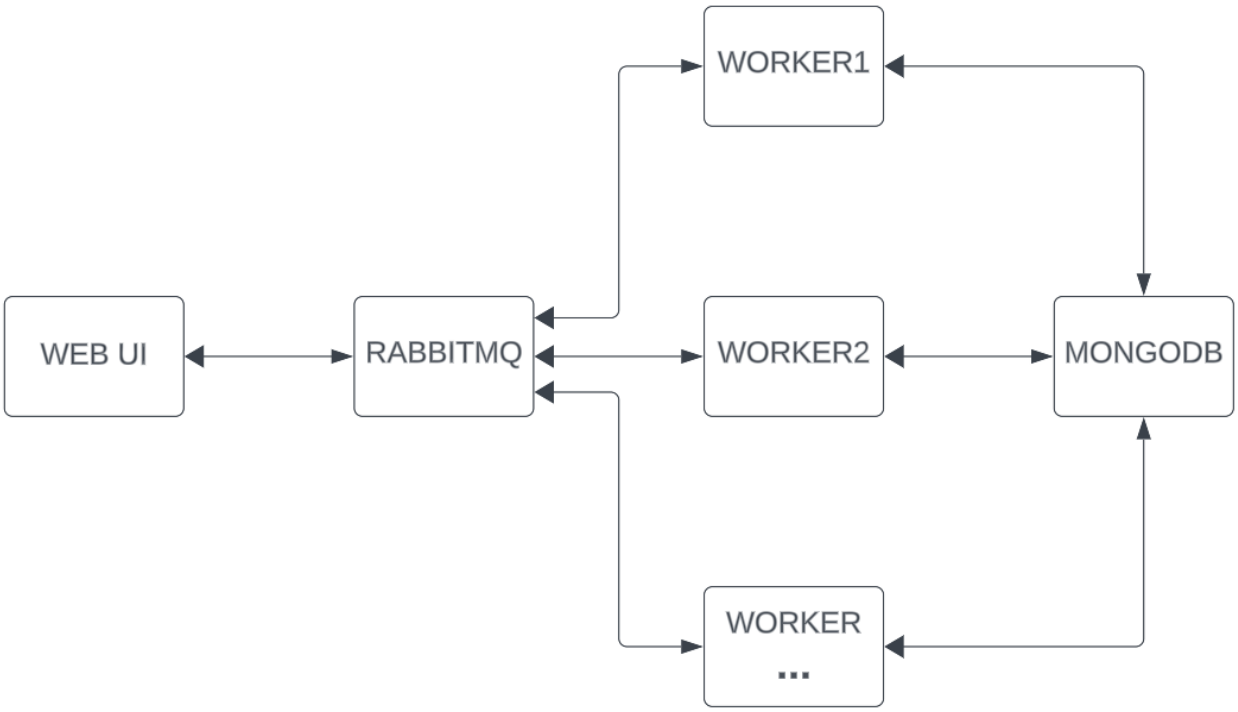
1) WEB UI – Flask server koji servira jednostavno korisničko sučelje za lakše testiranje funkcionalnosti aplikacije te pokretanje mjerenja performansi sustava (eng. benchmarking).

2) WORKER – Komponenta zadužena za simuliranje obrade podataka te zapisivanje i čitanje podataka u ili iz baze podataka. Sustav može istovremeno sadržavati više worker komponenti, a

upravo će učinak njihova broja na performanse sustava biti analiziran u ovom eksperimentalnom radu.

3) RABBITMQ – Posrednik za poruke. Ova komponenta je zadužena za prosljeđivanje poruka iz web ui komponente u neku od worker komponenti ili obratno.

4) MONGODB – Baza podataka. Ovdje worker komponente zapisuju obrađene podatke ili dohvaćaju prethodno obrađene podatke.



Slika 1, Arhitektura sustava.

Tok podataka u sustavu:

- Web UI šalje zahtjev za zapisivanje ili čitanje podataka na RabbitMQ red (eng. queue).
- Jedan od pretplatnika na red (neka od worker komponenti) obrađuje zahtjev (čita ili zapisuje podatke iz/u MongoDB bazu podataka).
- Worker komponenta šalje poruku sa zatraženim podatcima na pripadajući RabbitMQ red za odgovor (eng. response queue).
- Web UI dobiva odgovor te ga prikazuje korisniku.

Obrada podataka simulira se sleep naredbom podesive duljine, a nakon nje se obrađeni podatci zapisuju u MongoDB bazu podataka. Iako zapisivanje podataka nije nužno za svrhu mjerenja kašnjenja obrade ovisno o broju pretplatnika, pokazalo se korisno za validaciju izvršenih operacija. Npr. nakon izvršetka zahtjeva za zapisivanje podataka više puta (u svrhu mjerenja performansi), možemo zatražiti ispis podataka iz baze i uvjeriti se da su svi podatci uistinu zapisani te da se simulirana obrada podataka uspješno izvršila (broj ispisanih zapisa biti će jednak broju zahtjeva za zapisivanje podataka).

```python
def write_callback(ch, method, props, body):
    try:
        message_data = json.loads(body)

        user_id = message_data.get('user_id')
        date = message_data.get('date')
        event_text = message_data.get('event_text')
        sleep_time = message_data.get('sleep_time')

        time.sleep(sleep_time)

        event_document = {
            'user_id': user_id,
            'date': date,
            'event_text': event_text,
            'time_inserted': datetime.now().strftime('%d-%m-%y %H:%M:%S'),
        }

        db['events'].insert_one(event_document)

        print(f"Event data written to MongoDB: {event_document}")

        response = event_document
        response['_id'] = str(response['_id'])
        print("My response is ", response)
        ch.basic_publish(
            exchange='',
            routing_key=props.reply_to,
            properties=pika.BasicProperties(correlation_id=props.correlation_id),
            body=json.dumps(response)
        )
```

Slika 2, Dio programskog koda funkcije za zapisivanje podataka.

4. Korišteni podatci

Podatci korišteni za mjerenje utjecaja korištenja posrednika za poruke i dodavanja pretplatnika na performanse sustava generirani su pomoću Python biblioteke Faker. Generirani podatci predstavljaju podatke o događajima (npr. unutar aplikacije za kalendare) te sadrže sljedeće atribute:

- user_id – Cijeli broj (integer) koji predstavlja identifikacijski broj korisnika koji je stvorio događaj.
- date – Znakovni niz (string) koji predstavlja datum događaja.
- event_text – Znakovni niz (string) koji predstavlja naslov (ime) događaja.
- sleep_time – Decimalni broj (float) koji predstavlja vrijeme simulirane obrade podatka, odnosno koliko sekundi bi worker komponenta trebala provesti na simuliranje obrade ovog podatka (parametar za sleep funkciju). Ovaj podatak se ne upisuje u bazu, već samo služi za definiranje vremenskog trajanja simuliranja obrade podatka.
- time_inserted – Dodatan podatak koji se dodaju zapisu prilikom njegova zapisivanja u bazu podataka. Znakovni niz (string) koji predstavlja trenutak u vremenu kada je zapis zapisan u bazu podataka.
- _id – Instanca klase „ObjectId“ koja se koristi kao identifikacijska oznaka zapisa. Ovaj atribut se automatski generira prilikom zapisa događaja u MongoDB bazu podataka. Prilikom prikazivanja podataka u korisničkom sučelju, ovaj podatak se pretvara u znakovni niz.

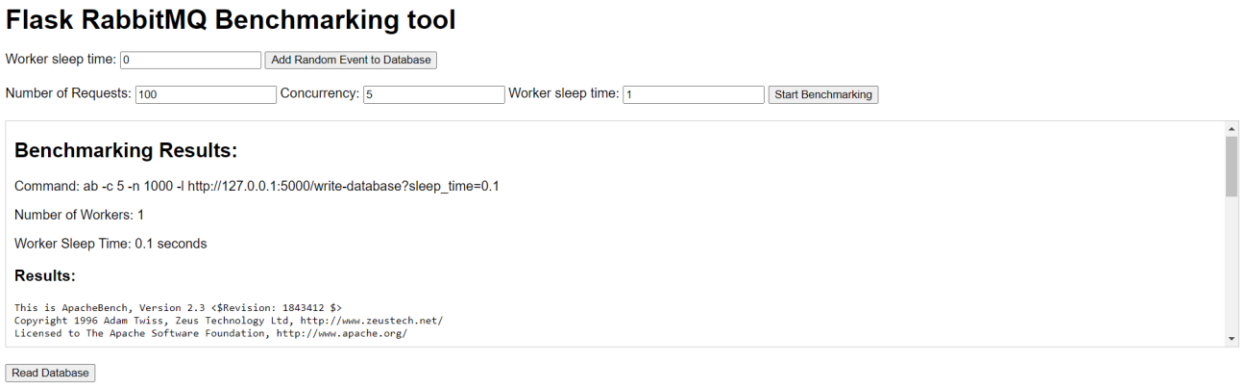
```
message = {
    'user_id': fake.random_int(min=1, max=1000),
    'date': fake.date_this_decade().strftime('%d-%m-%Y'),
    'event_text': fake.text(max_nb_chars=20),
    'sleep_time': sleep_time,
}
```

Slika 3, Primjer generiranja podatka pomoću Faker biblioteke.

Način generiranja podataka kod ovog eksperimentalnog rada nije naročito bitan jer se ne gledaju performanse zapisivanja i čitanja podataka u/iz baze već utjecaj dodavanja pretplatnika na kašnjenje kod obrade zahtjeva. Vrijeme potrebno za zapisivanje ili čitanje podataka u bazu je zanemarivo u usporedbi s vremenom potrošenim na simuliranje obrade podataka, odnosno vremenom izvođenja sleep funkcije. Podatci vezani uz događaje kod aplikacije za kalendare odabrani su samo kao primjer te su lako zamjenjivi s podatcima iz bilo koje domene tj. odabir podataka nema značajan utjecaj na test performansi. Čak iako se odaberu značajno opširniji podatci (npr. podatci s puno više atributa), relativan učinak dodavanja novih pretplatnika u sustav na performanse trebao bi ostati približno jednak.

5. Mjerenje performansi

Kako bi se olakšao proces mjerenja performansi (benchmarking), web UI komponenta sadrži jednostavno korisničko sučelje za pokretanje mjerenja performansi.



Slika 4, Korisničko sučelje aplikacije – mjerenje performansi.

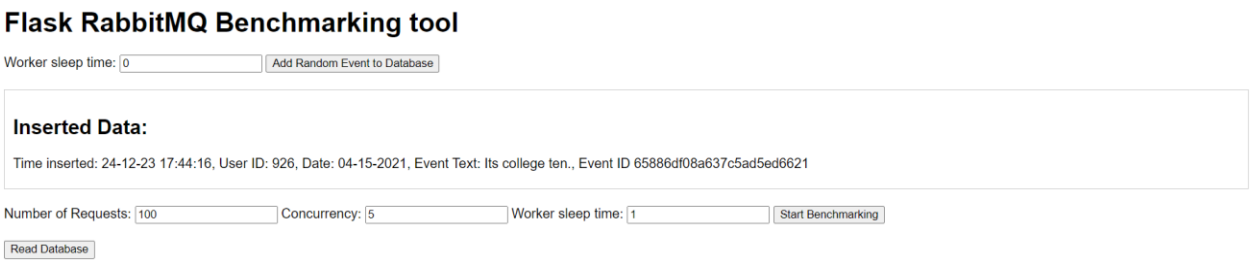
Za mjerenje performansi korištena je naredba

„ab -c <concurrency> -n <number of requests> -l http://127.0.0.1:5000/write-database?sleep_time=<worker sleep time>“

iz alata ab - Apache HTTP server benchmarking tool. Ova naredba šalje određen broj zahtjeva te računa vrijeme potrebno za pristizanje odgovora. Parametri naredbe koje korisnici mogu mijenjati putem sučelja su:

- Number of Requests – Ukupan broj zahtjeva koji će se poslati sustavu.
- Concurrency – Broj istovremenih zahtjeva.
- Worker sleep time – Vremensko trajanje simulirane obrade podataka (parametar funkcije sleep kod worker komponenti) u sekundama.

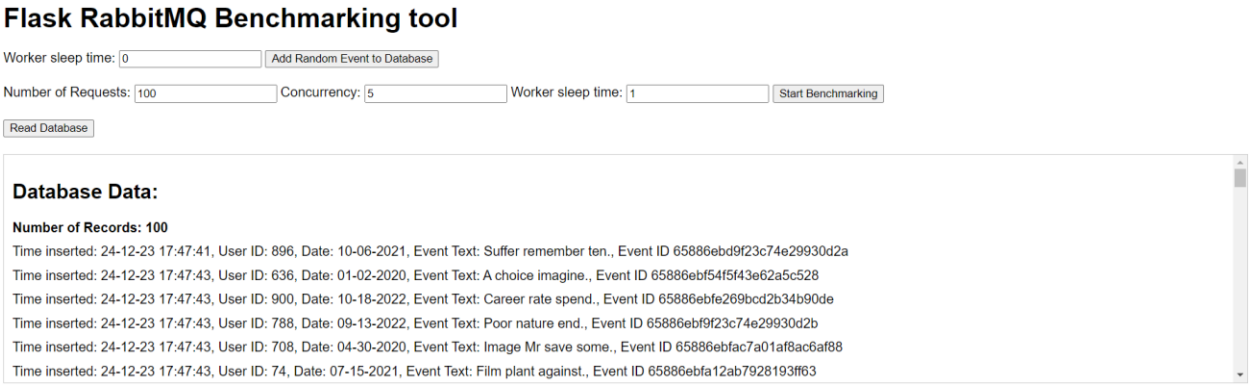
Korištena ruta „write-database“ pokreće proces stvaranja jednog novog zapisa u bazi podataka, s nasumičnim vrijednostima generiranim pomoću Faker biblioteke. Testiranje ove funkcionalnosti moguće je i pomoću gumba „Add Random Event to Database“ koji će dodati jedan zapis s nasumičnim podatcima u bazu podataka te ga prikazati u korisničkom sučelju.



Slika 5, Korisničko sučelje aplikacije – dodavanje nasumičnog zapisa.

Zastavica –l označava da se neočekivana duljina odgovora neće smatrati greškom prilikom mjerenja. To je bitno jer se podatci koje zapisujemo (pa samim time i odgovor) sastoje od nasumičnih vrijednosti pa bismo bez zastavice –l dobivali upozorenja o neuspjelim zahtjevima, čak i kada aplikacija ispravno radi.

Ispravan rad aplikacije možemo provjeriti i izlistavanjem zapisa iz MongoDB baze podataka pomoću gumba „Read Database“. Ako je broj zapisa u bazi jednak broju zahtjeva kojeg smo postavili tijekom mjerenja performansi, znači da su svi zahtjevi uspješno obrađeni.



Slika 6, Korisničko sučelje aplikacije – dohvaćanje zapisa.

Za svrhe ovog eksperimentalnog rada odabrani su sljedeći parametri:

- Number of Requests: 1000 requests.
- Concurrency: 5 requests.
- Worker sleep time: 0.1 seconds.

Ovi parametri su odabrani jer dobro prikazuju učinak korištenja posrednika za poruke na performanse sustava. Tisuću zapisa po pet istovremenih zapisa je dovoljno velik broj da uspješno pokaže poboljšanje performansi kod dodavanja pretplatnika koji mogu paralelno izvršavati istovremene zapise, dok ujedno nije prevelik broj zahtjeva koji bi potrošio resurse dostupne sustavu (resurse s PC-a na kojem se provodi

eksperiment). Iako su u ovom konkretnom eksperimentu korišteni ovi parametri, korisnici su ih dobrodošli mijenjati prema svojim željama.

Za rezultate mjerenja performansi je naravno bitan i korišten hardver. Neizbježno je da će isti sustav imati različite performanse ovisno o hardveru na kojem je pokrenut. U ovom eksperimentalnom radu korišten je PC sa sljedećim hardverom:

- CPU: AMD Ryzen 5 Mobile 3500U
- Matična ploča: Lenovo LNVNB161216
- Memorija: 12 GB DDR4
- Grafička kartica: AMD Radeon Vega 8 Graphics

Osim hardvera, bitan je i korišten softver, odnosno verzije softvera. Korišten softver je:

- Python 3.9.5
- Docker version 24.0.7, build afdd53b
- Razni Python paketi čije su verzije vidljive u requirements.txt datotekama unutar projekta

6. Korištenje

Kako bi se projekt uspješno pokrenuo, potrebno je pratiti sljedeće korake:

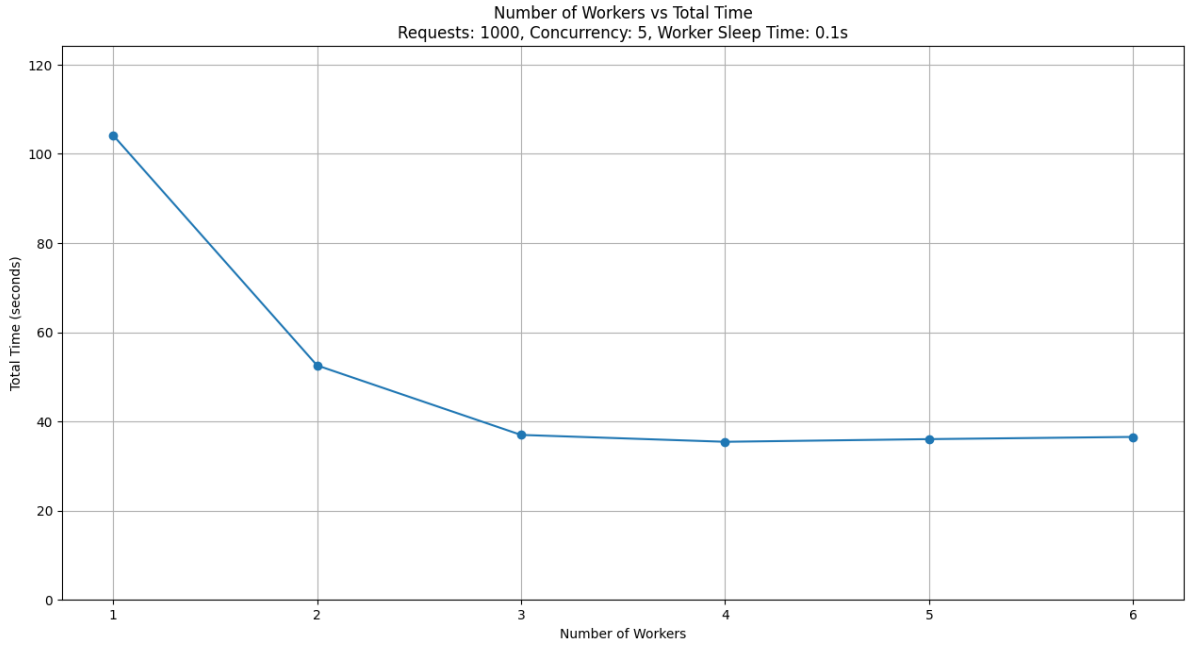
1) Klonirati projekt s GitHub-a na lokalno računalo.
2) Otvoriti terminal u korijenskom direktoriju projekta.
3) Pokrenuti naredbu „docker-compose up --scale app-worker=<N>“ gdje je <N> željeni broj worker komponenti (pretplatnika).
4) Pričekati da se sustav inicijalizira. Sustav je inicijaliziran kada rabbitmq komponenta izbaci poruku „Server startup complete“ i kada app-web komponenta javi da je server pokrenut (Running on http://127.0.0.1:5000).
5) Pristupiti serveru pomoću web preglednika na poveznici http://127.0.0.1:5000.
6) Pomoću sučelja pokrenuti mjerenje performansi sa željenim parametrima klikom na gumb „Start Benchmarking“.

7. Rezultati

Mjerenje performansi sustava izvedeno je šest puta, prvi put na sustavu s jednom worker komponentom pa na sustavu s dvije worker komponente pa s tri itd. zaključno sa sustavom sa šest worker komponenti. Rezultati mjerenja su sljedeći:

Broj pretplatnika (worker komponenti)	Vrijeme izvođenja mjerenja u sekundama	Broj obrađenih zahtjeva u sekundi (prosječna vrijednost)	Prosječno vrijeme za obradu jednog zahtjeva u milisekundama	Najdulje vrijeme za obradu zahtjeva u milisekundama
1	104.184	9.60	520.918	610
2	52.551	19.03	262.754	408
3	36.964	27.05	184.819	319
4	35.424	28.23	177.120	316
5	36.021	27.76	180.106	280
6	36.517	27.38	182.586	279

Na sljedećom grafikonu možemo vidjeti vizualizaciju ovih rezultata:



Slika 7, Grafikon koji prikazuje odnos broja pretplatnika (worker komponenti) i vremena potrebnog za izvršavanje mjerenja performansi.

Za sustave s jednim do tri pretplatnika možemo vidjeti gotovo linearan pad vremena izvođenja mjerenja s povećanjem broja pretplatnika (npr. kada se broj pretplatnika podupla s jednog pretplatnika na dva, vrijeme izvođenja testa je otprilike duplo manje itd.). Nakon sustava s tri pretplatnika uočavamo stagnaciju smanjenja vremena izvođenja testa, odnosno dodavanje dodanih pretplatnika nema značajan utjecaj na performanse sustava. Postoji više mogućih razloga za javljanje ovog fenomena:

- Točka saturacije – Moguće je da zbog ograničenja resursa hardvera, poput CPU-a, memorije ili propusnosti mreže, dodavanje dodatnih pretplatnika prestaje imati utjecaja na performanse.

- Zaključavanje i natjecanja – Moguće je da se povećanjem broja pretplatnika povećava i konkurentnost kod korištenja zajedničkih resursa (npr. baze podataka). Povećanje konkurentnosti može dovesti do natjecanja i problema sa zaključavanjem resursa, uzrokujući kašnjenja jer radnici konkuriraju za pristup.

- Neizbježno kašnjenje – Određenu razinu kašnjenja odgovora nemoguće je u potpunosti ukloniti. Ova vrsta kašnjenja javlja se zbog vremena potrebnog za simulaciju obrade podataka, prijenos podataka kroz različite komponente mreže, zapisivanje podataka u bazu itd.

Sljedeća slika prikazuje tablice koje pokazuje postotak zahtjeva koji su posluženi unutar određenog vremena za sustave s jednim, dva i tri pretplatnika (redom od lijeva na desno):

One worker

```
Percentage of the requests served
within a certain time (ms)
  50%    520
  66%    521
  75%    521
  80%    522
  90%    523
  95%    524
  98%    525
  99%    527
 100%    610 (longest request)
```

Two workers

```
Percentage of the requests served
within a certain time (ms)
  50%    261
  66%    263
  75%    265
  80%    266
  90%    271
  95%    279
  98%    290
  99%    295
 100%    408 (longest request)
```

Three workers

```
Percentage of the requests served
within a certain time (ms)
  50%    180
  66%    188
  75%    191
  80%    194
  90%    204
  95%    216
  98%    234
  99%    244
 100%    319 (longest request)
```

Slika 8, Postotci zahtjeva posluženi unutar određenog vremena za sustave s jednim, dva i tri pretplatnika.

Iz ovih podataka također možemo izvući isti zaključak: povećanjem broja pretplatnika vrijeme potrebno za obradu zahjteva opada.

Također, moramo imati na umu da postoje određene neizbježne varijacije u vremenu izvođenja mjerenja performansi. Čak i kada više puta izvršimo mjerenje s identičnim parametrima i na identičnom hardveru, neizbježno je da će se rezultati među mjerenjima najvjerojatnije razlikovati. Ova pojava se javlja zbog velikog niza faktora koji utječu na performanse sustava, a neki od njih su mogući pozadinski procesi na PC-u, mrežne fluktuacije, hardverske fluktuacije itd.

8. Zaključak

Rezultati eksperimenta pokazali su značajno poboljšanje performansi sustava kroz dodavanje pretplatnika tijekom povećanja opterećenja. Do određenog broja pretplatnika, sustav je pokazao gotovo linearno poboljšanje vremena izvođenja testa, što sugerira da horizontalno skaliranje ima pozitivan utjecaj na sustav. Korištenje posrednika za poruke, poput RabbitMQ, pokazalo se kao korisna strategija za olakšavanje komunikacije između različitih dijelova sustava. Posrednici za poruke omogućuju fleksibilnost u dodavanju i uklanjanju komponenti sustava te pridonose horizontalnoj skalabilnosti. Također, pridonose otpornosti sustava redistribucijom opterećenja i upravljanjem porukama tijekom razdoblja opterećenja. Unatoč pozitivnim rezultatima, važno je napomenuti da svaki sustav ima svoje specifičnosti i da je potrebno pažljivo prilagoditi strategije skaliranja prema konkretnim zahtjevima i karakteristikama sustava.

9. Popis slika

10. Literatura

1) Bojan, Radulović, „rabbitmq-benchmark“, https://github.com/Bojan-Radulovic/rabbitmq-benchmark (Pristupljeno: 24. prosinca 2023.)

2) RabbitMQ, „RabbitMQ“, https://www.rabbitmq.com/ (Pristupljeno: 24. prosinca 2023.)

3) MongoDB, „MongoDB“, https://www.mongodb.com/ (Pristupljeno: 24. prosinca 2023.)

4) Docker, „Docker“, https://www.docker.com/ (Pristupljeno: 24. prosinca 2023.)

5) Apache, „ab - Apache HTTP server benchmarking tool“, https://httpd.apache.org/docs/2.4/programs/ab.html (Pristupljeno: 24. prosinca 2023.)