Dokumentacija projekta

Load Balancer

Predmet:

Industrijski komunikacioni protokoli u EES

Članovi tima:

Aleksandar Stanković PR43/202
Milan Reljin PR 119/2021

Uvod

Opis problema koji se rešava

Ovaj dokument opisuje arhitekturu i funkcionisanje sistema za distribuciju i replikaciju podataka u okviru mrežnog okruženja koje uključuje Load Balancer, Workere i Replikator. Ključna komponenta sistema je Load Balancer, koji upravlja tokovima podataka između klijenata i dostupnih radnih jedinica (workera), kao i njihovom replikacijom radi očuvanja integriteta i dostupnosti podataka.

Ciljevi zadatka

Implementacija višenitnog sistem u kojem Load Balancer upravlja celokupnim tokom podataka. Sistem treba da omogući:

- Prijem podataka od klijenta I njihovo privremeno skladistenje
- Prikupljanje informacija o dostupnosti memorije svakog Workera
- Upis podataka kod Workera I njihovo prosledjivanje Replikatoru
- Replikacija podataka radi osiguravanja njihove konzistentnosti

Time se postiže stabilan, skalabilan i efikasan sistem koji može da funkcioniše u uslovima promenljivog opterećenja i paralelnih zahteva.

Dizajn i tehnička objašnjenja

Client

Komponenta koja predstavlja krajnjeg korisnika sistema i šalje poruke prema Load Balanceru.

Glavne odgovornosti:

- Uspostavljanje TCP konekcije sa Load Balancerom (`localhost:5059`)
- Validacija poruka pre slanja (npr. dužina, format)
- Slanje poruka Load Balanceru
- Prijem odgovora od Load Balancera
- Čuvanje svih poruka koje stigne od Load Balancera u fajl `clientOutput.txt`

Ključne funkcije:

- `sendMessages(SOCKET clientSocket)` – šalje poruke korisnički unesene ili generisane, sa upravljanjem greškama.

- `receiveMessages(SOCKET clientSocket)` – prima poruke od Load Balancera i prati potvrde (ACK/FAIL).

- `saveData(const char* buffer)` – sinhronizovano upisivanje primljenih poruka u fajl.

- `generateRandomMessage(int messageNum)` – kreira nasumičnu poruku za automatsko slanje.

Load Balancer

Središnja komponenta koja prima poruke od klijenata i distribuira ih među dostupnim radnicima (Worker) korišćenjem **Round Robin** algoritma.

Glavne odgovornosti:

- Prihvatanje konekcija od klijenata

- Održavanje liste aktivnih Worker instance, port 6060
- Slanje poruka radnicima u kružnom redosledu (Round Robin)
- Prijem odgovora od Workera i prosleđivanje nazad klijentima
- Praćenje odgovora Workera, izdvajanje ID poruke, slanje ACK ka klijentu i uklanjanje poruke iz radnikovog reda poruka.
- Redistribucija poruka u slučaju diskonektovanja Workera.
- Sinhronizacija više niti za prihvatanje klijenata i radnika, kao i za komunikaciju sa njima.

Ključne funkcije i niti:

- `clientListener(SOCKET clientListenSocket)` – nit koja prihvata nove konekcije od klijenata i pokreće nit za svaki novi klijent.
- `handleClient(SOCKET clientSocket)` – prima poruke od klijenta, stavlja ih u red i pokušava ih poslati odgovarajućem Workeru.
- `workerListener(SOCKET workerListenSocket)` – prihvata nove konekcije od Workera i kreira novu nit za njihovo praćenje.
- `handleWorkerResponse(Worker* worker)` – prima poruke od Workera, obrađuje potvrde, oslobadja zauzete resurse i redistribuira poruke u slučaju diskonektovanja.
- `selectWorker(Node* workers)` – bira radnika sa najmanje zauzetošću ili novog radnika za balansirano raspoređivanje.
- `sendDataToWorker(Worker* worker, Queue* clientMessages)` – pokušava poslati jednu poruku Workeru iz reda klijentskih poruka.
- `redistributeMessagesDead(Queue* clientMessages, Node* workers)` – u slučaju da je Worker pao, poruke se redistribuiraju na ostale radnike.

Worker

Komponenta koja vrši obradu poruka i prosleđuje rezultat Replikatoru. Svaki Worker koristi sopstveni red poruka i komunicira sa Replikatorom putem posebne TCP konekcije.

Glavne odgovornosti:

- Uspostavljanje TCP konekcije sa LoadBalancerom na portu 6060, prijem poruka od njega i upisivanje u red

- Paralelna obrada poruka iz reda i upisivanje u fajl `workerOutput.txt`
- Sinhronizacija pristupa redu poruka pomocu mutexa i uslovnih varijabli
- Slanje rezultata nazad Load Balanceru i Replikatoru

Ključne funkcije i niti:

- `receiveData(SOCKET workerSocket)` – nit za primanje poruka od Load Balancera; poruke deli po linijama i smešta u red.
- `processMessages(SOCKET workerSocket)` – nit koja čita poruke iz reda, upisuje ih u fajl, zatim šalje nazad Load Balanceru i Replikatoru.
- `saveData(const char* buffer)` – bezbedan upis u fajl koristeći Windows named mutex.
- `handleWorker(SOCKET workerSocket)` – orkestrira rad worker niti za primanje i obradu poruka.
- Glavna funkcija inicijalizuje konekcije, kreira red poruka, pokreće `handleWorker` i održava thread u petlji.

Replikator

Komponenta zadužena za arhiviranje poruka koje su prošle kroz Workere. Radi kao poseban server koji prihvata poruke od svih Workera.

Glavne odgovornosti:

- Otvaranje TCP server-socketa na portu 6061
- Prihvatanje poruka od više Workera
- Za svaku konekciju pokretanje zasebne niti za prijem poruka
- Čuvanje primljenih poruka u lokalni fajl (`replicatorOutput.txt`)
- Upis poruka u interni red i paralelna obrada (consumer nit)

Ključne funkcije i niti:

- `receiveAndStoreFromWorker()` – prijem i upis poruka u red
- `saveData()` – trajno čuvanje primljenih podataka
- Glavna f-ja pokrece server socket, prihvata konekciju i startuje niti za prijem podataka

Strukture Podataka

- **Red (Queue)**

Implementiran kao dinamički povezani niz elemenata (`Node` lista). Red je korišćen za privremeno skladištenje poruka u komponentama Worker i Replikator, omogućavajući **FIFO** (First In, First Out) pristup. Prednost je efikasna sinhronizacija višestrukih niti koje proizvode i konzumiraju poruke, čime se obezbeđuje uredan i pouzdan protok podataka.

- **Lista (Node lista)**

Koristi se kao osnovna struktura za implementaciju reda. Omogućava dinamičko dodavanje i uklanjanje elemenata bez potrebe za predefinisanim kapacitetom, što je važno za fleksibilnost i skalabilnost sistema.

• **Haš mapa za poruke (MessageMap)**

Struktura koja omogućava efikasno skladištenje i pristup porukama preko njihovih jedinstvenih ID-jeva. Omogućava brzo pretraživanje, ubacivanje i uklanjanje poruka, što je ključno za praćenje i upravljanje porukama koje su u obradi u okviru svakog Workera.

• **Struktura Worker**

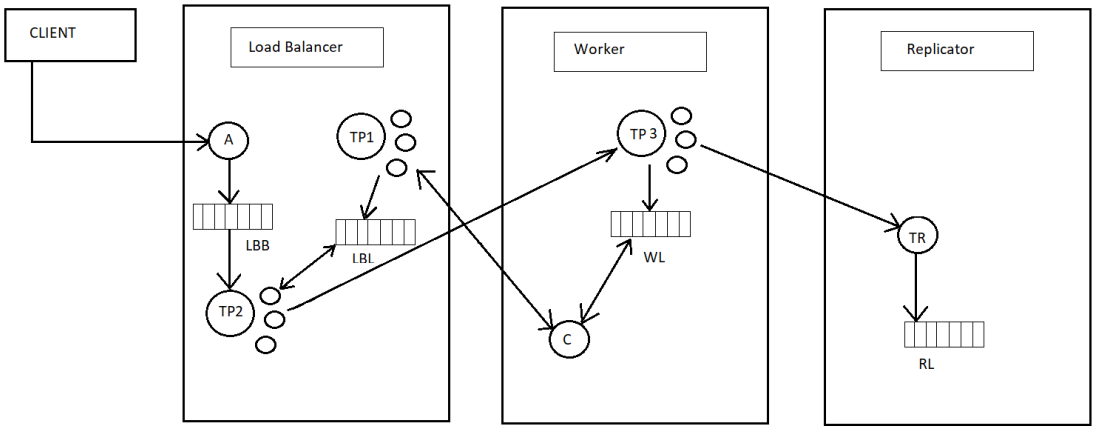
Predstavlja radnika koji obrađuje poruke i koristi mutex za zaštitu podataka u višedretvenom okruženju. Poseduje dinamički niz poruka i haš mapu za praćenje aktivnih poruka, čime se omogućava efikasno upravljanje njihovim životnim ciklusom.

Prednosti korišćenja ovih struktura:

- **Efikasna obrada i skladištenje poruka** — Korišćenjem reda i liste omogućena je brzo i sinhronizovano ubacivanje i uklanjanje poruka u višestrukim nitima bez gubitka podataka.
- **Sinhronizacija višestrukih niti** — Mutexi i condition_variable omogućavaju bezbednu komunikaciju između niti, sprečavajući race condition i deadlock.

- **Fleksibilnost i skalabilnost** — Dinamičke strukture bez fiksne veličine dozvoljavaju sistemu da prilagođava kapacitete prema potrebama u realnom vremenu.
- **Jasna organizacija podataka** — Struktura Message sa ID-jem olakšava praćenje i upravljanje porukama kroz ceo sistem.

Arhitektonski dijagram



Primer upotrebe

Klijent šalje jednu poruku Load Balanceru.

Load Balancer prosleđuje poruku Distributoru.

Distributor šalje poruku jednom od Workera (po Round Robin principu).

Worker obrađuje poruku, čuva je u fajlu i prosleđuje Replikatoru.

Replikator beleži poruku za sinhronizaciju.

Odgovor se vraća nazad Klijentu.

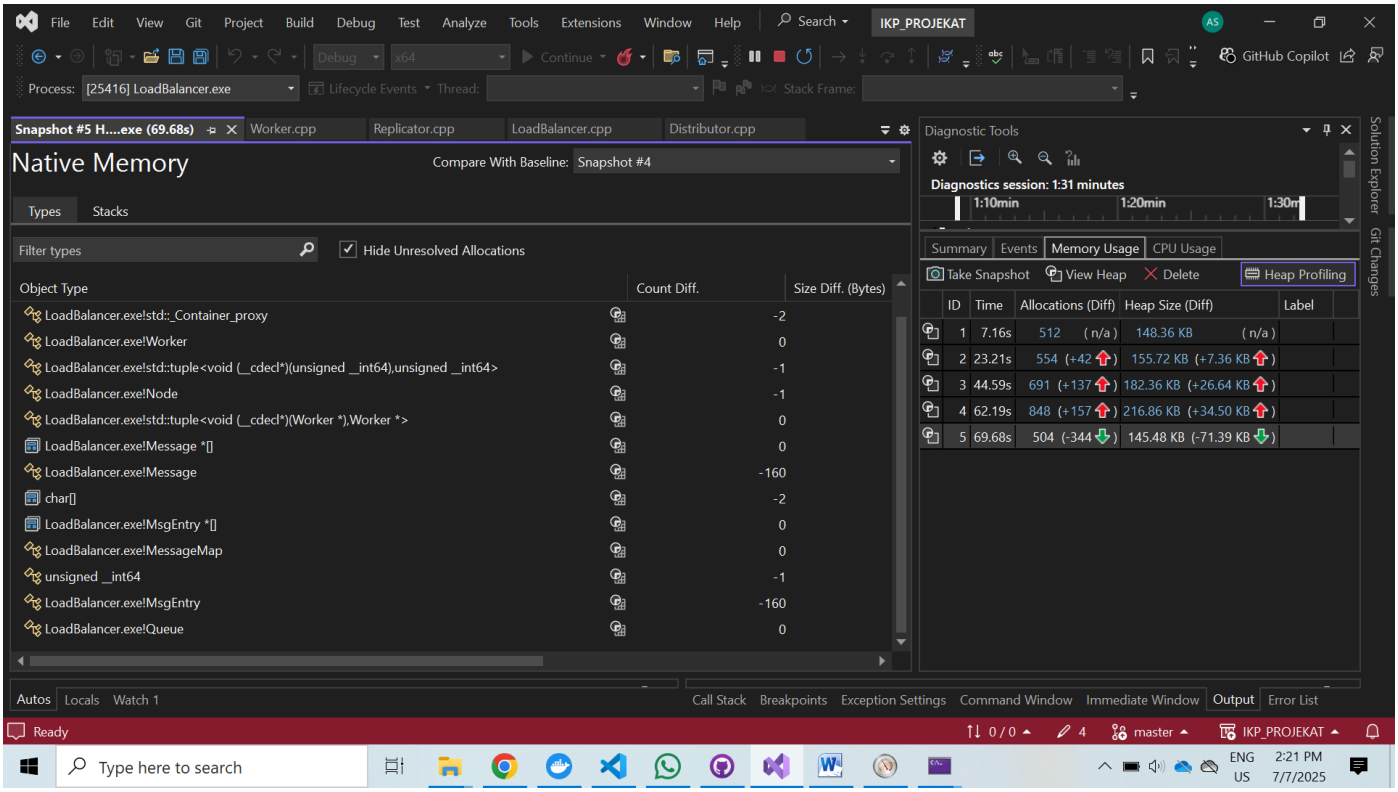
Rezultati Testiranja

Opis:

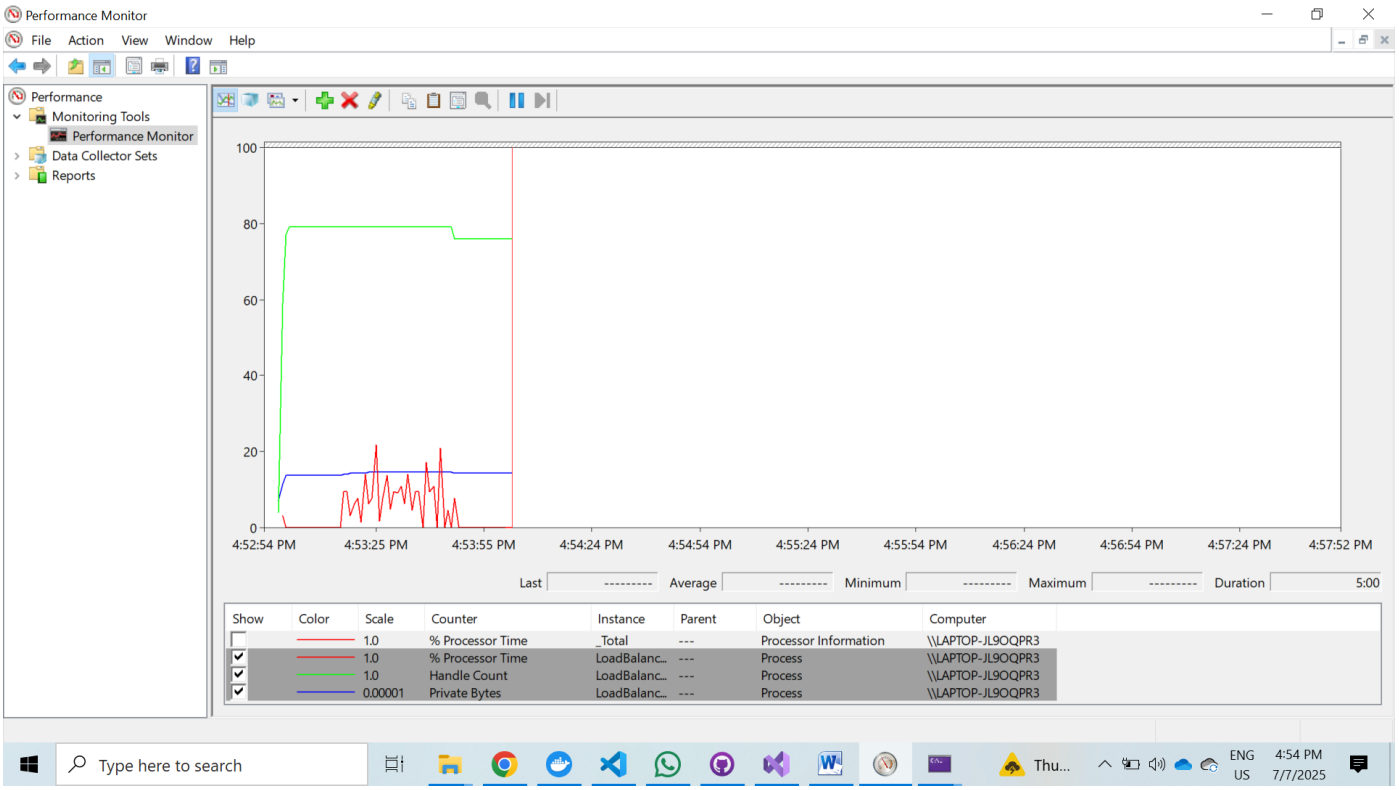
- Testiranje heap memorije, njene alokacije i čišćenja nakon završetka rada programa
- Performance counter testiranje

Prikazani rezultati testova:

Testiranje memorije:



Performance monitor:



Zaključak:

Rezultati testiranja memorije pokazuju da se skoro svi resursi koji su zauzeti nakon završetka rada programa očiste što znači da je curenje minimalno, odnosno da su rezultati dobri.

Potencijalna unapređenja

- **Otpornost na greške i ponovno slanje**

Dodati mehanizme potvrde prijema (acknowledgment) poruka i ponovno slanje poruka u slučaju gubitka veze ili greške, kako bi se povećala pouzdanost komunikacije.

- **Load Balancing na višem nivou**

Umesto statičkog Round Robin-a, koristiti inteligentnije algoritme za raspodelu opterećenja, npr. na osnovu trenutnog broja zauzetih zahteva kod Workera ili težine poruka.

- **Korišćenje modernih mrežnih protokola**

Prelazak sa TCP socket-a na protokole kao što su gRPC ili WebSocket za efikasniju i skalabilniju komunikaciju.

- **Sigurnost komunikacije**

Implementirati šifrovanje (TLS/SSL) za zaštitu podataka koji se šalju između komponenti, kao i autentifikaciju klijenata i radni