Benchmark pentru a testa rata de ieșire și timpul de răspuns al unui server Web.

Frăsineanu Claudia-Andreea

I.S. 1.1B

Cuprins

Introducere

În era digitală contemporană, serverele web constituie o componentă vitală a infrastructurii IT, facilitând accesul și gestionarea eficientă a resurselor informatice pe internet. Performanța acestor servere are un impact direct asupra experienței utilizatorilor și asupra eficienței operaționale a serviciilor online. Prin urmare, evaluarea și îmbunătățirea performanței serverelor web devine o prioritate strategică pentru orice organizație ce depinde de prezența online.

Unul dintre principalele instrumente utilizate în evaluarea performanței serverelor web este benchmarking-ul. Benchmarking-ul este un proces de măsurare a calității performanței unui sistem prin compararea acesteia cu un standard sau cu performanțele altor sisteme similare. În contextul serverelor web, principalele metrici de performanță care sunt frecvent evaluate includ rata de ieșire (throughput) și timpul de răspuns.

Rata de ieșire se referă la numărul de cereri pe care un server le poate procesa într-o unitate de timp, fiind un indicator al capacității de a gestiona volumul de trafic. Pe de altă parte, timpul de răspuns reprezintă intervalul de timp necesar pentru a răspunde la o cerere a utilizatorului, fiind un indicator al rapidității cu care serverul poate oferi răspunsuri la interogările primite.

Măsurarea acestor parametri și înțelegerea comportamentului serverului sub diferite sarcini de lucru sunt esențiale pentru optimizarea performanței și asigurarea unei experiențe utilizator satisfăcătoare. Acest proiect propune dezvoltarea și implementarea unui benchmark care să testeze rata de ieșire și timpul de răspuns al unui server web, oferind astfel o analiză detaliată a capabilităților acestuia în condiții variate de încărcare.

Prin urmare, scopul acestui proiect este nu doar să dezvolte un instrument de testare, ci și să ofere o perspectivă profundă asupra dinamicii performanței serverului, permițând identificarea punctelor slabe și formularea de recomandări pentru îmbunătățiri tehnice. Acest demers are potențialul de a contribui la literatura de specialitate prin oferirea de noi insight-uri și metodologii de testare, îmbunătățind astfel practicile actuale în domeniul IT.

Obiectivele proiectului

Scopul acestui proiect este de a dezvolta un benchmark pentru evaluarea performanței unui server web, concentrându-se în mod specific pe două aspecte critice: rata de ieșire și timpul de răspuns. Aceste metrici sunt esențiale pentru a determina eficiența cu care serverele web pot gestiona cererile utilizatorilor în condiții de trafic variabil. Prin urmare, obiectivele detaliate ale proiectului includ:

Măsurarea timpului de răspuns:

- *Definirea timpului de răspuns*: Timpul de răspuns va fi măsurat ca intervalul de timp dintre trimiterea unei cereri către server și primirea răspunsului de către client.
- *Evaluarea performanței sub sarcini variate*: Testarea serverului sub diferite nivele de cerere pentru a evalua stabilitatea și scalabilitatea acestuia în funcție de timpul de răspuns.

Evaluarea ratei de ieșire:

- *Definirea ratei de ieșire*: Rata de ieșire va fi măsurată ca numărul de cereri pe care serverul le poate procesa succesiv într-o unitate de timp.
- *Testarea sub diferite condiții de încărcare*: Realizarea de teste pentru a determina capacitatea maximă a serverului de a procesa cereri în diverse condiții de trafic.
- *Analiza impactului optimizărilor de performanță*: Investigarea efectelor pe care diferite ajustări hardware și software le pot avea asupra ratei de ieșire a serverului.

Dezvoltarea și implementarea benchmark-ului:

- *Proiectarea unui sistem modular*: Crearea unui cadru de lucru flexibil care permite adaptarea ușoară a testelor la diferite scenarii și configurații.
- *Implementarea automatizării testelor*: Utilizarea scripturilor și a utilitarelor automate pentru a facilita colectarea și analiza datelor, reducând astfel eroarea umană și sporind eficiența testării.
- *Integrarea analizei de date*: Implementarea unor metode de vizualizare și analiză statistică pentru a interpreta rezultatele și a trage concluzii valide.

Specificatii tehnice și instrumente utilizate

Python este ales drept limbaj de programare pentru proiectul de benchmarking al serverului web datorită caracteristicilor sale unice care îl fac extrem de adecvat pentru acest tip de aplicație. Următoarele detalii subliniază motivele pentru care Python este ideal pentru dezvoltarea și implementarea unui benchmark robust și eficient.

Python este renumit pentru sintaxa sa intuitivă și lizibilă, care reduce semnificativ curba de învățare pentru programatori și accelerează dezvoltarea. În contextul creării unui instrument de benchmarking, aceasta permite cercetătorilor și dezvoltatorilor să se concentreze pe logica testării performanței, mai degrabă decât pe complexitățile limbajului de programare. Python permite scrierea de scripturi compacte și eficiente, aspect crucial în

testarea și evaluarea performanței unui server web, unde timpul de execuție și acuratețea măsurătorilor sunt esențiale.

Unul dintre cele mai mari avantaje ale utilizării Python în proiecte tehnice este disponibilitatea unei game largi de biblioteci dezvoltate pentru a ușura sarcini complexe. Pentru acest proiect, biblioteci precum Requests pentru realizarea cererilor HTTP și Asyncio pentru gestionarea cererilor asincrone sunt vitale. Aceste biblioteci permit un control detaliat al sesiunilor HTTP, gestionarea excepțiilor și colectarea datelor într-un mod eficient, aspecte critice pentru măsurarea performanței unui server web. Mai mult, Pandas și Matplotlib oferă instrumente extinse pentru analiza și vizualizarea datelor, permițând utilizatorilor să genereze rapid rapoarte detaliate și vizualizări grafice ale datelor colectate, facilitând interpretarea rezultatelor benchmark-ului.

Python rulează pe diverse platforme - de la Windows și Linux până la macOS. Aceasta portabilitate asigură că benchmark-ul dezvoltat poate fi rulat în diferite medii fără modificări semnificative ale codului, un avantaj major în testarea performanței serverelor web care pot opera pe sisteme diferite.

De asemenea, Python facilitează integrarea cu alte servicii și aplicații, fie că este vorba de interfețe de programare a aplicațiilor (API-uri) externe sau de alte sisteme de gestionare a datelor. Acest lucru este esențial pentru un proiect de benchmarking, unde este adesea necesară colectarea datelor din multiple surse și integrarea cu alte instrumente de monitorizare și analiză.

Cu introducerea suportului pentru programare asincronă în Python 3.5 prin intermediul bibliotecii Asyncio, Python a devenit o opțiune viabilă pentru dezvoltarea de aplicații care necesită performanță înaltă în gestionarea simultană a multiple cereri, un aspect critic în benchmarking-ul serverelor web. Asincronismul permite gestionarea eficientă a mii de cereri în paralel, simulând condiții de trafic ridicat pe server și evaluând capacitatea acestuia de a procesa cereri sub presiune.

Biblioteci utilizate

Pentru dezvoltarea benchmark-ului serverului web, au fost folosite diverse biblioteci Python, fiecare având rolul său specific în funcționalitatea generală a aplicației. Aceste biblioteci nu doar că optimizează procesul de dezvoltare, dar aduc și funcționalități esențiale care permit o analiză detaliată și eficientă a performanței serverului. Următoarele sunt cele mai relevante biblioteci utilizate în proiectul nostru:

- `requests`

Biblioteca `requests` este una dintre cele mai populare biblioteci Python pentru efectuarea cererilor HTTP într-un mod simplu și intuitiv. A fost utilizată în proiect pentru a trimite cereri GET către serverul web testat. requests permite manipularea ușoară a headerelor, parametrilor și cookie-urilor cererii, oferind suport complet pentru sesiuni, ceea ce permite persistența anumitor parametri pe durata mai multor cereri.

Pentru scopurile benchmark-ului, utilizarea bibliotecii requests este esențială deoarece permite simularea unui comportament realist al utilizatorilor prin trimiterea de cereri HTTP către server și măsurarea timpului de răspuns, fără a necesita implementarea complexă a protocolului HTTP de la zero.

- `asyncio`

Asyncio este o bibliotecă pentru scrierea de cod concurent folosind sintaxa async/await introdusă în Python 3.5. Este ideală pentru scrierea de programe asincrone pentru a gestiona un număr mare de conexiuni în rețea. asyncio a fost folosită în proiectul nostru pentru a permite trimiterea asincronă a cererilor către server, ceea ce este crucial pentru simularea unui mediu sub sarcină intensă, unde multiple cereri trebuie gestionate în paralel.

Utilizarea asyncio în contextul nostru asigură că benchmark-ul poate măsura eficient cum se comportă serverul sub condiții de utilizare intensă, fără a bloca execuția programului în timp ce așteaptă răspunsuri de la server.

- `matplotlib`

`Matplotlib` este o bibliotecă de vizualizare a datelor în Python, utilizată pentru a genera o gamă largă de grafice statice, animate și interactive. Pentru acest proiect, matplotlib este folosită pentru a vizualiza distribuția timpilor de răspuns și alte metrici relevante prin histogramă, diagrame boxplot și grafice de linie. Aceasta permite o interpretare vizuală directă a performanței serverului, facilitând identificarea rapidă a problemelor potențiale și evaluarea eficienței acestuia.

Una dintre caracteristicile cheie ale Matplotlib este abilitatea sa de a genera o gamă largă de tipuri de grafice, cum ar fi graficele de linie, graficele de bare, histograma, diagramele circulare, diagramele de dispersie și multe altele. Aceste grafice pot fi personalizate în mod detaliat pentru a se potrivi nevoilor utilizatorilor, cu control asupra culorilor, stilurilor, etichetelor și multe altele. Este o resursă valoroasă pentru oricine

lucrează cu date în Python și caută să exploreze și să prezinte informațiile într-un mod vizual și accesibil.

- `numpy`

`Numpy` este o bibliotecă fundamentală pentru calculul numeric în Python. Oferă suport pentru array-uri multidimensionale mari și o colecție vastă de funcții matematice de înalt nivel pentru a opera pe aceste array-uri. Deși în scriptul prezentat nu este evidențiată direct utilizarea numpy, aceasta este adesea folosită în combinație cu pandas pentru manipulări de date mai complexe și pentru optimizarea performanțelor, datorită eficienței sale în calcule numerice.

O funcție cheie a bibliotecii `numpy` utilizată în proiect este capacitatea de a crea și manipula array-uri multidimensionale. Aceste array-uri permit stocarea și manipularea eficientă a datelor în mai multe dimensiuni, ceea ce este util în cazul datelor tabulare și a operațiilor matematice complexe. De exemplu, putem utiliza funcții numpy precum `numpy.array()` pentru a crea array-uri din datele noastre și `numpy.mean()` pentru a calcula media datelor.

În plus, `numpy` oferă funcții pentru efectuarea unei game largi de operații matematice, cum ar fi adunarea, scăderea, înmulțirea și împărțirea elementelor din array-uri. Aceste funcții sunt implementate în C, ceea ce asigură performanță și eficiență în calcul, lucru crucial în contextul măsurării performanței serverului web.

- `statistics`

Modulul `statistics` oferă funcții pentru calculul statisticilor matematice: medii, mediane, moduri și alte măsuri statistice. În cadrul aplicației noastre, statistics poate fi utilizat pentru a calcula medii și deviațiile standard ale timpilor de răspuns, oferind o înțelegere mai profundă a distribuției performanței serverului. Acest lucru este crucial pentru identificarea comportamentelor atipice sau a potențialelor deficiențe de performanță.

În cadrul proiectului, statistics este folosită pentru a calcula statistici cheie necesare în evaluarea performanței serverului web. Funcții precum `statistics.mean()` ne permit să calculăm timpul mediu de răspuns al serverului, iar `statistics.stdev()` este utilizată pentru a determina variația timpilor de răspuns, oferind o imagine despre consistența performanței serverului în diverse condiții de încărcare.

De asemenea, utilizarea bibliotecii statistics facilitează calculul robust și precis al acestor metrici, fără a necesita implementări suplimentare sau personalizate, asigurând că analiza datelor este atât eficientă cât și accesibilă.

- `argparse`

`Argparse` este o bibliotecă destinată gestionării argumentelor și opțiunilor de linie de comandă. Este folosită în proiect pentru a permite utilizatorului să specifice diferite configurări ale benchmark-ului, cum ar fi fișierul de configurație sau numărul de cereri, direct prin linia de comandă. Aceasta adaugă flexibilitate și ușurință în utilizarea și testarea diferitelor scenarii de încărcare pe server.

`Argparse` joacă un rol crucial în configurarea și personalizarea rulărilor testelor. Prin intermediul acestei biblioteci, proiectul nostru permite utilizatorilor să specifice parametrii esențiali, cum ar fi URL-ul serverului și numărul de cereri de test, direct prin argumente de linie de comandă. Aceasta crește flexibilitatea și scalabilitatea instrumentului de benchmark, oferind utilizatorilor posibilitatea de a ajusta teste în funcție de necesitățile specifice.

Utilizând argparse, definim argumente cum ar fi `--url` pentru a specifica adresa serverului care va fi testat și `--num_requests` pentru a indica numărul de solicitări pe care scriptul ar trebui să le trimită serverului. Acest lucru simplifică considerabil procesul de inițiere a unui test, făcându-l accesibil chiar și pentru utilizatorii care nu sunt familiarizați cu detaliile tehnice ale implementării.

- `PyQt5`

PyQt5 este o bibliotecă puternică pentru crearea de aplicații grafice (GUI) în Python. Este folosită în proiect pentru a dezvolta o interfață grafică user-friendly care permite utilizatorilor să interacționeze cu benchmark-ul fără a interacționa direct cu linia de comandă. Acest lucru face ca aplicația să fie mai accesibilă pentru utilizatorii fără experiență tehnică profundă și îmbunătățește experiența generală de utilizare prin oferirea unei interfețe vizuale intuitive.

PyQt5 joacă un rol esențial în interacțiunea utilizatorului cu instrumentul de benchmark, permițând configurarea testelor prin input-uri grafice și afișarea rezultatelor într-un mod vizual atractiv și ușor de interpretat. De exemplu, utilizatorii pot introduce URL-ul serverului, selecta numărul de cereri și vizualiza graficele de performanță generate direct din interfața grafică.

Utilizând PyQt5, am creat forme și controale, cum ar fi butoane, casete de text și diagrame, care sunt integrate într-o fereastră principală. Utilizatorii pot configura parametrii de testare printr-o serie de meniuri și widgets, apăsând butoane pentru a iniția teste și pentru a vizualiza rezultatele. Aceasta simplifică considerabil procesul de setare a unui test și face analiza datelor mai accesibilă și interactivă.

- `json`

Biblioteca `json` este folosită pentru manipularea datelor în format JSON, un format ușor și flexibil pentru schimbul de date. În proiectul nostru, json este esențială pentru încărcarea și salvarea configurațiilor testelor, cum ar fi URL-ul serverului și numărul de cereri, dintr-un fișier de configurație. Aceasta facilitează modificarea și adaptarea parametrilor de testare fără a necesita modificarea codului sursă, permițând un grad mai mare de flexibilitate în testare.

Utilizarea bibliotecii json este vitală pentru gestionarea configurărilor de testare. Prin aceasta, putem citi și interpreta fișierele de configurare care conțin parametri esențiali, cum ar fi URL-ul serverului web și numărul de cereri de test. Aceasta ne permite să personalizăm și să adaptăm testele fără a modifica codul sursă, oferind o flexibilitate crescută în utilizarea instrumentului.

De exemplu, în cadrul proiectului, funcția `load_config()` folosește json.load pentru a deschide și a citi un fișier de configurare, transformând conținutul său JSON într-un dicționar Python accesibil. Această abordare simplifică procesul de modificare a setărilor de test, permițând utilizatorilor să ajusteze parametrii precum adresa URL a serverului și volumul de cereri printr-un fișier de configurare extern, fără a necesita recompilarea sau modificări majore în cod.

- `datetime`

Biblioteca `datetime` este utilizată pentru manipularea datelor și timpului. În contextul proiectului de benchmark, datetime este folosită pentru a marca fiecare răspuns cu timestamp-ul exact al recepției. Aceasta permite analize temporale precise, cum ar fi calculul latențelor și evaluarea performanței serverului în diferite momente ale zilei.

`Datetime` este folosită pentru a marca timpul exact al fiecărui răspuns primit de la server. Acest lucru este crucial pentru a putea analiza și evalua performanța serverului în timp și pentru a detecta eventuale anomalii sau variații de performanță în funcție de momentul zilei sau de încărcătura serverului.

De exemplu, în funcția `send_request`, utilizăm `datetime.datetime.now().isoformat()` pentru a genera un șir de timp în format ISO 8601, care reprezintă momentul exact în care a fost primit răspunsul de la server. Acest timp este apoi înregistrat în fișierul de date pentru a putea fi ulterior analizat și vizualizat.

Arhitectura si proiectarea benchmark-ului

Arhitectura Sistemului

Sistemul propus pentru benchmark-ul serverului web este proiectat să fie simplu dar eficient, capabil să măsoare și să raporteze metrici precum rata de ieșire și timpul de răspuns în diferite condiții de încărcare. Structura acestui sistem este bazată pe o arhitectură client-server, unde clientul este un script Python care trimite cereri către serverul web targetat și colectează răspunsurile.

Componente Principale:

- *Clientul de Benchmark*: Implementat în Python, clientul gestionează generarea și trimiterea unui număr configurabil de cereri HTTP către server. Acesta utilizează biblioteca asyncio pentru a face cererile asincron, permițând un volum mare de cereri într-un timp scurt.
- *Serverul Web*: Ținta testelor, serverul web poate fi orice server configurabil accesibil prin URL. Nu este implementat în cadrul acestui proiect, dar interacțiunile cu acesta sunt esențiale pentru testare.

Decizii de Proiectare

Proiectul a fost ghidat de necesitatea de a fi ușor de utilizat și de capacitatea de a colecta date în mod eficient. De aceea, s-au luat următoarele decizii:

- *Utilizarea Python*: Alegerea Python ca limbaj de programare pentru implementarea clientului de benchmark a fost motivată de simplitatea și flexibilitatea limbajului, precum și de disponibilitatea extinsă a bibliotecilor pentru lucrul cu rețele și date.
- *Asincronitate*: Pentru a îmbunătăți performanța și a maximiza numărul de cereri trimise într-o perioadă scurtă de timp, s-a optat pentru o abordare asincronă a cererilor HTTP, care permite clientului să nu aștepte finalizarea fiecărei cereri înainte de a iniția următoarea.

Structura Proiectului

Proiectul de benchmarking este organizat în câteva componente principale, fiecare având un rol specific în funcționarea și analiza performanței serverului:

- `benchmark.py`: Acest fișier conține logica principală pentru executarea benchmark-ului. Este responsabil pentru inițierea cererilor HTTP asincrone către server, utilizând configurațiile specificate.
- `config_handler.py`: Gestionează încărcarea și salvarea configurațiilor testului din și în fișiere JSON. Acest modul facilitează personalizarea testelor, permițând utilizatorilor să seteze parametri precum URL-ul serverului și numărul de cereri.
- `data_plotting.py`: Acest modul utilizează biblioteci precum Pandas și Matplotlib pentru a procesa și vizualiza datele colectate. Funcțiile din acest fișier sunt folosite pentru a genera grafice și pentru a analiza distribuția timpilor de răspuns, throughput-ul și alte metrice de performanță.
- `test_scenarios.py`: Definește diferite scenarii de testare, cum ar fi teste de încărcare constantă, creștere treptată sau teste de vârf, pentru a evalua performanța serverului sub diverse condiții de încărcare.
- `main.py`: Punctul de intrare al aplicației, care coordonează întregul proces de benchmarking, de la citirea configurațiilor, inițierea testelor prin intermediul funcțiilor definite în benchmark.py și test_scenarios.py, până la salvarea și analiza rezultatelor.
- `response_data.csv`: Un fișier CSV unde sunt stocate răspunsurile primite de la server. Aceste date sunt ulterior utilizate pentru analiza performanței.

Fluxul de execuție al proiectului începe cu main.py, care citește configurațiile din config.json și coordonează lansarea cererilor HTTP către server. Răspunsurile sunt colectate și salvate în response_data.csv. Ulterior, data_plotting.py încarcă acest fișier CSV și efectuează diverse analize statistice și vizualizări ale datelor pentru a evalua performanța serverului.

Această structură modulară facilitează o separare clară a responsabilităților, îmbunătățind întreținerea și scalabilitatea codului. Analiza profundă și accesibilă a performanței permite identificarea rapidă a problemelor și a oportunităților de optimizare a serverului.

Implementare

Implementarea proiectului se bazează pe un set de funcționalități esențiale pentru efectuarea benchmark-ului serverului web. Codul sursă, scris în Python, utilizează o serie de biblioteci standard și externe pentru a efectua testele și a analiza rezultatele. Iată o prezentare detaliată a implementării:

Inițializarea și Configurarea

- `main.py`

Modulul main.py este componenta centrală a aplicației noastre de benchmarking, responsabilă pentru coordonarea întregului proces de testare. Acesta permite utilizatorilor să configureze dinamic testele de benchmark, oferind flexibilitate crescută și adaptabilitate la diverse scenarii de testare.

Utilizatorii pot ajusta parametrii și opțiunile testului direct prin intermediul interfeței grafice dezvoltate cu PyQt5, fără a fi nevoie să modifice codul sursă al benchmark-ului. Aceasta oferă o flexibilitate crescută și o adaptabilitate la diferitele scenarii și medii de testare.

Configurarea Dinamică a Testelor

- URL și Număr de Cereri: Utilizatorii pot introduce dinamic URL-ul serverului și numărul de cereri în interfața grafică, facilitând testarea diferitelor servere și încărcături fără a schimba fișierul de configurare.
- Selecia Tipului de Test: Tipul de test (de exemplu, test constant, ramp-up, spike) poate fi selectat dintr-un meniu drop-down, permițând utilizatorilor să customizeze modul în care sunt trimise cererile către server.

Funcționalități Cheie

- Deschiderea Fișierului: Se utilizează with open() pentru a deschide fișierul de configurare în modul citire ('r'), asigurând închiderea automată după utilizare.
- Încărcarea Configurațiilor: Folosește json.load() pentru a încărca conținutul fișierului JSON într-un dicționar Python, care include URL-ul serverului, numărul de cereri, metoda HTTP, antetele, datele de încărcare și nivelul de concurență.

Trimiterea Cererilor către Server

- ❖ `send_request()`

Această funcție este responsabilă pentru trimiterea unei singure cereri către serverul web țintă. Utilizează biblioteca asyncio pentru a efectua cererea asincron, pentru a nu bloca execuția programului în timp ce așteaptă un răspuns.

- ➔ Argumentul `url`: Funcția primește ca argument adresa URL a serverului web către care se trimite cererea.
- ➔ Măsurarea Timpului de Răspuns: Inițializează un cronometru înainte de trimiterea cererii și îl oprește după primirea răspunsului, pentru a măsura timpul total de răspuns al serverului.
- ➔ Trimiterea Cererii:
  - ◆ Utilizează funcția `asyncio.to_thread()` pentru a trimite cererea către server într-un thread separat, astfel încât să nu blocheze bucla evenimentelor principale.
  - ◆ Folosește metoda `requests.get()` pentru a trimite o cerere de tip GET către URL-ul specificat.
- ➔ Verifică codul de stare al răspunsului pentru a detecta eventuale erori HTTP. Dacă codul de stare este în intervalul 4XX sau 5XX, este generată o excepție folosind metoda `raise_for_status()` a obiectului de răspuns.
- ➔ După primirea răspunsului sau în cazul unei erori, înregistrează timestamp-ul, timpul de răspuns, codul de stare și mesajul de eroare (dacă există) într-un dicționar.
- ➔ Returnează dicționarul cu informațiile despre răspuns sub forma unui obiect `ResponseData`, care conține timestamp-ul, timpul de răspuns, codul de stare și mesajul de eroare.

Prin intermediul acestei funcții, se realizează trimiterea și gestionarea cererilor către serverul web țintă, colectându-se datele de răspuns pentru analiza ulterioară. Este un element fundamental în procesul de benchmarking și în obținerea metricilor relevante despre performanța serverului web.

- ❖ `send_requests()`

Această funcție inițiază un număr specificat de cereri către serverul web, folosind funcția `send_requests` pentru a trimite fiecare cerere. Folosește asyncio.gather pentru a executa cererile în paralel și pentru a aștepta finalizarea tuturor acestora înainte de a continua.

- ➔ Argumentele Funcției:
  - ◆ `url`: URL-ul serverului către care se trimit cererile.
  - ◆ `num_requests`: Numărul total de cereri de trimis.
  - ◆ `concurrency`: Numărul maxim de cereri care pot fi trimise simultan.
- ➔ Implementarea Concurenței:
  - ◆ Funcția folosește un semafor (`asyncio.Semaphore`) pentru a limita numărul de cereri HTTP care pot fi executate simultan. Acest mecanism previne supraîncărcarea serverului și a clientului, menținând performanța în limite optimale.
  - ◆ `semaphore` este setat cu valoarea de concurență, care controlează câte task-uri (cereri) pot rula în paralel.
- ➔ Crearea Task-urilor Asincrone:
  - ◆ O listă comprehensivă de task-uri este creată, unde fiecare task este o invocare a funcției `send_request_limited`. Funcția `send_request_limited` gestionează accesul la semafor și apoi apelează send_request.
  - ◆ Se folosește înțelegerea de listă pentru a crea un task pentru fiecare cerere din numărul total de cereri specificat.
- ➔ Colectarea Răspunsurilor:
  - ◆ Funcția `asyncio.gather` este folosită pentru a rula toate task-urile în paralel și pentru a colecta răspunsurile. Aceasta așteaptă finalizarea tuturor task-urilor, adunând rezultatele într-o listă de răspunsuri.
- ➔ Returnarea Răspunsurilor:
  - ◆ După finalizarea tuturor cererilor, lista de răspunsuri este returnată. Aceasta conține răspunsuri de la fiecare cerere trimisă, inclusiv timpul de răspuns, codul de stare, și orice mesaje de eroare.

Prin această funcție, se pot simula scenarii reale de utilizare, verificându-se modul în care serverul gestionează multiple cereri concurente și colectând date valoroase pentru optimizare și scalare.

Scenarii de test

Modulul test_scenarios.py gestionează diverse scenarii de testare pentru evaluarea performanței serverului sub diferite condiții de încărcare. Acesta utilizează funcții asincrone pentru a simula condiții reale de trafic și pentru a colecta datele necesare analizei performanței.

- ❖ `run_constant_load_test()`

- ➢ Descriere: Simulează o încărcare constantă pe server, trimițând un număr fix de cereri.
- ➢ Apel Funcție: Această funcție apelează send_requests din modulul benchmark, specificând un număr fix de cereri, fără ramp-up time.
- ➢ Returnare Rezultate: Returnează rezultatele cererilor, care includ timpul de răspuns și alte date relevante colectate de la server.

❖ `run_ramp_up_test()`

- ➢ Descriere: Crește treptat încărcarea pe server pe parcursul unui interval de timp specificat.
- ➢ Apel Funcție: Trimite cereri incremental pe durata specificată de ramp_up_time, creșterea numărului de cereri fiind distribuită pe întreg intervalul.
- ➢ Returnare Rezultate: Colectează și returnează datele de la fiecare cerere pentru analiză ulterioară.

❖ `run_spike_test()`

- ➢ Descriere: Creează vârfuri de încărcare bruscă, urmate de perioade de repaus, pentru a simula accesul sporadic intens la server.
- ➢ Cicluri de Test: Execută un număr specificat de cicluri de vârfuri, trimițând un număr mare de cereri simultan, urmate de pauze.
- ➢ Colectare și Agregare Date: Datele de la fiecare vârf sunt colectate și agregate într-o listă de rezultate.

❖ `run_stress_test()`

- ➢ Descriere: Incrementează încărcarea până când serverul eșuează consistent sau atinge numărul maxim de cereri specificat.
- ➢ Bucle de Test: Începe cu un număr mic de cereri și crește treptat încărcarea. Monitorizează rata de eroare și oprește testul dacă eroarea depășește un prag specificat.
- ➢ Returnare Rezultate: Rezultatele fiecărui ciclu de test sunt stocate pentru analiză, oferind o imagine detaliată a comportamentului serverului sub stres.

Argumente Funcții

- ❖ url: URL-ul serverului ce urmează să fie testat.
- ❖ num_requests, ramp_up_time, peak_load, num_spikes, rest_period, start_requests, max_requests, step, threshold_errors, method, data: Parametri configurabili care permit adaptarea testului la specificațiile necesare.

Prin folosirea acestor scenarii de test, test_scenarios.py permite o evaluare detaliată a capacității serverului de a răspunde sub diverse condiții de trafic. Aceste teste ajută la identificarea limitelor de performanță ale serverului și la optimizarea configurațiilor pentru a asigura o funcționare optimă în condiții reale.

Colectarea și Salvarea Datelor

- `store_responses_to_csv()`

Această funcție primește lista de rezultate ale cererilor și le salvează într-un fișier CSV (`response_data.csv`). Fiecare răspuns este salvat ca o înregistrare în fișier, cuprinzând timestamp-ul, timpul de răspuns, codul de stare și mesajul de eroare (dacă există).

- Argumentul Funcției:
  - `results`: Lista de dicționare care conține datele de răspuns pentru fiecare cerere efectuată în cadrul testului de benchmark.
- Deschiderea Fișierului CSV:
  - Folosind declarația with `open()`, fișierul CSV `response_data.csv` este deschis în modul de scriere ('w'). Specificatorul `newline=''` este folosit pentru a evita adăugarea liniilor suplimentare între rândurile scrise în fișier.
- Inițializarea Scriitorului CSV:
  - Inițializează un obiect `DictWriter` pentru a scrie datele în format CSV. Se specifică numele câmpurilor (coloanelor) folosind lista fieldnames.
  - Se apelează metoda `writeheader()` pentru a scrie rândul de antet în fișierul CSV, folosind numele câmpurilor specificate.
- Scrierea Rândurilor de Date:
  - Pentru fiecare dicționar din lista results, se utilizează metoda `writerow()` pentru a scrie un rând de date în fișierul CSV. Fiecare dicționar reprezintă datele de răspuns pentru o cerere, iar cheile dicționarului corespund câmpurilor din antetul fișierului CSV.

Prin apelul acestei funcții, rezultatele testului sunt salvate în fișierul `response_data.csv`, permițând ulterior analiza și interpretarea datelor colectate în cadrul testului de benchmark.

Analiza și Vizualizarea Datelor:

Aceasta are loc in `fisierul data_analysis.ipynb`, care conține analiza performanței unui server web prin intermediul unui set de date colectate în timpul unui test de benchmark. Testul a fost realizat pentru a evalua și monitoriza performanța serverului în timpul gestionării unui volum mare de cereri HTTP. Analiza se concentrează pe diferite

aspecte ale performanței, inclusiv timpul de răspuns al serverului, throughput-ul și rata de eroare.

- ➔ Histograma Distribuției Timpului de Răspuns:
  - ◆ Primul grafic este un histogramă care prezintă *distribuția timpului de răspuns al serverului web*. Acesta oferă o vedere de ansamblu asupra frecvenței diferitelor intervale de timp de răspuns, permițând identificarea comportamentelor obișnuite și a posibilelor anomalii.
- ➔ Graficul Procentelor Timpului de Răspuns:
  - ◆ Urmează un grafic cu liniile care reprezintă percentilele timpului de răspuns în funcție de timp. Acesta oferă o perspectivă asupra timpului de răspuns în diferitele procente ale setului de date, permițând identificarea tendințelor de performanță.
- ➔ Graficul Throughput-ului în Timp Real:
  - ◆ Acest grafic arată throughput-ul (numărul de cereri pe secundă) în funcție de timp. El oferă o vedere asupra volumului de trafic care a fost procesat de serverul web în timpul testului, indicând capacitățile sale de procesare a cererilor.
- ➔ Graficul Ratei de Eroare în Timp Real:
  - ◆ Ultimul grafic prezintă rata de eroare (în procente) în funcție de timp. Acesta evidențiază proporția de cereri care au eșuat sau au întâmpinat probleme, oferind informații despre stabilitatea și fiabilitatea serverului web.

Analiza oferă o perspectivă comprehensivă asupra performanței serverului web și furnizează date și insights utile pentru evaluarea și optimizarea continuă a performanței acestuia în mediul său de utilizare. Prin interpretarea acestor grafice și a datelor asociate, se pot identifica punctele forte și punctele slabe ale serverului web și se pot lua măsuri corespunzătoare pentru a îmbunătăți performanța și fiabilitatea acestuia.

Rezultate

Testul de încărcare constantă

Testul de încărcare constantă este conceput pentru a evalua capacitatea serverului de a gestiona un flux stabil și continuu de cereri la o rată fixă. Acest tip de test este crucial pentru a înțelege performanța de bază a serverului în condiții de trafic uniform.

Obiectivele Testului

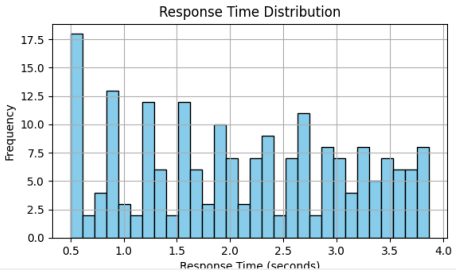
- ❖ Consistența Răspunsurilor: Verificarea dacă serverul poate menține un nivel constant de performanță pe durata testului.
- ❖ Timpul de Răspuns: Măsurarea timpilor de răspuns pentru a determina dacă există întârzieri sau variații majore.

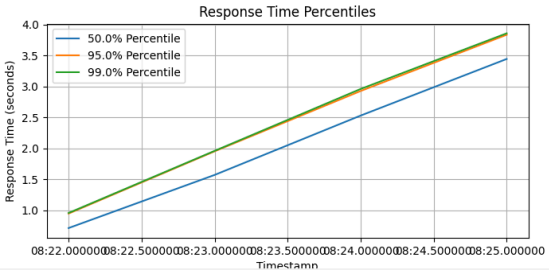
Metodologia Testului

- ❖ Configurație: Testul a fost configurat pentru a trimite un număr specific de cereri (e.g., 100 cereri) la un interval constant (e.g., 1 cerere pe secundă).
- ❖ Execuție: Cererile au fost trimise către server fără întârziere între ele pentru a simula un scenariu de utilizare constantă.

Rezultate Observate

- ❖ Timp Mediu de Răspuns: Calculat ca media timpilor de răspuns pentru toate cererile efectuate în cadrul testului.
- ❖ Variabilitate: Analiza variațiilor timpului de răspuns pentru a identifica orice potențiale fluctuații în capacitatea de procesare a serverului.
- ❖ Erori: Numărul de erori întâlnite, dacă există, care ar putea indica probleme de stabilitate sau configurare a serverului.





Testul de încărcare constantă oferă o bază solidă pentru a înțelege cum se comportă serverul sub o sarcină predictibilă și constantă. Rezultatele acestui test sunt esențiale pentru a asigura că serverul poate gestiona traficul anticipat fără probleme de performanță sau stabilitate.

Rezultatele Testului de Ramp-Up

Testul de ramp-up își propune să determine capacitatea serverului de a răspunde eficient la o creștere graduală a numărului de cereri pe o perioadă determinată. Acesta simulează un scenariu realist în care traficul către server crește treptat, un context comun în utilizarea de zi cu zi a aplicațiilor web.

Obiectivele Testului

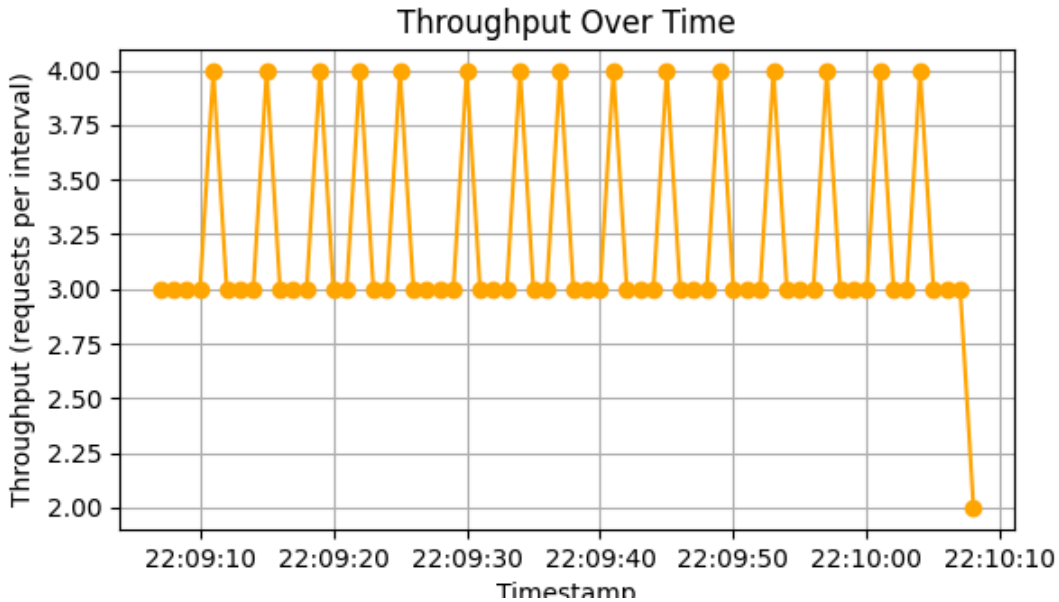
- Capacitatea de Scalare: Evaluarea modului în care serverul scalează când este supus unei creșteri progresive a cererilor.
- Timpul de Răspuns la Creșterea Traficului: Observarea variațiilor timpilor de răspuns pe măsură ce numărul de cereri crește.

Metodologia Testului

- Configurație: Testul a fost configurat pentru a crește numărul de cereri de la un număr inițial mic până la o valoare țintă pe durata unei perioade specificate (e.g., de la 10 la 200 cereri în 60 de secunde).
- Execuție: Cererile sunt trimise la intervale crescătoare, intensificând gradat presiunea asupra serverului pentru a monitoriza adaptabilitatea acestuia.

Rezultate Observate

- Timpul Mediu de Răspuns: A fost calculat pentru fiecare interval de timp, oferind o perspectivă asupra modului în care timpul de răspuns evoluează pe măsură ce încărcătura crește.
- Vârfuri în Timpul de Răspuns: Identificarea momentelor în care timpul de răspuns a crescut semnificativ, ceea ce ar putea indica limitări ale capacității serverului.
- Erori și Rate de Eșec: Numărul și tipul de erori întâlnite în timpul testului, care pot reflecta punctele critice ale infrastructurii sub încărcare.



Rezultatele Testului de Spike

Testul de spike investighează robustețea și elasticitatea serverului în fața unor creșteri bruste și temporare ale încărcăturii, oferind insights despre capacitatea serverului de a gestiona fluxuri de trafic intensiv fără avarii majore.

Obiectivele Testului

- ❖ Reziliența la Vârfuri de Trafic: Determinarea dacă serverul poate menține funcționalitatea sub condiții extreme de încărcare pentru scurte perioade de timp.
- ❖ Recuperarea Post-Spike: Analiza timpului necesar serverului pentru a reveni la performanța normală după un spike de trafic.

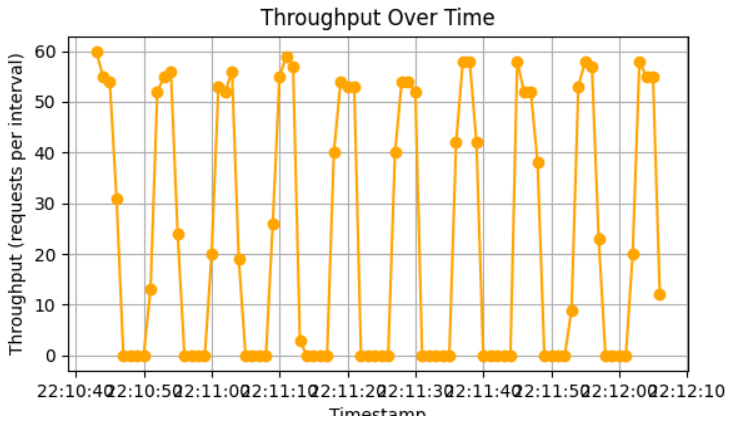
Metodologia Testului

- ❖ Configurație: Testul generează vârfuri de trafic prin trimiterea unui număr mare de cereri simultan (e.g., 500 de cereri într-o singură secundă), urmate de o perioadă de repaus pentru a permite serverului să se stabilizeze.
- ❖ Execuție: Acest ciclu se repetă de mai multe ori pentru a testa consecvența răspunsurilor serverului și capacitatea sa de recuperare.

Rezultate Observate

- ❖ Viteza de Răspuns în timpul Spike-urilor: Monitorizarea timpilor de răspuns în timpul fiecărui spike pentru a evalua impactul asupra performanței.
- ❖ Erori întâlnite: Înregistrarea și analiza erorilor produse în timpul spike-urilor, care pot indica suprasolicitarea resurselor serverului.
- ❖ Recuperare: Timpul necesar serverului pentru a reveni la performanța de bază după fiecare spike, evaluând astfel capacitatea sa de recuperare.

Testul de spike este vital pentru organizațiile care se așteaptă la trafic web sporadic sau evenimente promoționale cu trafic mare, oferind date valoroase despre cum trebuie pregătită infrastructura pentru a preveni întreruperi de serviciu. Rezultatele acestui test arată capacitatea serverului de a răspunde prompt și de a se recupera după vârfuri intense de cereri, indicând posibile nevoi de ajustare a capacității sau de îmbunătățire a strategiilor de scalare.



Rezultatele Testului de Stres

Testul de stres este esențial pentru a identifica punctele de saturație ale serverului și pentru a evalua capacitatea acestuia de a gestiona încărcături maxime pe perioade lungi de timp. Acest test ajută la descoperirea limitelor infrastructurii și la identificarea potențialelor probleme de stabilitate și performanță.

Obiectivele Testului

- ❖ Determinarea Limitelor de Performanță: Evaluarea punctului în care serverul începe să eșueze sub încărcături crescânde.
- ❖ Comportamentul sub Încărcare Maximă: Analiza performanței serverului în condiții de stres extrem și prelungit.
- ❖ Rate de Eroare și Stabilitate: Identificarea frecvenței și naturii erorilor pe măsură ce încărcătura crește.

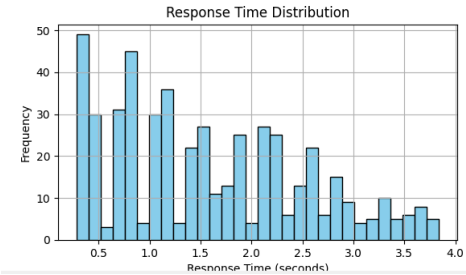
Metodologia Testului

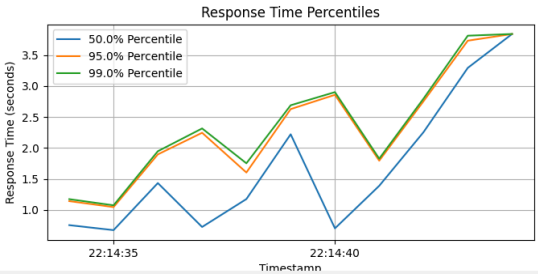
- ❖ Configurație: Testul începe cu un număr mic de cereri și crește treptat încărcătura până când serverul atinge un punct de eșec consistent sau se ajunge la numărul maxim de cereri specificat.
- ❖ Execuție: Cererile sunt trimise în cicluri, cu un număr de cereri crescând constant, monitorizând în același timp rata de eroare și timpul de răspuns.

Rezultate Observate

- ❖ Punctul de Saturație: Identificarea momentului în care serverul nu mai poate gestiona creșterea încărcăturii și începe să returneze erori.
- ❖ Timpul de Răspuns: Monitorizarea evoluției timpilor de răspuns pe măsură ce încărcătura crește.
- ❖ Rate de Eroare: Evaluarea numărului și tipului de erori întâlnite în timpul testului, care pot indica punctele de saturație ale resurselor serverului.

Testul de stres este crucial pentru a înțelege limitele infrastructurii serverului și pentru a identifica când și de ce serverul începe să eșueze sub încărcături extreme. Rezultatele obținute din acest test oferă informații valoroase pentru planificarea capacității, implementarea măsurilor de optimizare și asigurarea unei performanțe consistente și fiabile sub condiții de stres maxim.





Concluzii

Implementarea proiectului de benchmarking pentru serverul web a fost esențială pentru a evalua performanța actuală și pentru a identifica domeniile care necesită îmbunătățiri. Proiectul a inclus dezvoltarea unui script pentru efectuarea cererilor HTTP concurente, colectarea datelor privind răspunsurile serverului și analiza acestora prin intermediul vizualizărilor statistice detaliate.

Unul dintre principalele realizări ale acestui proiect a fost dezvoltarea și implementarea cu succes a funcțiilor asincrone pentru trimiterea cererilor HTTP, ceea ce a permis simularea diferitelor scenarii de încărcare (constantă, ramp-up, spike și stres). Acest lucru a permis o evaluare precisă a capacității serverului de a gestiona un număr mare de cereri într-un interval de timp scurt. Datele colectate au fost analizate pentru a genera vizualizări detaliate ale performanței serverului, incluzând distribuția timpului de răspuns, throughput-ul și rata de eroare. Aceste vizualizări au oferit o perspectivă valoroasă asupra capacității serverului de a gestiona cererile și asupra punctelor de congestie.

Un alt aspect important a fost utilizarea fișierului config.json și a interfeței grafice dezvoltate cu PyQt5, care a permis configurarea dinamică a parametrilor de test, făcând aplicația accesibilă și ușor de utilizat. Flexibilitatea oferită de această metodă a îmbunătățit semnificativ experiența utilizatorului și a facilitat testarea în diverse scenarii.

Cu toate acestea, există mai multe domenii care necesită îmbunătățiri pentru a extinde și aprofunda analiza. În viitor, ar fi benefică integrarea unor teste suplimentare care să evalueze performanța serverului sub diferite configurații de rețea și sarcini de lucru. De asemenea, ar fi utilă îmbunătățirea colectării datelor pentru a include metrici adiționali, cum ar fi utilizarea memoriei și a CPU-ului de către server. Aceasta ar oferi o imagine mai completă a performanței și ar permite identificarea mai precisă a bottleneck-urilor.

În concluzie, deși proiectul a făcut progrese semnificative în evaluarea performanței serverului web, este esențial să continuăm cu dezvoltarea și rafinarea metodelor noastre de testare și analiză pentru a asigura că serverul poate fi optimizat pentru a răspunde cerințelor crescute în mod eficient și fiabil.

Bibliografie

- *pandas - Python Data Analysis Library*, https://pandas.pydata.org/. Accessed 15 April 2024.
- Agnew, Sam. “Asynchronous HTTP Requests in Python with aiohttp and asyncio.” *Twilio*, 25 March 2021, https://www.twilio.com/en-us/blog/asynchronous-http-requests-in-python-with-aiohttp. Accessed 15 April 2024.
- Brownlee, Jason. “Asynchronous Requests in Python.” *Super Fast Python*, 10 December 2023, https://superfastpython.com/python-async-requests/. Accessed 15 April 2024.
- Fincher, Jon. “Reading and Writing CSV Files in Python – Real Python.” *Real Python*, https://realpython.com/python-csv/. Accessed 15 April 2024.
- “How to measure the elapsed time in Python.” *Python Engineer*, 13 January 2023, https://www.python-engineer.com/posts/measure-elapsed-time/. Accessed 15 April 2024.
- “Measuring Neural Network Performance: Latency and Throughput on GPU.” *YOUNESS-ELBRAG*, 1 February 2023, https://younsess-elbrag.medium.com/measuring-neural-network-performance-latency-and-throughput-on-gpu-5d54657871f0. Accessed 15 April 2024.
- Ronquillo, Alex. “Python's Requests Library (Guide) – Real Python.” *Real Python*, 28 February 2024, https://realpython.com/python-requests/. Accessed 15 April 2024.