Compararea performanței a unei aplicații de chat implementată utilizând tehnologii diferite: WebSockets - Publish/Subscribe

Autori:

Carcadea Răzvan Alexandru, SSA-2 (razvan.carcadea@stud.acs.upb.ro)

Draica Violeta Ana-Maria, AAC (violeta\_ana.draica@stud.mec.upb.ro)

**Rezumat:**

*Tematica lucrării se concentrează pe analiza și compararea performanței a două tehnologii utilizate pentru implementarea aplicațiilor de chat în timp real: WebSockets și modelul Publish/Subscribe (Pub/Sub). Sunt analizate metricele de performanță esențiale, în scopul identificării soluției optime pentru diverse scenarii de utilizare. Un aspect important al lucrării este dezvoltarea și testarea a două aplicații demonstrative utilizând tehnologiile specificate. Compararea detaliată a performanțelor celor două tehnologii oferă arhitecților și dezvoltatorilor informații utile pentru alegerea soluției potrivite.*

**Cuvinte cheie:** aplicații de chat, WebSocket, Publish/Subscribe, metrici de performanță, comunicare în timp real, mesaje

# Introducere

Aplicațiile de chat au devenit indispensabile în era digitală, având un impact semnificativ asupra modului în care oamenii comunică și colaborează. Domenii precum mesageria instantanee, jocurile multiplayer, colaborarea online și suportul clienți depind de capacitatea de a oferi comunicare în timp real. Pentru a implementa această funcționalitate, două tehnologii de bază sunt utilizate pe scară largă: WebSockets și modelul de mesagerie Publish/Subscribe (Pub/Sub).

Această lucrare își propune să analizeze comparativ performanța a două tehnologii esențiale pentru aplicațiile moderne de chat: WebSockets și modelul de mesagerie Publish/Subscribe (Pub/Sub). Comunicarea în timp real este o componentă critică pentru aplicațiile actuale, având utilizări variate, de la mesagerie personală la colaborare online și servicii de suport. Alegerea corectă a tehnologiei este crucială pentru arhitectura unei aplicații, iar această comparație oferă informații valoroase pentru arhitecți și dezvoltatori în procesul de selecție a soluției optime. Cele două tehnologii analizate sunt fundamentale pentru comunicarea în timp real: WebSockets, un protocol bidirecțional cu latență scăzută, ideal pentru aplicații interactive, dar cu limitări de scalabilitate, și Pub/Sub, un model scalabil și tolerant la erori, care utilizează un broker pentru a decupla publicatorii de consumatori. În acest scop, au fost dezvoltate două aplicații demonstrative: una bazată pe WebSockets, care oferă comunicare bidirecțională în timp real între client și server, și cealaltă bazată pe Pub/Sub, care gestionează fluxul mesajelor între producători și consumatori prin intermediul unui broker.

Pentru această comparație, au fost realizate:

* Două aplicații complet containerizate pentru portabilitate și izolare îmbunătățite;
* Testări detaliate utilizând unelte precum Artillery, pentru măsurarea performanțelor;
* Implementări flexibile, incluzând accesibilitatea Pub/Sub atât prin conexiuni interne Redis, cât și prin HTTP.

# Bază teoretică

## WebSockets

Protocolul WebSocket a fost standardizat de către IETF ca RFC 6455 în anul 2011. WebSocket este un protocol de comunicație care oferă canale de comunicare bidirecționale și full-duplex peste o singură conexiune TCP persistentă. Acesta permite schimbul de date în timp real între un client (de exemplu, un browser web) și un server. [1] Spre deosebire de HTTP, care operează pe baza unui model de tip cerere-răspuns, WebSocket-urile creează un canal deschis pentru schimb continuu de date în ambele direcții – exact ca în exemplul nostru cu Google Docs. [3]

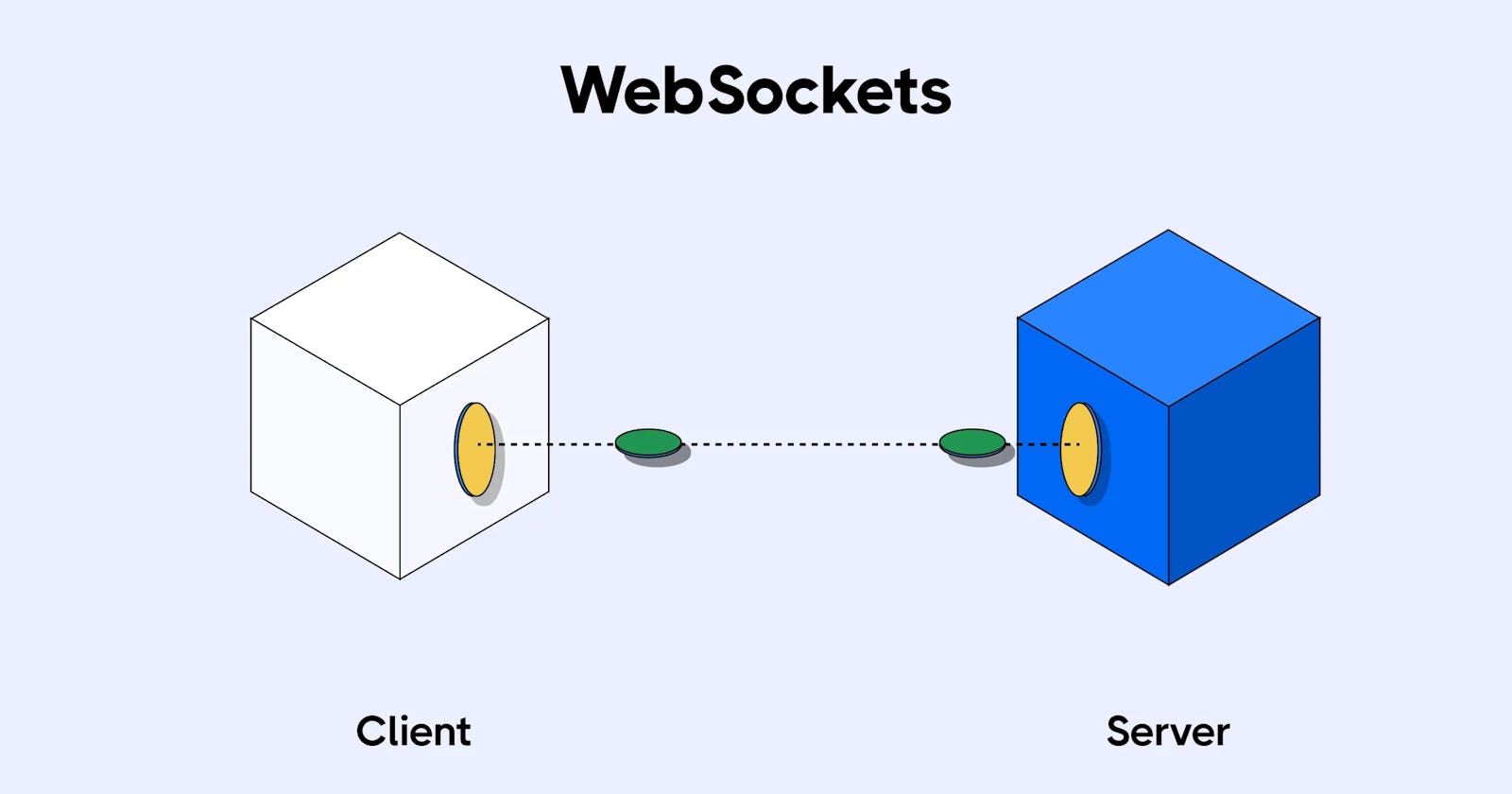


Figura 1. Arhitectura WebSocket Client-Server [3]

Este un protocol care păstrează starea conexiunii, ceea ce înseamnă că legătura dintre client și server rămâne activă până când una dintre părți (clientul sau serverul) o închide. Odată ce conexiunea este închisă de către una dintre părți, aceasta este întreruptă la ambele capete. [2]

Modul de funcționare al WebSockets începe cu clientul care trimite o cerere HTTP ce include un antet „Connection: Upgrade” pentru a iniția o negociere a conexiunii (handshake) WebSocket. Ca răspuns, serverul trimite un cod de stare HTTP 101, confirmând schimbarea protocolului către WebSocket. Odată ce conexiunea este stabilită, aceasta facilitează o comunicare bidirecțională, în timp real, fără cereri HTTP suplimentare. Mesajele sunt transmise, iar conexiunea rămâne activă până când este închisă explicit de oricare dintre părți sau este întreruptă din cauza problemelor de rețea. [1]

Este important de menționat că, la rularea pe nivelul protocolului WebSocket, acesta necesită un identificator uniform de resurse (URI) care să utilizeze schema „ws:” sau „wss:”, similar cu modul în care URL-urile HTTP folosesc întotdeauna schema „http:” sau „https:”.[1]

Google Chrome a fost primul browser care a inclus suport standard pentru WebSockets în 2009. RFC 6455—Protocolul WebSocket—a fost publicat oficial online în 2011. Protocolul WebSocket Google și API-ul WebSocket sunt standardizate de W3C și IETF, iar suportul pentru WebSockets este larg răspândit între browsere. [1]

Se recomandă folosirea WebSocket la:

* Aplicații web în timp real:O aplicație web în timp real folosește un WebSocket pentru a afișa date pe partea clientului, care sunt trimise continuu de serverul backend. În WebSocket, datele sunt transmise continuu prin aceeași conexiune deja deschisă, ceea ce face ca WebSocket să fie mai rapid și să îmbunătățească performanța aplicației. [2]
* Exemplu: Pe un site de tranzacționare sau tranzacționarea de Bitcoin, pentru afișarea fluctuației prețurilor și a mișcărilor, datele sunt transmise continuu de serverul backend către client printr-un canal WebSocket. [2]
* Aplicații de gaming: Într-o aplicație de gaming, datele sunt primite continuu de server, iar fără reîmprospătarea interfeței (UI), modificările apar automat pe ecran. UI-ul este actualizat automat fără a stabili o conexiune nouă, ceea ce face WebSocket extrem de util pentru astfel de aplicații. [2]
* Aplicații de chat: Aplicațiile de chat utilizează WebSockets pentru a stabili o conexiune o singură dată, care este folosită pentru schimbul, publicarea și difuzarea mesajelor între utilizatori. Aceeași conexiune WebSocket este reutilizată pentru trimiterea și primirea mesajelor, precum și pentru transferul mesajelor între utilizatori în mod direct („one-to-one”). [2]

Avantaje ale protocolului WebSokets:

* Comunicare full-duplex: Permite trimiterea și primirea simultană a datelor, fiind ideal pentru aplicații interactive, cum ar fi jocurile online și chat-urile live. [7]
* Latenta scăzută: Datele sunt transmise rapid după stabilirea conexiunii, cu întârziere minimă. [7]
* Reducerea suprasarcinii: Elimină cererile HTTP repetate prin menținerea unei conexiuni deschise, economisind resursele serverului și rețelei. [7]
* Scalabilitate: Gestionarea mai bună a resurselor pentru utilizatori simultani datorită conexiunilor persistente, mai ales cu strategii de scalare precum „load balancing”. [7]
* Suport extins: Este compatibil cu browserele moderne și ușor de integrat, fiind o opțiune versatilă pentru aplicații web și mobile. [7]

Dezavantaje ale protocolului WebSockets:

* Suport în browsere: Nu sunt compatibile cu browserele mai vechi, necesitând fallback-uri. [1]
* Limitări proxy/firewall: Pot fi blocate de unele firewall-uri sau proxy-uri. [1]
* Scalabilitate: Mențin conexiuni persistente care pot solicita resursele serverului. [1]
* Natura cu stare: Necesită gestionarea stării conexiunilor, consumând mai multă memorie. [1]
* Securitate: Necesită criptare (SSL/TLS) și protecție împotriva atacurilor (XSS, CSRF).[1]
* Pierderea conexiunii: Nu au mecanisme integrate pentru reconectare sau echilibrare a încărcării. [1]
* Detectarea prezenței: Deconectările sunt greu de detectat, limitând anumite funcționalități. [1]

## Publish/Subscribe

Pub/Sub (sau Publish/Subscribe) este un model arhitectural folosit în sistemele distribuite pentru comunicarea asincronă între diferite componente sau servicii. Deși se bazează pe modele anterioare precum cozile de mesaje și brokeri de evenimente, Pub/Sub este mai flexibil și mai scalabil. Cheia acestei flexibilități constă în faptul că Pub/Sub permite transmiterea mesajelor între diferite componente ale sistemului fără ca acestea să fie conștiente de identitatea reciprocă (sunt decuplate). [4]

**O imagine care conține text, diagramă, captură de ecran, Plan

Descriere generată automat**

Figura 2. Arhitectura Publish/Subscribe [4]

Modelul Publish/Subscribe (Pub/Sub) funcționează pe principiul decuplării producătorilor de mesaje de consumatori în sistemele distribuite. Componentele principale ale modelului Publish/Subscribe sunt: [5]

1. Publishers (Publicatori): Entități care generează și trimit mesaje către un sistem centralizat cunoscut sub numele de broker de mesaje (message broker). Aceste mesaje sunt asociate cu subiecte sau canale specifice, însă cei care publică nu le adresează către destinatari specifici. [5]
2. Broker-ul de mesaje: Componenta centrală în modelul Pub/Sub, responsabilă pentru gestionarea, stocarea și rutarea mesajelor. Broker-ul ține evidența tuturor abonaților activi și a subiectelor la care aceștia s-au abonat. [5]
3. Subscribers (Abonați): Entități care își înregistrează interesul pentru subiecte specifice la broker-ul de mesaje. Acestea nu trebuie să cunoască direct publicatorii. Ori de câte ori un mesaj ajunge la broker pentru un subiect la care s-au abonat, broker-ul direcționează mesajul către aceștia. [5]
4. Rutarea mesajelor: Când broker-ul primește un mesaj de la un publisher, acesta identifică subiectul asociat mesajului. După aceea, verifică toți abonații interesați de acel subiect și le transmite mesajul. [5]
5. Procesare asincronă: Livrarea și procesarea mesajelor sunt asincrone. Abonații primesc și procesează mesajele în fundal, permițând sistemului să gestioneze volume mari de mesaje în mod concurent. [5]
6. Garanții de livrare: În funcție de capacitățile și configurația broker-ului, pot fi asigurate diferite garanții de livrare: [5]

* Livrare cel mult o dată („At-Most-Once Delivery”): Mesajele sunt livrate abonaților o singură dată sau deloc, asigurând absența copiilor, dar cu posibilitatea pierderii unor mesaje. [5]
* Livrare cel puțin o dată („At-Least-Once Delivery”): Mesajele sunt garantate să ajungă la abonați, dar există posibilitatea să fie livrate de mai multe ori, ceea ce poate duce la duplicate. [5]
* Livrare exact o dată („Exactly-Once Delivery”): Fiecare mesaj este garantat să fie livrat și procesat de către abonați o singură dată, prevenind atât pierderile, cât și duplicarea mesajelor. [5]

Se recomandă utilizarea Pub/Sub la:

* Notificări în timp real: Dacă trebuie să informezi utilizatorii despre activități din aplicația ta folosind mesaje în aplicație, Pub/Sub poate trimite eficient notificări în timp real. Garanția de livrare „cel mult o dată” este suficientă aici, deoarece pierderea ocazională a mesajelor ar putea fi acceptabilă, iar sistemul ar trebui să prevină spamarea utilizatorilor cu notificări duplicate. [5]
* Sisteme financiare: Pe platformele de tranzacționare de acțiuni, este esențial ca fluxurile de activitate să actualizeze traderii despre modificările de preț, tranzacțiile sau știrile care ar putea influența valorile acțiunilor. Datorită naturii critice a tranzacțiilor financiare, garanția de livrare „exact o dată” devine crucială pentru a preveni acțiuni precum tranzacțiile duble sau pierderea unor evenimente importante de pe piață. [5]
* Aplicații de chat: Soluții de chat multi-utilizator în care utilizatorii se abonează la camere de chat sau canale specifice și primesc mesaje publicate de alți utilizatori. Garanția „cel puțin o dată” funcționează aici pentru a asigura livrarea constantă a mesajelor. [5]
* Procesare de fluxuri și analize: Pentru analiza fluxurilor mari de date în timp real, cum ar fi fluxurile de clicuri ale utilizatorilor sau tranzacțiile din comerțul electronic, sistemele pot utiliza Pub/Sub. Aceste fluxuri pot fi procesate pentru a obține informații, a genera rapoarte sau a declanșa alte acțiuni. Aici, garanția „cel puțin o dată” asigură că nu se pierd date în timpul analizei. [5]
* Coordonarea sistemelor distribuite: În arhitecturile microservicii, serviciile trebuie adesea să coordoneze acțiuni sau să fie informate despre evenimentele din alte servicii. Pub/Sub oferă un mecanism pentru ca aceste servicii să comunice fără a fi direct conectate. Garanția de livrare aici va fi aleasă în funcție de importanța comunicării între servicii. [5]
* Distribuirea de conținut: Când devine disponibil un conținut nou, precum articole într-un flux de știri sau videoclipuri pe o platformă de streaming, Pub/Sub poate notifica utilizatorii sau sistemele interesate. În funcție de preferințele utilizatorilor pentru experiența acestora, garanțiile „cel mult o dată” sau „cel puțin o dată” ar putea fi adecvate. [5]

Avantajele Pub/Sub:

* Decuplare flexibilă între componente, ceea ce face sistemul mai modular și mai flexibil.
* Scalabilitate mare (în teorie, Pub/Sub permite orice număr de publicatori să comunice cu orice număr de abonați). [4]
* Independență față de limbaje și protocoale, ceea ce face integrarea modelului Pub/Sub în tehnologiile tale rapidă și simplă. [4]
* Comunicare asincronă, bazată pe evenimente, ideală pentru aplicații în timp real, cu latență scăzută. [4]

Dezavantajele Pub/Sub:

* Livrări de mesaje eșuate: Problemele pot apărea dacă cei care publică nu pot verifica dacă mesajele lor ajung la toți abonații și dacă aceștia nu primesc toate informațiile, ceea ce poate duce la pierderi de date. Folosirea unui broker de mesaje poate ajuta la rezolvarea acestei probleme. [6]
* Instabilitate la scalare rapidă: Scalarea rapidă a unui sistem Pub/Sub poate cauza instabilitate, provocând o suprasolicitare a rețelei și încetinirea proceselor din cauza cererilor multiple ale abonaților. [6]
* Scalare dinamică poate fi inutilă: Pub/Sub poate fi prea complex pentru operațiuni de mică amploare, iar pentru sisteme care nu necesită scalare exponențială, metode mai simple precum polling-ul pot fi mai eficiente. [6]

# Metodologie

## Metrici de performanță

Pentru evaluarea performanțelor WebSockets și Pub/Sub, s-au analizat metrici precum latența, scalabilitatea, toleranța la erori, suprasarcina de conexiune și debitul. Studiul a inclus dezvoltarea a două aplicații demonstrative, fiecare implementând una dintre tehnologii, și testarea acestora într-un mediu controlat pentru a evidenția diferențele dintre cele două abordări.

Latența sau întârzierea rețelei este un metric de performanță a rețelei care măsoară timpul necesar pentru a transfera date de la o destinație la alta [8], în cazul acesta reprezentând timpul necesar pentru livrarea unui mesaj.

Scalabilitatea reprezintă capacitatea unui sistem de a ajusta performanța și costurile în funcție de variațiile cerințelor de procesare ale aplicațiilor și infrastructurii, asigurându-se că resursele sunt gestionate eficient pe măsură ce volumul de date sau numărul de utilizatori crește sau scade. [9] În contextul aplicațiilor de chat, scalabilitatea reprezintă capacitatea de a gestiona cât mai mulți utilizatori și mesaje simultan.

Debitul permite măsurarea ratei reale de transmitere a datelor prin diferite zone ale rețelei. În timp ce lățimea de bandă îți oferă limita teoretică a transferului de date, debitul măsoară cantitatea reală de pachete de date trimise cu succes la destinație prin rețea. Debitul poate varia în funcție de zonele rețelei. Un debit scăzut indică pachete pierdute care trebuie retransmise. [8]

Toleranța la erori reprezintă capacitatea unui sistem de a continua să funcționeze chiar și atunci când unul sau mai multe dintre componentele sale nu funcționează corespunzător. Problemele de întrerupere a serviciilor cauzate de defecțiuni software sau erori logice sunt gestionate prin toleranța la erori. [10]

Garanția de livrare reprezintă un aspect esențial în analiza performanțelor unui sistem de mesagerie, indicând fiabilitatea cu care mesajele sunt livrate între sursă și destinație. Aceasta este un metric crucial în alegerea unei tehnologii de comunicare, deoarece determină modul în care sunt gestionate erorile și pierderile de mesaje în timpul transmiterii.

Suprasarcina de conexiune este o măsură a resurselor consumate pentru menținerea activă a conexiunilor între client și server pe durata unui proces de comunicare. Aceasta include resurse precum procesorul, memoria și lățimea de bandă utilizată pentru stabilirea, menținerea și închiderea conexiunilor.

## Mediul de testare

Pentru testare, a fost ales mediul de testare Artillery, ce este o platformă scalabilă, flexibilă și ușor de utilizat, care conține tot ce este necesar pentru testarea performanței la scară largă, destinată mediilor de producție. Artillery.io este un instrument „open-source” pentru testarea încărcării aplicațiilor care rulează pe protocoalele HTTP, WebSocket și Socket.io. Acesta oferă dezvoltatorilor o soluție cuprinzătoare și flexibilă pentru testarea încărcării. Prin simularea comportamentului real al utilizatorilor, Artillery permite dezvoltatorilor să identifice posibile probleme de performanță și să se asigure că aplicațiile lor pot gestiona încărcături mari de utilizatori, fără a compromite funcționalitatea sau experiența utilizatorului. [11]

Artillery funcționează prin generarea și trimiterea unui volum mare de cereri concurente către aplicația țintă. Aceste cereri pot fi personalizate pentru a imita diverse scenarii de utilizatori, cum ar fi înregistrarea, navigarea prin produse sau efectuarea de cumpărături. Prin varierea încărcăturii de utilizatori și a comportamentului acestora, Artillery oferă informații valoroase despre performanța sistemului, timpii de răspuns și scalabilitatea generală. [11]

Una dintre caracteristicile unice ale Artillery este capacitatea sa de a executa scenarii complexe cu ușurință. Dezvoltatorii pot crea fluxuri de lucru personalizate folosind JavaScript, permițând simularea mai precisă a comportamentului utilizatorului. Această flexibilitate le permite dezvoltatorilor să testeze diverse scenarii, inclusiv cazuri limită și scenarii de performanță extremă, asigurându-se că aplicațiile lor pot gestiona orice situație care poate apărea în producție. [11]

# Soluția propusă

Pentru a realiza comparația între cele două tehnologii, s-au realizat două aplicații în JavaScript, fiecare implementând o soluție distinctă: una bazată pe WebSockets pentru a permite comunicarea bidirecțională în timp real între client și server și cealaltă utilizând modelul Publish/Subscribe (Pub/Sub) pentru a gestiona schimbul de mesaje prin intermediul unui broker Redis. De asemenea, metoda Pub/Sub este accesibilă și prin HTTP, oferind o alternativă suplimentară pentru comunicarea între client și server, pe lângă conexiunea internă prin Redis. Fiecare aplicație a fost implementată pe un site creat cu HTML, care servește drept interfață web pentru utilizatori, permițându-le să interacționeze cu sistemul de chat.

# Testare și Rezultate

Ca prim test, ambele tehnologii au gestionat un număr de 1200 de cereri, respectiv mesaje, cu o rată de succes de 100% în ambele cazuri. În tabelele 4.1 și 4.2 se află rezultatele în urma simulării. Pentru numărul de 1200 de cereri/mesaje, Pub/Sub a obținut o rată de procesare de aproximativ 35 de cereri HTTP/s, iar WebSocket a procesat aproximativ 30 de mesaje WebSocket/s.

În ceea ce privește timpii de răspuns, Pub/Sub a avut un timp mediu de răspuns de 2.8 ms, cu un P95 de 7.9 ms și un P99 de 12.1 ms. Pe de altă parte, WebSocket a avut o latență mai mare, inferată din durata sesiunii, cu o medie de aproximativ 4 ms și un P95 de 4.1 ms. Durata medie a sesiunilor a fost mai scurtă pentru Pub/Sub, cu aproximativ 3024 ms, comparativ cu WebSocket, unde durata medie a fost de 4012 ms, sugerând o latență ușor mai mare per operațiune în cazul WebSocket datorită conexiunii persistente.

În ceea ce privește scalabilitatea, ambele tehnologii au reușit să gestioneze 300 de utilizatori virtuali simultani pe o perioadă de 30 de secunde fără erori. Totuși, procesarea mesajelor în WebSocket a fost mai rapidă per mesaj datorită dimensiunii mai mici a acestora, în timp ce Pub/Sub a implicat o suprasarcină suplimentară pentru stabilirea și închiderea conexiunii, dar s-a comportat bine în aplicațiile ce necesită operațiuni fără stare.

Dintre punctele forte și slăbiciunile celor două tehnologii regăsite în tabelul 4.2, Pub/Sub a avut o suprasarcină mai mare din cauza configurării frecvente a conexiunilor, dar a oferit o latență scăzută și consistentă (~2.8ms), deoarece fiecare cerere-răspuns este rapid și izolat, fiind potrivit pentru mesaje ușoare și cereri independente. WebSocket, în schimb, a avut o suprasarcină mai mică datorită conexiunilor persistente și a demonstrat o scalabilitate bună pentru aplicații ce necesită comunicare continuă în timp real, însă a avut o latență ușor mai mare.

Tabel 4.1. Compararea performanțelor pentru 1200 mesaje, respectiv cereri

|  |  |  |
| --- | --- | --- |
| **Metrica** | **WebSocket** | **Pub/Sub (HTTP)** |
| Număr total operații | 1200 mesaje WebSocket trimise | 1200 cereri HTTP |
| Rata de succes | 100% (vusers.failed = 0) | 100% (http.codes.200 = 1200) |
| Debit (Rata de Cereri/Mesaje) | ~30 mesaje WebSocket/s | ~35 cereri HTTP/s |
| Latență (Timp de Răspuns – Medie) | ~ 4 ms | ~ 2.8 ms |
| Latență (Timp de Răspuns – P95) | ~ 4.1 ms (dedus) | ~ 7.9 ms |
| Durata Sesiunii | ~ 4012 ms | ~ 3024 ms |
| Date Descărcate | - | ~ 54 KB răspunsuri HTTP |
| Scalabilitate | 300 utilizatori/30 sec | 300 utilizatori/30 sec |

Tabel 4.2. Puncte forte și slăbiciuni pentru 1200 mesaje, respectiv cereri

|  |  |  |
| --- | --- | --- |
| **Metrica** | **WebSocket** | **Pub/Sub (HTTP)** |
| Suprasarcina | < datorită conexiunilor persistente | > din cauza configurării frecvente a conexiunilor |
| Latență | Usor mai mare, dedusă din durata sesiunii ( ~4ms) | Scăzută, măsurabilă si constantă (~2.8 ms mediu) |

Mai jos este prezentată analiza performanței pentru testarea cu 6000 de mesaje WebSocket și 6000 de cereri HTTP utilizând Pub/Sub, rezultatele regăsindu-se în tabelele 4.3 și 4.4. Ambele tehnologii au obținut o rată de succes de 100%, gestionând fără erori toate cererile și mesajele procesate, indicând stabilitatea și fiabilitatea infrastructurii testate.

În ceea ce privește debitul, WebSocket a procesat aproximativ 250 de mesaje pe secundă, comparativ cu Pub/Sub, care a gestionat aproximativ 180 de cereri HTTP pe secundă. Acest avantaj al WebSocket este atribuit conexiunilor persistente, care reduc suprasarcina asociată configurării frecvente a conexiunilor.

Pub/Sub a înregistrat timpi de răspuns mai scăzuți, cu o medie de aproximativ 7.4ms și un P95 de 15ms, ceea ce demonstrează consistență în livrarea răspunsurilor. Pe de altă parte, latența WebSocket a fost mai mare, cu o medie de aproximativ 10.2ms și un P95 dedus de aproximativ 20ms, datorită gestionării conexiunilor persistente și a comunicării bidirecționale.

Durata medie a sesiunilor pentru WebSocket a fost de aproximativ 10200ms, semnificativ mai lungă decât pentru Pub/Sub (~3055ms), ceea ce reflectă modelul persistent al WebSocket comparativ cu modelul de cerere-răspuns al Pub/Sub. În ceea ce privește descărcarea datelor, Pub/Sub a transferat aproximativ 270KB de date în răspunsurile HTTP, în timp ce pentru WebSocket datele descărcate nu au fost monitorizate în acest test.

Din punct de vedere al scalabilității, ambele tehnologii au demonstrat o scalabilitate bună, gestionând simultan 1500 de utilizatori virtuali pe o perioadă de 30 de secunde fără erori.

Tabel 4.3. Compararea performantelor pentru 6000 de mesaje, respectiv cereri

|  |  |  |
| --- | --- | --- |
| **Metrica** | **WebSocket** | **Pub/Sub (HTTP)** |
| Număr total operații | 6000 mesaje WebSocket trimise | 6000 cereri HTTP |
| Rata de succes | 100% (vusers.failed = 0) | 100% (http.codes.200 = 6000) |
| Debit (Rata de Cereri/Mesaje) | ~250 mesaje WebSocket/s | ~180 cereri HTTP/s |
| Latență (Timp de Răspuns – Medie) | ~ 10.2 ms | ~ 7.4 ms |
| Latență (Timp de Răspuns – P95) | ~ 20 ms (dedus din sesiuni) | ~ 15 ms |
| Durata Sesiunii | ~ 10200 ms | ~ 3055 ms |
| Date Descărcate | - | ~ 270 KB răspunsuri HTTP |
| Scalabilitate | 1500 utilizatori/30 sec | 1500 utilizatori/30 sec |

Tabel 4.4. Puncte forte și slăbiciuni pentru 6000 mesaje, respectiv cereri

|  |  |  |
| --- | --- | --- |
| **Metrica** | **WebSocket** | **Pub/Sub (HTTP)** |
| Suprasarcina | Redusă datorită conexiunilor persistente | > mai mare din cauza configurării frecvente a conexiunilor |
| Latență | Mai mare (~10.2ms medie), dar cu avantaj pentru mesaje frecvente. | Mai mică (~7.4ms medie), potrivită pentru cereri independente. |

Ca test suplimentar, ambele tehnologii au fost evaluate pentru un volum mai mare de operațiuni, respectiv 12000 de mesaje în cazul WebSocket și 12000 de cereri HTTP pentru Pub/Sub. Ambele tehnologii au înregistrat o rată de succes de 100%, demonstrând capacitatea de a gestiona cererile fără erori, așa cum este ilustrat în tabelele 4.5 și 4.6.

În ceea ce privește debitul, WebSocket a obținut un rezultat semnificativ mai bun, procesând aproximativ 450 de mesaje pe secundă, comparativ cu Pub/Sub, care a gestionat aproximativ 349 de cereri HTTP pe secundă. Acest avantaj al WebSocket este atribuit frecvenței ridicate de mesaje posibilă datorită conexiunilor persistente.

Din perspectiva latenței, Pub/Sub a avut un timp mediu de răspuns mai mic (~11ms) comparativ cu WebSocket (~16.4ms). Totuși, la percentila 95 (P95), latența pentru Pub/Sub a crescut semnificativ (~51ms), în timp ce WebSocket a menținut un P95 mai mic (~28.3ms), ceea ce sugerează o performanță mai consistentă în condiții de sarcină ridicată.

Durata medie a sesiunilor a fost considerabil mai mică pentru Pub/Sub (~3087ms), reflectând modelul său de cerere-răspuns, în care fiecare conexiune este închisă după completarea cererii. În contrast, WebSocket a avut o durată medie a sesiunii de aproximativ 16442ms, indicând o latență mai mare per operațiune datorată gestionării conexiunilor persistente.

În privința scalabilității, ambele tehnologii au reușit să gestioneze 3000 de utilizatori virtuali simultani pe o perioadă de 30 de secunde fără erori, demonstrând o scalabilitate excelentă în condiții de testare intensivă.

Conform tabelului 4.6, WebSocket a avut o suprasarcină mai mică datorită utilizării conexiunilor persistente, ceea ce l-a făcut mai eficient pentru gestionarea unui număr mare de mesaje frecvente în timp real. În schimb, Pub/Sub a avut o suprasarcină mai mare din cauza configurării și închiderii frecvente a conexiunilor, ceea ce poate influența performanța în aplicațiile cu multe cereri simultane.

Latența Pub/Sub a fost mai bună și constantă (~11ms în medie), fiind potrivită pentru mesaje ușoare și operațiuni independente. Pe de altă parte, WebSocket a demonstrat un debitsuperior și o latență mai mică la percentila 95, ceea ce îl face ideal pentru aplicații ce necesită comunicare continuă, în timp real.

Din punct de vedere al implementării, Pub/Sub este mai simplu de configurat datorită modelului său de cerere-răspuns, în timp ce WebSocket necesită gestionarea mai complexă a sesiunilor, dar oferă avantaje semnificative pentru aplicațiile interactive.

Tabel 4.5. Compararea performantelor pentru 12000 de mesaje, respectiv cereri

|  |  |  |
| --- | --- | --- |
| **Metrica** | **WebSocket** | **Pub/Sub (HTTP)** |
| Număr total operații | 12000 mesaje WebSocket trimise | 12000 cereri HTTP |
| Rata de succes | 100% (vusers.failed = 0) | 100% (http.codes.200 = 12000) |
| Debit (Rata de Cereri/Mesaje) | ~450 mesaje WebSocket/s | ~349 cereri HTTP/s |
| Latență (Timp de Răspuns – Medie) | ~ 16.4 ms | ~ 11 ms |
| Latență (Timp de Răspuns – P95) | ~ 28.3 ms (dedus) | ~ 51 ms |
| Durata Sesiunii | ~ 16442 ms | ~ 3087 ms |
| Date Descărcate | - | ~ 540 KB răspunsuri HTTP |
| Scalabilitate | 3000 utilizatori/30 s | 3000 utilizatori/30 s |

Tabel 4.6. Puncte forte și slăbiciuni pentru 12000 mesaje, respectiv cereri

|  |  |  |
| --- | --- | --- |
| **Metrica** | **WebSocket** | **Pub/Sub (HTTP)** |
| Suprasarcina | < datorită conexiunilor persistente, are o frecvență ridicată | > din cauza configurării frecvente a conexiunilor, direct proporțională cu numărul utilizatorilor |
| Latență | Mai mare, dedusă din durata sesiunii, dar cu un debit mai bun ( ~16.4ms) | Mai bună și constantă, dar mai ineficientă (~11ms mediu) |

Pentru o analiză mai clară și o comparație detaliată a performanței celor două tehnologii, am realizat o serie de grafice care evidențiază comportamentul fiecărei tehnologii în funcție de diverși parametri. Astfel, pentru a vizualiza mai eficient datele obținute în urma testelor, am structurat informațiile pe următoarele grafice:

* Graficul pentru debit, ilustrând numărul de cereri procesate pe secundă de fiecare tehnologie, este prezentat în Figura 3. Acest grafic oferă o imagine de ansamblu asupra capacității fiecărei tehnologii de a gestiona mesaje în funcție de volumul acestora.
* Graficul pentru durata sesiunii, care arată timpul total necesar pentru procesarea unui set de mesaje, este regăsit în Figura 4. Acesta subliniază diferențele dintre WebSocket și Pub/Sub în ceea ce privește menținerea sesiunii active și gestionarea conexiunilor pe termen lung.
* Graficul pentru latență include două componente esențiale:
* Latența medie, care reflectă timpul mediu de răspuns al fiecărei tehnologii, și care poate fi observată în Figura 5. Acest grafic subliniază diferențele semnificative în ceea ce privește timpul de livrare a mesajelor într-un interval de timp definit.
* Latența P95, care oferă o imagine mai detaliată a performanței la percentila 95, adică timpul de răspuns în care 95% din mesaje sunt livrate, iar aceasta este reprezentată în Figura 6. Acesta evidențiază variațiile de performanță pe măsură ce numărul de mesaje crește, fiind util pentru înțelegerea stabilității fiecărei tehnologii în condiții de încărcare ridicată.

O imagine care conține text, captură de ecran, linie, Interval

Descriere generată automat

Figura 3. Graficul comparativ al debitului pentru cele trei teste realizate

O imagine care conține text, captură de ecran, linie, Interval

Descriere generată automat

Figura 4. Graficul comparativ al duratei sesiunilor pentru cele trei teste realizate

O imagine care conține captură de ecran, text, linie, Interval

Descriere generată automat

Figura 5. Graficul comparativ al latenței medii pentru cele trei teste realizate

O imagine care conține captură de ecran, text, linie, Interval

Descriere generată automat

Figura 6. Graficul comparativ al latenței P95 pentru cele trei teste realizate

# Concluzii

Lucrarea de față prezintă comparația dintre tehnologiile WebSocket și Publish/Subscribe (Pub/Sub) în implementarea aplicațiilor de chat în timp real, s-a concluzionat că fiecare tehnologie are avantaje unice care o fac potrivită pentru anumite scenarii. WebSocket se remarcă prin latență scăzută și comunicare continuă, fiind ideal pentru aplicații în timp real, precum jocuri multiplayer sau colaborare live. Pe de altă parte, Pub/Sub oferă o arhitectură simplificată și flexibilitate în sisteme distribuite, fiind potrivit pentru aplicații orientate pe evenimente, cum ar fi notificările și distribuția de conținut.

În urma testelor realizate, ambele tehnologii, WebSocket și Pub/Sub (HTTP), au demonstrat performanțe excelente, cu o rată de succes de 100% pentru toate testele, indiferent de numărul de cereri sau mesaje procesate. Totuși, diferențele între cele două au fost evidente în ceea ce privește debitul, latența și scalabilitatea, fiecare tehnologie având avantaje în funcție de tipul aplicației și cerințele de performanță.

WebSocket a excelat în ceea ce privește debitul, procesând un număr mai mare de mesaje pe secundă datorită conexiunilor persistente, care reduc suprasarcina asociată configurării frecvente a conexiunilor. De asemenea, WebSocket a demonstrat o latență mai mică la percentila 95, ceea ce îl face mai eficient în condiții de sarcină mai mare. Cu toate acestea, durata medie a sesiunilor a fost semnificativ mai mare, ceea ce sugerează o latență mai mare per operațiune datorită conexiunilor persistente.

Pub/Sub, pe de altă parte, a înregistrat o latență mai scăzută și mai constantă, fiind potrivit pentru aplicațiile ce necesită mesaje ușoare și operațiuni independente. Totuși, Pub/Sub a avut o suprasarcină mai mare din cauza configurării frecvente a conexiunilor, ceea ce poate afecta performanța în cazul aplicațiilor cu cereri simultane. Latența sa constantă și rapidă îl face ideal pentru aplicațiile ce necesită timpi de răspuns predictibili și operațiuni rapide.

Scalabilitatea ambelor tehnologii a fost remarcabilă, ambele reușind să gestioneze 3000 de utilizatori simultani pe o perioadă de 30 de secunde fără erori. De asemenea, ambele soluții s-au comportat excelent la volum mai mare de cereri și mesaje, respectiv 6000 și 12000 de operațiuni, fără a întâmpina erori semnificative.

În ceea ce privește implementarea, WebSocket a necesitat o gestionare mai complexă a sesiunilor datorită conexiunilor persistente, dar a oferit performanțe superioare pentru aplicațiile interactive ce necesită comunicare continuă în timp real. În schimb, Pub/Sub a avut un model simplu de cerere-răspuns, fiind mai ușor de implementat, dar mai puțin eficient pentru aplicațiile ce necesită mesaje frecvente și în timp real.

Alegerea tehnologiei potrivite trebuie să fie ghidată de cerințele specifice ale aplicației, asigurând un echilibru optim între performanță, scalabilitate și ușurința de implementare. Acest studiu subliniază importanța unei analize atente în proiectarea sistemelor moderne de comunicare.

În concluzie, WebSocket este recomandat pentru aplicațiile care necesită comunicare continuă, interactivă, și un debit ridicat de mesaje, fiind ideal pentru aplicațiile în timp real, dar care pot tolera o latență mai mare. Pub/Sub este mai potrivit pentru aplicațiile care se bazează pe mesaje ușoare, independente și cereri simple, unde latența scăzută și consistența timpurilor de răspuns sunt critice.

# 7. Bibliografie

[1] What are WebSockets ws:// & wss:// connections, pubnub.com, <https://www.pubnub.com/guides/websockets/>

[2] What is web socket and how it is different from the HTTP?, www.geeksforgeeks.org, <https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/>

[3] Quick and Simple Method for Building a WebSocket Server in Node.js, <https://gauravbytes.hashnode.dev/quick-and-simple-method-for-building-a-websocket-server-in-nodejs>

[4] What is Pub/Sub? The Publish/Subscribe model explained, <https://ably.com/topic/pub-sub>

[5] Publish/Subscribe (Pub/Sub), <https://getstream.io/glossary/publish-subscribe/>

[6] What is Publish-Subscribe (Pub/Sub Model)?, <https://www.pubnub.com/guides/pub-sub/>

[7]WebSocket Advantages, Disadvantages, and Some Practical Advice, <https://maybe.works/blogs/websocket-what-it-is-when-to-use>

[8] What are Network Performance Metrics?, <https://www.solarwinds.com/resources/it-glossary/network-metrics?utm_source=chatgpt.com>

[9] <https://www.gartner.com/en/glossary>

[10] Fault Tolerance Explored: Navigating Systems Resilience, Security, and Best Practices, <https://www.zenarmor.com/docs/network-security-tutorials/what-is-fault-tolerance>   
[11] Introducing Artillery: The Next-Gen Tool for Load Testing, <https://www.workwithloop.com/blog/introducing-artillery-the-next-gen-tool-for-load-testing>