Analiza algoritmilor de load balancing si scaling

Stefanica Robert-Manuel

Cuprins

Introducere

Tehnologia a avut un impact important în viețile tuturor, odată cu apariția acesteia, modul in care muncim, învățăm, socializăm și ne informăm s-a schimbat radical. Tehnologia ne-a permis să vorbim cu persoanele dragi din viață noastră, să facem cumpărături și multe altele din confortul propriei case.

Pentru ca toate aceste lucruri sa fie posibile, ca toți oamenii sa fie interconectați prin intermediul Internetului și să acceseze aceste servicii într-un mod rapid și fiabil, cu timpi de răspuns scăzuți este nevoie de utilizarea de sisteme distribuite pentru a face față traficului ridicat la care sunt supuse unele dintre aceste aplicații.

Un sistem distribuit poate fi văzut ca o grupare de mai multe unități de procesare sau noduri care sunt conectate între ele, partajând astfel resursele.

În cadrul acestei lucrări voi detalia implementarea unui mecanism care gestionează un astfel de sistem distribuit, alocarea dinamică de resurse și gestionarea traficului din cadrul acestuia, utilizând diverși algoritmi: round robin, least connection, weighted response time și randomized. Platforma care urmează să fie dezvoltată își propune de asemenea să permită administratorului să selecteze tipul de algoritm care dorește să fie utilizat pentru load balancing la nivel granular pentru fiecare resursă de procesare gestionată, precum si setarea metricilor care vor fi utilizate pentru scalarea resurselor. Platforma va permite de asemenea adăugarea ulterioară, cu ușurință, de noi algoritmi pentru load balancing. Lucrarea îți propune de asemenea să măsoare performanța acestor algoritmi in diverse scenarii de utilizare.

Motivație

Consider că sistemele și mecanismele de load balancing si scalling se află printre principalele componente pe care trebuie să le aibă încorporat un sistem modern, care dorește să poată acomoda un număr ridicat de utilizatori și să le ofere acestora o experiență rapidă și plăcută de utilizare.

De asemenea datorită modularității platformei care urmează să fie implementată acesta va permite introducerea ulterioară de noi algoritmi pentru load balancing, precum si adaptarea acesteia pentru diverși furnizori de servicii cloud.

Un alt avantaj al acestei platforme este interfața prietenoasa cu utilizatorul de tip web, în care acesta va putea sa configureze modalitatea de load balancing și scaling, să adauge noi tipuri de resurse care trebuie sa fie administrate sau să le șteargă, precum și vizualizare nodurilor active din sistemul distribuit, al performantei și utilizării resurselor gestionate.

Context

În ultimii ani, numărul de persoane cu acces la Internet a crescut considerabil, astfel crescând și numărul de utilizatori ai aplicațiilor și platformelor populare. Pentru a permite utilizatorilor să le acceseze serviciile, și pentru a permite un număr tot mai mare de utilizatori, aceste platforme își rulează aplicațiile în cadrul unui sistem distribuit, care se scalează în funcție de numărul de utilizatori activi la un moment.

Prin introducerea de sisteme distribuite, au apărut o serie de probleme noi, care trebuie gestionate de către arhitecții de sistem, și anume cum să se scaleaze o anumită resursă, când se creează o noua resursă pentru a acomoda numărul în creștere de utilizatori, dar și cum se distruge una dintre resurse atunci când numărul de utilizare activi scade. O altă problemă în contextul sistemelor distribuite este modalitatea în care se redirectează cererile de date de la utilizatori către resursele de procesare corespunzătoare, astfel încât timpul de așteptare pentru utilizator sa fie cât mai mic.

Pentru a gestiona această redirectare a cererilor către resursele specializate pentru procesarea lor, exista numeroși algoritmi a căror performanță poate fi diferită în funcție de tipul de aplicație și hardwarul pe care este rulat sistemul. Prin alegerea unui tip de algoritm care nu este potrivit pentru scenariul de utilizare al aplicației, se poate ajunge la un dezechilibru a încărcării sistemului si astfel la o degradare a performanței acestuia.

Cerințe funcționale

Platforma are drept funcționalitate de bază gestionarea de resurse care rulează sub forma de containere Docker, aceasta asigurând scalabilitatea si distribuirea cererilor de la client către acestea.

Printre cerințele funcționale ale platformei se află următoarele:

- Administratorul va putea sa adauge noi tipuri de resurse a căror acces sa fie gestionat de către platformă
- Administratorul va putea să gestioneze algoritmul de load balancing utilizat de către fiecare tip de resursa gestionată
- În cadrul platformei web se vor afișa statusul, utilizarea și numărul de replicări a resurselor gestionate
- Platforma va putea să aloce noi resurse și distruge în cadrul sistemului distribuit în funcție de nivelul de încărcare a acestuia

Aplicații similare

Acest subiect al load balancing-ului între resurse, fiind un subiect important și un aspect absolut necesar pentru aplicațiile și serviciile care rulează la nivel de producție, există câteva soluții care sunt utilizate de regulă în practică. Printre acestea, putem aminti Nginx și serviciile de load balancing oferite de către furnizorii de cloud, cum ar fi AWS Elastic Load Balancer. Cu toate acestea, serviciile amintite au unele limitări și dezavantaje.

În cadrul load balancerului oferit de către AWS, similar celui prezentat în cadrul acestei lucrări, oferă un mecanism de verificare a stării active a unui resurse, adică identifică resursele active prin analizarea răspunsurilor primite la verificări asincrone. Principalul dezavantaj a acestuia însă, este că numărul de configurări disponibile și de algoritmi de load balancing este unul limitat. În cadrul acestuia, singurii algoritmi disponibili sunt round robin și least connections. Un alt dezavantaj este că scalabilitatea trebuie configurată separat nefiind inclusă în serviciul de load balancing, spre deosebire de sistemul propus în cadrul acestei lucrări în cadrul căruia scalabilitate este configurată din start, administratorul putând mai apoi sa ajusteze ponderile și limitele pentru aceasta.

Spre deosebire de serviciul oferit de către AWS, Nginx, un alt load balancer utilizat în producție, are un număr de algoritmi disponibil mai mare, având spre deosebire de acesta și IP Hash și Random. Un dezavantaj al acestuia este că verificare resurselor din cadrul său se realizează pasiv, prin verificarea statusului răspunsurilor primite de la cererile administrate de către acesta, dacă nu există cereri pentru o anumită resursa, sistemul nu are informații cu privire la starea acesteia. Un alt dezavantaj al Nginx, este că pentru fiecare modificare de configurare, adăugare de resursă, sistemul trebuie sa fie repornit, astfel fiind foarte dificil de a se reconfigura fără a avea un timp de inactivitate a întregului sistem.

Sistemul care urmează să fie prezentat în cadrul acestei lucrări rezolvă aceste limitări ale sistemelor prezentate. Acesta oferă o interfață ușoară și intuitivă de configurare, care permite adăugare de noi resurse si noduri de procesare în timp real, gândită de la început pentru a permite scalabilitatea automată a sistemului în funcție de cerințele curente ale acestuia.

Abordarea tehnică

Pentru realizarea interfeței web de control s-a folosit framework-ul React, aceasta este utilizat pentru gestionarea și vizualizarea resurselor gestionate de către platformă.

Componenta centrală de control care are ca si responsabilitate gestionarea si scalabilitatea resurselor precum și load balancing-ul propriu zis este realizată în limbajul Kotlin. O altă componentă din cadrul acestuia, care expune api-ul utilizat de către interfața web este realizată in Ktor. Acesta permite configurarea resurselor și vizualizarea utilizării acestora.

Fiecare nod al sistemului conține un container de control și monitorizare realizat în Kotlin, care expune un api de gestionare a containerelor de pe nodul respectiv și vizualizare utilizării resurselor din cadrul nodului, realizat în Ktor. Acest api este accesat securizat de către unitatea centrală de control pentru a obținerea informații în timp real în legătură cu performanța resurselor de pe nodul respectiv, precum si pornirea sau oprirea acestora.

Containerul de control pentru a putea gestiona containerele Docker, folosește librăria "com.spotify:docker-client", iar pentru a putea monitoriza resursele nodului respectiv pe care rulează, librăria "com.github.oshi:oshi-core".

Pentru persistența datelor de autentificare și a datelor despre resursele gestionate, pentru a putea reface sistemul in caz de eșec, se folosește o bază de date Postgres, care este accesată de către Componenta centrală de control și de către componenta care conține api-ul de control.

Arhitectura sistemului

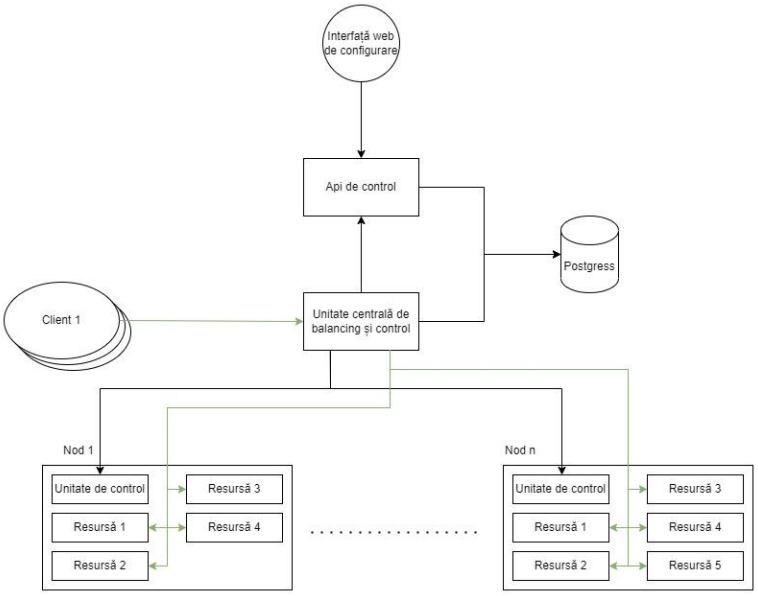


Figura 1: Arhitectura platformei

Sistemul de load balancing și scalling dezvolatat din cadrul acestei lucrări este compus din mai multe componente cu funcții bine definite:

- Interfața web de monitorizare și configurare
- Api-ul interfeței web
- Unitatea centrală de control și gestionare a resurselor
- Unitatea locală fiecărui nod de control și gestionare a resurselor

Interfața web de monitorizare și configurare

Prin intermediul acestei interfețe, administratorul sistemului are posibilitatea de a analiza modul de utilizare a sistemului, de adăugare de noi noduri de procesare sau tipuri de resurse care trebuie administrate. De asemenea acesta interfață web oferă posibilitatea de a adapta în timp real algoritmul care este folosit de către fiecare tip de resursă în parte.

Pagina principală oferă utilizatorului informații de bază cu privire la funcționarea sistemului. Printre aceste informații se află un grafic al numărului de cereri și a timpului mediu de răspuns la acestea, precum și informații în legătură cu statusul curent de replicare a resurselor administrate.

De asemenea interfața web oferă o pagină în care se pot configura parametrii de utilizare, gestionare și sensibilitate a sistemului. Printre acești parametri amintim: intervalul la care se face verificarea stării unui nod, numărul de verificări consecutive eșuate până când un nod este declarat inactiv, ponderea nivelului de utilizare a procesorului sau a procentului de memorie liberă în deciderea resursei care va gestiona o anumită cerere sau a nodului pe care se va rula un anumit tip de resursă, dar și parametrii care controlează modul de funcționare a unui anumit algoritm de load balancing.

O alta pagină din cadrul interfeței web este pagina în care se administrează nodurile din cadrul sistemului, tipul de resurse si algoritmul de load balancing folosit de către fiecare tip de resursă și limitele de scalabilitate a acesteia.

Api-ul interfeței web de monitorizare și contorl

Pentru a vizualiza starea sistemului și gestionarea acestuia în timp real, interfața web se folosește de un api care permite executarea operațiilor expuse prin intermediul interfeței web.

Acest api se folosește de către o bază de date Postgres pentru persistența configurărilor și pentru citirea datelor de utilizare a sistemului care sunt stocate în aceasta.

Acest server de gestionare oferă de asemenea un endpoint care expune un uuid care se generează la fiecare pornire a acestuia și câte o versiune a configurărilor sistemului, nodurilor care fac parte din acesta și a resurselor care trebuie gestionate, versiuni care sunt incrementate la fiecare modificare a caracteristicilor asociate.

Componenta centrală de balancing, control și gestionare a resurselor

Aceasta este componenta centrală din cadrul sistemului care se ocupă cu realizarea procesului propriu-zis de load balancing și de asemenea de controlul resurselor administrate și asigurarea funcționării lor.

Pentru asigurarea corectă a sistemului, această componentă are trei module care rulează la intervale fixe de timp:

- HealthChecker: verifică pentru fiecare nod din sistem resursele care rulează pe acesta și menține procentul de utilizare a procesorului și a memoriei acestora. De asemenea dacă nodurile nu răspund la un anumit număr de verificări consecutive, acest modul le mută în statusul de inactiv, eliminându-le din cadrul sistemului
- DeploymentsManager: se asigură că toate resursele care trebuie administrate de către sistem sunt active și se ocupă cu scalling-ul lor în funcție de utilizare, folosind datele obținute de către HealthChecker
- MasterChangesManager: Acest modul interoghează api-ul interfeței web și aplică în cadrul sistemului schimbările de configurare

Pentru procesarea și delegarea cererilor în cadrul sistemului de load balancing, către resursa corespunzătoare, această componentă citește din socket-ul clientului antetul cererii pentru identificarea tipului de resursă căreia îi este destinată, apoi selectează o resursă dintre cele disponibile pentru tipul respectiv folosind algoritmul de load balancing configurat pentru aceasta și deschide un socket către resursa selectată în care scrie antetul cererii originale din care elimină sub calea configurată pentru resursa respectivă în cadrul sistemului și restul conținutului cererii. După aceasta așteaptă ca resursa sa proceseze resursa respectivă și scrie răspunsul primit de la aceasta prin socket-ul deschis către resursă în socket-ul original de la client.

Componenta locală fiecărui nod de monitorizre și gestionare a resurselor

Această componentă de monitorizare si gestionare rulează pe fiecare nod din cadrul sistemului și expune un api care este utilizat de către componenta centrală de control pentru analizarea și controlul sistemului.

Acest api expune un endpoint pentru crearea și ștergerea de containere care este folosit de către componenta centrală în mecanismul de gestiune a consistenței datelor și în mecanismul de scalling automat a resurselor.

De asemenea acest api expune un endpoint care oferă informații legate de containerele care rulează pe nod și au fost pornite de către acesta. Aceste informații conțin date despre procentul de utilizare a procesorului și a memoriei alocate fiecărui container. Pe lângă aceste informații, oferă date legate de utilizare procesorului și a memoriei nodului. Aceste informații sunt utilizate de către sistemul de scalling din cadrul unității centrale de control, datele fiind colectate prin intermediul sistemului de verificare al nodurilor care rulează pe acesta.

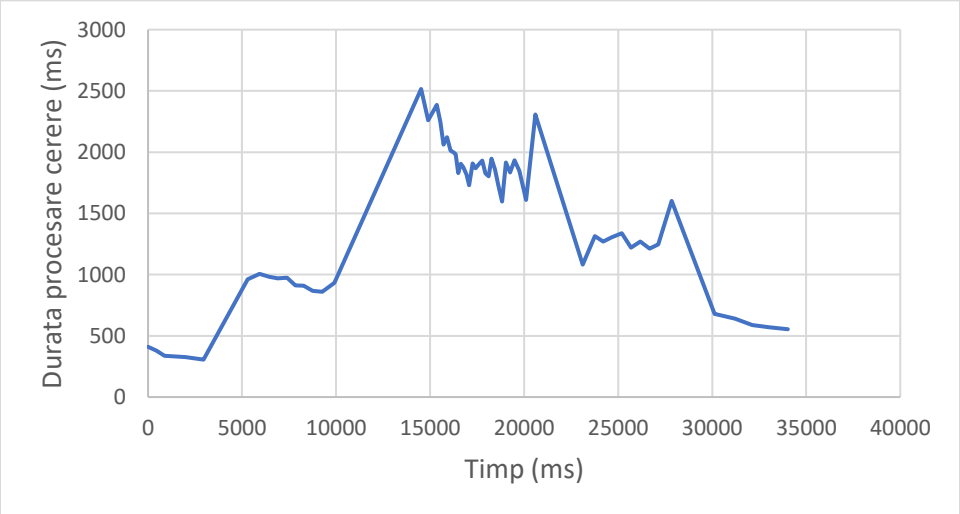
Verificare sistemului

Pentru verificarea sistemului s-a folosit un script care pornea un client care genera 500 de cereri in paralel, după care pornea 2 clienți în paralel care generau fiecare câte 500 de cereri în paralel. Apoi când aceștia terminau se lansau în paralel 6 clienți, fiecare dintre aceștia generând în paralel 500 de cereri. Când aceștia terminau, se lansa încă un client care genera 500 de cereri în paralel. Toate aceste cereri din cadrul testului erau destinate către același tip de resursă gestionată de către sistem, măsurându-se decalajul de răspuns al sistemului. Resursa folosită în cadrul testului a avut 4 instanțe care erau distribuite pe 2 noduri.

Clientul folosit în cadrul testului genera un număr aleator între 0 și 5000, care reprezenta numărul de secunde minime care trebuie sa dureze cererea către resursa din cadrul sistemului. Acesta pentru a simula o încărcare a sistemului pe acea durată, a rulat o buclă care se execută cât timp a fost setat pentru cererea respectivă.

Performanta algoritm round robin

În cadrul algoritmului de load balancing round robin, cererile de la client sunt distribuite către resursele care le pot procesa într-o ordine ciclică, bazat pe ordinea în care au ajuns la componenta centrală de procesare. Acest algoritm funcționează cel mai bine atunci când resursele de procesare au putere de procesare și memorie similară.

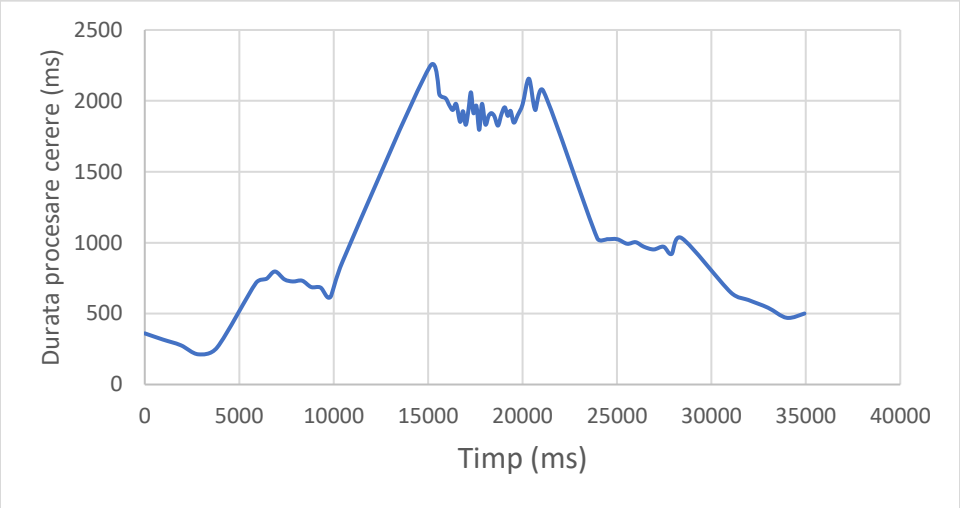


Diagramă 1: Performanța algoritmului round robin

Performanta algoritm randomized

Algoritmul Randomized redirectează o cerere în mod aleatorde la un client în mod aleator la una dintre resursele de procesare care pot să servească cererea respectivă. Acest algoritm

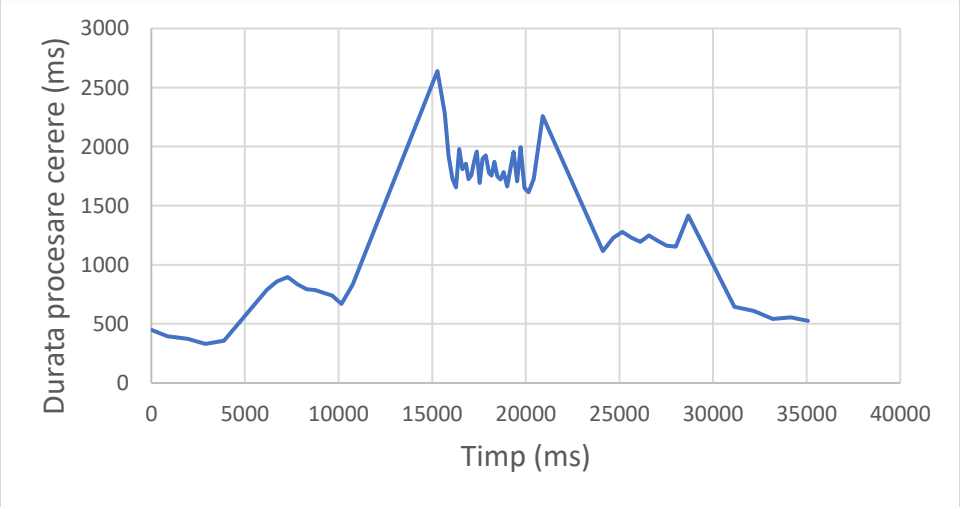
funcționează bine atunci când durata de procesare a cererilor de către resursele de procesare este echilibrată.



Diagramă 2: Performanța algoritmului randomized

Performanta algoritm least connection

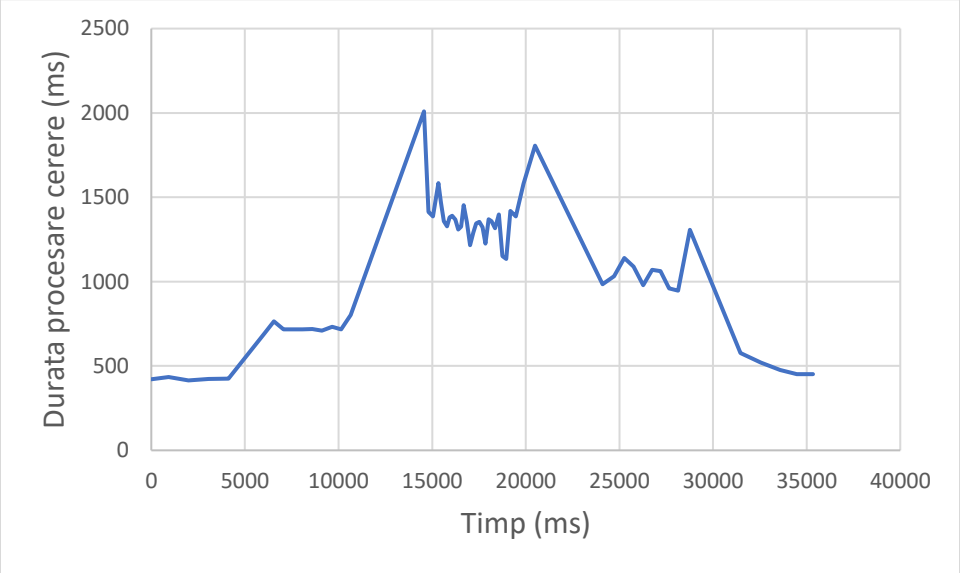
Algoritmul de load balancing least connection ține cont de numărul de conexiuni active redirectând cererile către resursa de procesare cu cele mai puține cereri active în momentul în care primește cererea. Acest algoritm se recomanda a fi utilizat atunci când unele cereri durează mai mult pentru a fi procesate comparativ cu altele, lucru care poate duce la suprasolicitarea unei anumite resurse de procesare în cadrul altor algoritmi cum ar fi round robin sau randomized.



Diagramă 3: Performanța algoritmului least connections

Performanta algoritm weighted response time

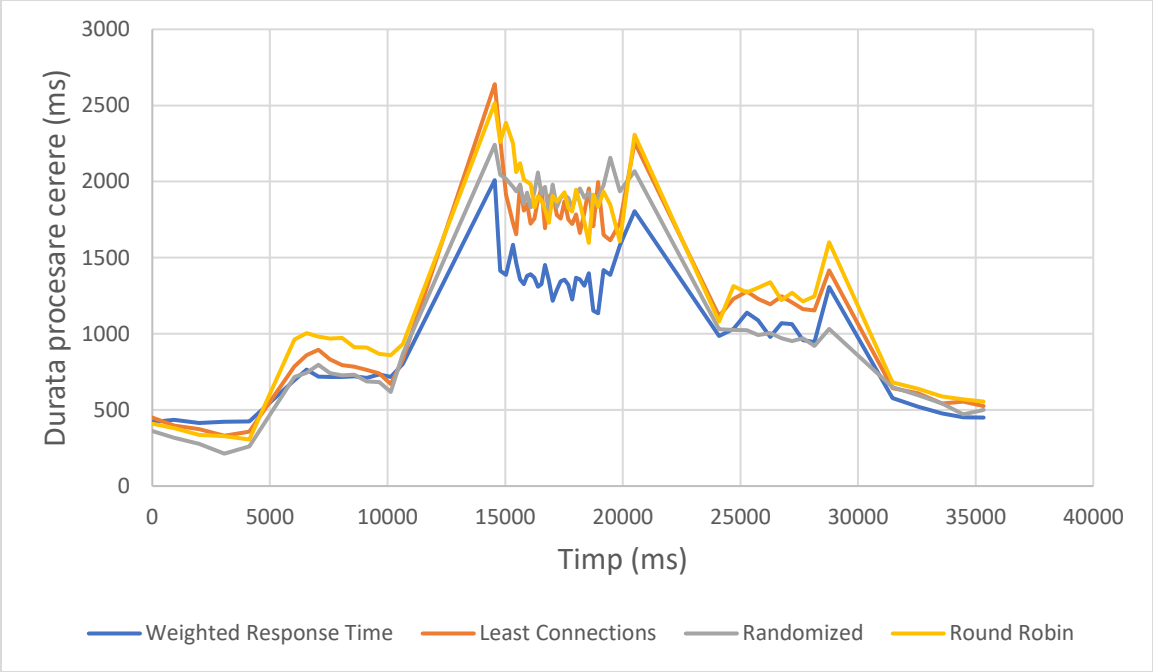
Algoritmul de load balancing weighted response time, redirectează următoarea cerere de la client către resursele care pot procesa cererea respectivă, fiecare resursa având o anumită pondere în funcție de timpul de răspuns la cererile anterioare. Acest algoritm adaptează ponderile în timp real în funcție de performanța efectivă de procesare a resurselor. Un dezavantaj al acestuia este memoria mai ridicată folosită comparativ cu ceilalți algoritmi, deoarece trebuie sa rețină timpii de răspuns anteriori pentru calcularea ponderilor.



Diagramă 4: Performanța algoritmului weighted response time

Compararea performanței algoritmilor

Din datele obținute în cadrul testelor se poate observa că algoritmii s-au comportat diferit în funcție de numărul de numărul de cereri executate în paralel. Pentru un număr relativ mic de cereri în paralel, algoritmul randomized si round robin se pare că au avut cele mai bune rezultate. Odată cu creșterea numărului de cereri, algoritmul Weighted Response Time a obținut performanțe mult mai bune comparativ cu ceilalți algoritmi. De asemenea trebuie amintit memoria mai ridicată necesară de către acesta prin comparație cu ceilalți algoritmi, care poate fi un factor important pentru alegerea unui anumit tip de algoritm in defavoarea celuilalt.



Diagramă 5: Compararea performanței algoritmilor

Concluzii

Utilizarea sistemelor de load balancing este importantă pentru fiecare site sau api care dorește să ofere utilizatorilor săi o experiență de utilizare fluidă și rapidă. Utilizarea unui mecanism de load balancing și scalling aduce numeroase avantaje printre care se numără o reducere a costurilor de rulare a aplicației respective, deoarece la orice moment rulează exact atâtea resurse cât sunt necesare, o creștere a fiabilității aplicației, deoarece traficul este redirectat numai către noduri care funcționează. De asemenea utilizarea unui astfel de sistem favorizează utilizarea a mai multor servere cu specificații mai scăzute și mai ieftine în defavoarea unui singur sistem foarte puternic și cu costuri ridicate, sistemul redirectând în mod echilibrat cererile în cadrul sistemului de servere.

După cum se poate observa din cadrul acestei lucrări, configurarea și alegerea corectă a algoritmului de load balancing poate avea o influență mare asupra performanței sistemului. Astfel, fiecare algoritm este mai potrivit pentru un anumit scenariu de utilizare, un anumit numărului de utilizatori activi simultan sau pentru o anumită cantitate de memorie alocată pentru mecanismul de balancing.

Referințe

[1] A. D. S. Soumya Ray, „Execution analysis of load balancing algorithms in cloud computin environment,” 2012. [Interactiv]. Available: https://www.academia.edu/14677381/EXECUTION_ANALYSIS_OF_LOAD_BALANCING_ALGORITHMS_IN_CLOUD_COMPUTING_ENVIRONMENT.

[2] A. G. Payal Beniwal, „A comparative study of static and dynamic Load Balancing Algorithms,” 2014. [Interactiv]. Available: https://www.researchgate.net/profile/Atul-Garg-3/publication/270728037_A_comparative_study_of_static_and_dynamic_Load_Balancing_Algorithms/links/54c709d00cf289f0ceccc809/A-comparative-study-of-static-and-dynamic-Load-Balancing-Algorithms.pdf.

[3] S. S. M. S. Sandeep Sharma, „Performance Analysis of Load Balancing Algorithms,” 2008. [Interactiv]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.307.1711&rep=rep1&type=pdf.

[4] S. A. Abhijit A. Rajguru, „A Comparative Performance Analysis of Load Balancing Algorithms in Distributed System using Qualitative Parameters,” 2012. [Interactiv]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.682.9450&rep=rep1&type=pdf.

[5] H. E. B. M. Z. D. E. K. A. Khiyaita, „Load balancing cloud computing: State of art,” 2012. [Interactiv]. Available: https://ieeexplore.ieee.org/abstract/document/6249253.

[6] D. C. T. Deepa, „A comparative study of static and dynamic load balancing algorithms in cloud computing,” 2017. [Interactiv]. Available: https://ieeexplore.ieee.org/abstract/document/8390086.

[7] P. M. Bhavsar, „Load Balancing in Grid Environment using Machine Learning - Innovative Approach,” [Interactiv]. Available: https://www.researchgate.net/profile/Ashish-Revar/publication/49586510_Load_Balancing_in_Grid_Environment_using_Machine_Learning_-_Innovative_Approach/links/00b49522c9d322e892000000/Load-Balancing-in-Grid-Environment-using-Machine-Learning-Innovative-Appr.

[8] B. P. Smaranika Parida, „An Efficient Dynamic Load Balancing Algorithm Using Machine Learning Technique in Cloud Environment,” 2018. [Interactiv]. Available: https://fardapaper.ir/mohavaha/uploads/2018/08/Fardapaper-An-Efficient-Dynamic-Load-Balancing-Algorithm-Using-Machine-Learning-Technique-in-Cloud-Environment.pdf.

[9] H. Yao, X. Yuan și P. Zhang, „Machine Learning Aided Load Balance Routing Scheme Considering Queue Utilization,” 2019. [Interactiv]. Available: https://ieeexplore.ieee.org/abstract/document/8733872.