Departamentul Automatică şi Informatică Industrială
Facultatea Automatică şi Calculatoare
Universitatea POLITEHNICA din Bucureşti

AUTOMATICĂ ŞI INFORMATICĂ INDUSTRIALĂ

RAPORT ŞTIINŢIFIC NR. 1

RAPORT DE CERCETARE PENTRU SEMESTRUL I

Eduard-Ştefan POPA

Conducător ştiinţific de doctorat: Conf. dr. ing. Mihnea Alexandru MOISESCU

Tema: Stress Testing pentru platforme web

Program de masterat: Managementul şi Protecţia Informaţiei

Cuprins

1 Rezumat

În acest raport de cercetare sunt cuprinse informații cu referire la metodologia de testare software a aplicațiilor în vederea asigurării calității acestora, discuția urmând să fie particularizată pe abordarea tehnicii de Stress Testing.

Lucrarea de față debutează cu o introducere generală asupra testării software, fiind prezentată în primul rând o justificare a necesității implementării acestor tehnici însoțită de câteva exemple concrete de incidente care au avut loc din cauza inexistenței acestora. De asemenea, practica testării aplicațiilor implementate pentru utilizare la nivel larg are la bază și interesele financiare ale organizației, deoarece o problemă semnalată înainte de lansarea produsului pe piață va implica costuri mai mici de rezolvare a bug-ului. Pentru a finaliza această introducere se descrie pe scurt clasificarea modalităților de testare în cele două modele de bază cunoscute ("white box" și "black box") după felul în care specialistul care concepe testele vizualizează întregul sistem informatic.

După această introducere este urmărită punctarea obiectivelor tehnicii de testare de tip Stress Testing. Se menționează astfel de ce această metodă de asigurare a calității poate avea un mare impact asupra clienților, dar și cum poate ajuta la identificarea anumitor breșe de securitate care pot pune în pericol infrastructura organizației. În următoarea fază vor fi prezentate câteva aplicații de Stress Testing implementate anterior, aspect ce va reprezenta un punct de plecare în proiectarea unei aplicații proprii.

În lucrare este prezentat un articol care aparține Universității Texas din Arlington care este o instituție puternic orientată pe partea de cercetare în domeniul ingineriei software. În acest articol se fac mențiuni asupra structurii framework-urilor de Stress Testing automate prin exemplificarea unui studiu de caz concret. Aceste framework-uri de testare automată sunt foarte utile în producție, având mai multe avantaje:

- sunt utile pentru repetarea testelor
- ajută la testarea aplicațiilor implementate pe mai multe configurații
- permite rularea de teste pentru un cod care se schimbă des
- pot fi rulate pe mai multe mașini și astfel scade timpul de testare

Studiul de caz precizat anterior constă în testarea unei aplicații de comerț electronic cu un volum mare de cereri HTTP pentru a determina performanțele aplicației prin identificarea unui model de performanță. Aplicația testată este abstractizată cu ajutorul unui model bazat pe două nivele de cozi pentru cereri. Cazurile de testare vor genera răspunsuri din partea aplicației supuse traficului ridicat care sunt analizate în vederea îmbunătățirii acestor cazuri care vor ajuta la calcularea parametrilor de interes precum numărul maxim de utilizatori care pot accesa aplicația simultan sau timpul de răspuns.

Finalitatea acestei activități de cercetare o constituie implementarea unei astfel de aplicații de Stress Testing, urmată de validarea funcționalităților prin testarea unor scenarii diverse și interpretarea rezultatelor obținute.

2 Introducere în tematica abordată

2.1 Introducere în testarea software

Testarea este definită ca fiind o serie de metode prin care se verifică faptul că un produs informatic îndeplinește toate cerințele de funcționare în mod corespunzător. Pentru realizarea acestei proceduri au fost implementate un număr ridicat de tool-uri de testare a componentelor software, dar și hardware.

Practica testării componentelor informatice este foarte importantă, deoarece o detectare timpurie a anumitor defecțiuni în funcționare, de preferat înainte de a ajunge pe piață spre utilizare, poate scuti compania producătoare de diverse probleme în relația cu clienții, dar și de pierderi de ordin financiar. În plus, anumite bug-uri care nu sunt identificate la timp se pot dovedi periculoase pentru utilizatori. Mai jos sunt prezentate sumar câteva cazuri de bug-uri care au generat mari probleme:

- În anul 2015 aproximativ 60% din magazinele Starbucks de pe teritoriul Statelor Unite ale Americii și al Canadei au fost închise din cauza unei defecțiuni la sistemul POS care nu permitea efectuarea tranzacțiilor bancare.
- Compania Nissan a fost nevoită să retragă peste un milion de mașini de pe piață din cauza unei erori software la sistemul de senzori al airbag-urilor.
- În luna mai a anului 1996 un bug la sistemul bancar al unei bănci din Statele Unite ale Americii a cauzat depunerea unei sume totale de 920 de milioane de dolari americani în conturile a 823 de persoane.
- În Canada, în anul 1985, masina Therac-25 destinată radioterapiei a funcționat necorespunzător din cauza unui bug, administrând o cantitate de radiații cu mult peste valoarea recomandată care a culminat cu 3 decese.

Aceste situații prezentate mai sus sunt total nedorite și tocmai de aceea implementarea unor teste de funcționalitate este vitală pentru asigurarea siguranței în utilizare a acestor produse. În principiu, finalitatea acestor testări cuprinde patru mari ramuri:

1. reducerea costurilor de implementare și debugging a aplicațiilor
2. securitatea datelor în vederea realizării unei relații de încredere cu utilizatorii
3. îndeplinirea tuturor cerințelor specificate în planul proiectului
4. satisfacerea nevoilor clienților

Paradigma testării sistemelor informatice este împărțită în două abordări care au rolul de a descrie care este punctul de vedere al persoanei responsabile de modelarea cazurilor de testare. Astfel, la ora actuală există două mari modele de testare numite “white-box” și, respectiv, “black-box”.

Modelul “white-box” se axează pe structura codului sursă a aplicației dezvoltate ca punct de plecare pentru formularea cazurilor de testare. Se aleg diverse valori de intrare și se urmărește evoluția acestora pe măsură ce codul este executat. În această categorie se încadrează testarea unitară (Unit Testing) care analizează bucăți din cod pentru testarea corectitudinii prin compararea output-ului generat de o intrare cu o valoare care este cunoscută ca fiind valoarea adevărată de ieșire corespunzătoare intrării selectate.

Diametral opus modelului prezentat mai sus, abordarea de tip “black-box” abordează sistemul ca o cutie neagră care nu oferă acces la structura sa internă, în acest caz codul sursă. În această situație se urmărește ce face sistemul, ci nu cum. Un caz interesant de testare care intră în această tipologie

este testarea de tip Fuzzy care presupune generarea de intrări aleatoare și urmărirea comportării sistemului. Mai exact se dorește să se verifice dacă sistemul va pica din cauza unor anumite valori ale input-ului sau dacă vor apărea anomalii de tip memory leaks.

2.2 Formularea problemei

Testarea software este un domeniu ce a apărut din rațiuni de afaceri pentru asigurarea calității sistemelor informatice administrate de diverse organizații. Companiile care dețin aplicații web intens folosite pot pierde o sumă considerabilă de bani dacă platformele ajung în situația de a fi indisponibile ca urmare a unei blocări generate de un trafic extrem de ridicat primit de către servere.

Din această cauză s-a dezvoltat tehnica Stress Testing care are rolul de a testa comportamentul unei platforme în condiții grele de funcționare. Mai precis, acest tip de testare non-funcțională solicită o aplicație web cu un număr mare de cereri de tip HTTP pentru a afla, de exemplu, care este numărul maxim de cereri primite într-o perioadă de timp mica astfel incat aplicația să răspundă în parametri normali. Cel mai simplu și intuitiv exemplu de parametru urmărit este timpul de răspuns al serverului la cererea trimisă de browser. Cu siguranță niciun utilizator sau potențial client nu dorește să aștepte un timp prea mare pentru a primi informația de care are nevoie. Acest fapt este cauzat de capacitatea scăzută de procesare a aplicației în raport cu numărul de cereri primit.

Aceste aplicații de testare nu se opresc la a monitoriza doar timpul de răspuns al serverului. Un alt motiv pentru care se implementează astfel de teste este pentru a verifica dacă sistemul poate să salveze datele sau nu înainte ca sistemul informatic să devină indisponibil. Este esențial ca o aplicație să asigure persistența datelor chiar dacă sistemul este dus în punctul de a suferi o cădere în fața unui trafic foarte mare, deoarece în caz contrar este afectată încrederea utilizatorului în organizație, deci tot din motive financiare. Și mai important, nu numai încrederea utilizatorului este implicată în cazul unor aplicații care prezinta diverse implicații legale si juridice cum ar fi aplicațiile bancare. În astfel de cazuri este vitală implementarea unor infrastructuri puternice în fața unor astfel de potențiale traficuri care apar aproape zilnic. [1]

Un alt scop al acestui tip de testare este asigurarea faptului că o eventuală apariție a unor astfel de condiții solicitante nu determină generarea unor probleme de securitate sau nu fac evidente anumite vulnerabilități deja existente. Există și situația în care traficul ridicat recepționat este cauzat de un atac de tip Distributed Denial of Service, prescurtat DDos, prin care un atacator se folosește de mai multe sisteme de calcul compromise pentru a genera un număr ridicat de conexiuni cu serverul atacat. Scopul acestor atacuri este de a face inoperabile diverse aplicații care găzduiesc servicii importante precum cele bancare, de email, de comerț electronic și așa mai departe.

De asemenea, există bug-uri în aplicații care nu pot fi detectate decât dacă sistemul nu este supus unor condiții de lucru dificile. Aceste bug-uri se pot concretiza în probleme de sincronizare, memory leaks sau race conditions. Race conditions este în sine o problemă de sincronizare între două fire de execuție, de exemplu, care au acces la o resursă partajată pe care incearca sa o modifice, dar operațiile de citire-scriere care aparțin celor două fire de execuție nu se execută secvențial ci intercalat, apărând rezultate neașteptate și nedorite la finalizarea execuției. Acest tip de bug poate deveni o vulnerabilitate când afectează un mecanism de control al securității. În acest sens, atacatorii pot genera acțiuni înainte de finalizarea acestor tipuri de controale de securitate.

Componentele hardware cele mai afectate în timpul realizării de Stress Testing sunt memoria si microprocesorul. Urmările acestei proceduri includ modificări ale parametrilor de funcționare ale componentelor, cum ar fi temperatura, variația frecvenței de ceas și a voltajului, așadar este necesară asigurarea alegerii unor componente potrivite pentru această acțiune. Verificarea se poate realiza prin

punerea memoriei într-un regim de funcționare ce presupune utilizarea în întregime a acesteia pe o perioadă de 24 de ore înainte de realizarea procedurii de testare.

Luând toate aceste aspecte în considerare, Stress Testing-ul devine o necesitate pentru asigurarea funcționării corecte ale aplicației în orice condiții și, în plus, pentru detectarea unor potențiale breșe de securitate ce pot pune în pericol siguranța în utilizare a produsului software.

2.3 Studiu asupra realizărilor similare din domeniu

Apache JMeter

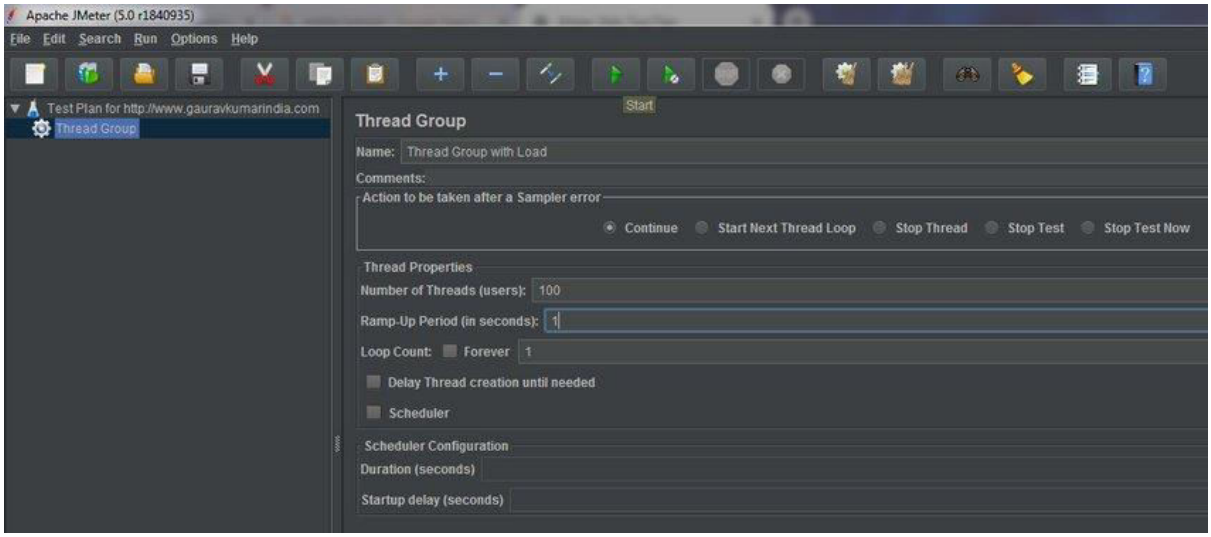

Figura 1. Interfața aplicației Apache JMeter [2]

Apache JMeter este o aplicație desktop de tip open source implementată în limbajul Java, utilizand framework-ul Swing pentru dezvoltarea interfeței. Acest software are rolul de a determina dacă o anumită aplicație web poate satisface cerințe de funcționare când este supusă unor condiții dificile, masurându-se performanțele acesteia. De exemplu timpul de răspuns pentru un anumit număr de utilizatori sau, mai important, numărul de cereri procesate în unitatea de timp, mărime numită debit. Alți parametri de urmărit sunt gradul de ocupare al memoriei, microprocesorului, adică gradul de exploatare al resurselor fizice de care dispune serverul. Avantajele pe care Apache JMeter le prezintă sunt următoarele:

- Este o platformă open source, deci este gratuită
- Are o interfață intuitivă, ușor de folosit
- Fiind o aplicație creată exclusiv în Java, este independentă de platformă
- Permite multithreading cu scopul simulării unor utilizatori care trimit simultan cereri către server
- Are suport multiprotocol (HTTP, JDBC, SOAP, FTP), putând fi utilizat în testarea bazelor de date
- Permite crearea de planuri de testare

Pentru specificarea unui plan de testare, utilizatorul poate menționa în primul rând detalii în legătură cu firele de execuție ce vor ține loc de utilizatori multipli, precum: numărul de thread-uri

(număr de utilizatori simulați), perioada de ramp-up (este un interval de timp între trimiterea a două cereri succesive) și numărul de bucle (de câte ori se va executa testul).

În etapa următoare se poate proceda în două moduri:

1. Se specifică URL-ul către pagina web ce urmează a fi supusă testării
2. Se specifică numele serverului sau adresa IP a acestuia alături de portul pe care acesta rulează

După ce aceste etape au fost parcurse se pot rula testele, fiind afișate răspunsurile primite în urma cererilor trimise și grafice care să ilustreze evoluția parametrilor ce descriu performanțele aplicației web (se poate alege să se adauge pe grafic diverse mărimi statistice precum medie sau deviație standard).

LoadRunner

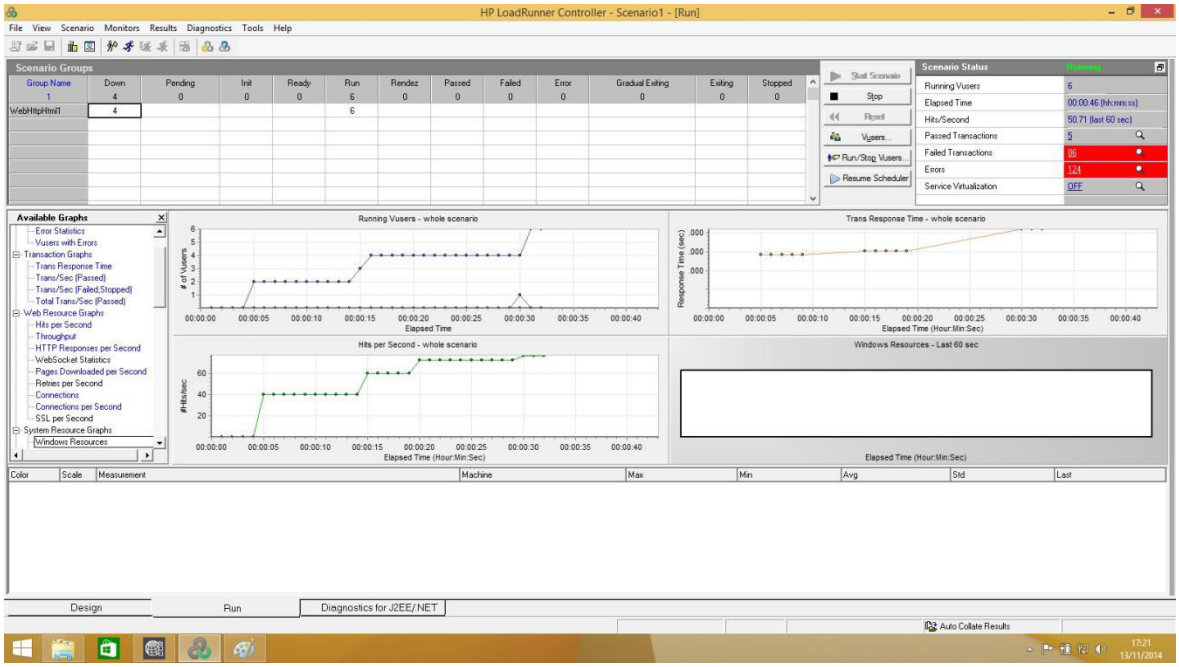

Figura 2. Interfața aplicației LoadRunner [3]

LoadRunner este o aplicație implementată în scop industrial de către compania Hewlett-Packard pentru examinarea comportării sistemelor informatice în condiții solicitante de funcționare. Din punctul de vedere al arhitecturii, LoadRunner este compusă din trei aplicații mai mici:

1. Generatorul de utilizatori virtuali permite scrierea de scripturi ce pot genera o serie de acțiuni din partea unor utilizatori virtuali. În general pentru crearea unui utilizator virtual sunt necesari dintre 2 și 4 MB de date.
2. Controllerul preia scripturile de la Virtual User Generator și le rulează după o anumită planificare setată. Această componentă se ocupă de monitorizarea activității utilizatorilor virtuali, putând să activeze un număr specificat de utilizatori la un moment de timp bine determinat. Pentru simularea acestor utilizatori, partea de controller utilizează mecanisme de tip IP Spoofing prin crearea de pachete IP ce conțin în partea de header adrese IP false pentru sursa pachetului.

3. Programul de analiză afișează rezultatele analizei sub diverse forme, indicând punctele vulnerabile ale aplicației web care pot genera blocaje.

Neoload

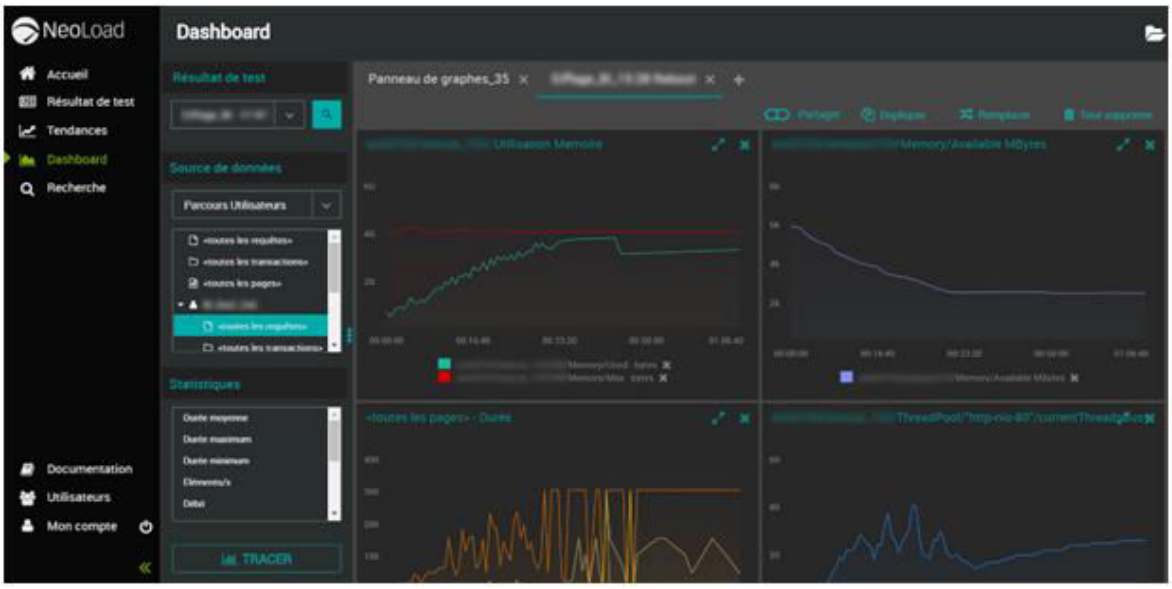

Figura 3. Interfața aplicației Neoload [4]

Neoload este o aplicație de testare software a platformelor web și mobile. Asemănător aplicațiilor mai sus menționate, Neoload funcționează prin simularea de trafic din partea unui număr de utilizatori de ordinul milioanelor. Neoload este folosit pentru monitorizarea unei game de tipuri de servere, precum: .NET, Oracle, JBoss, JOnAS, Tomcat, WebLogic, GlassFish, WebSphere sau NetWeaver. Tehnologiile utilizate pentru implementarea acestei aplicații sunt:

- AngularJS este un framework pentru limbajul JavaScript folosit pentru implementarea aplicațiilor web dinamice. Comunicarea dintre server și browser este facilitată de obiecte de tip XMLHttpRequest sau JSON. Aplicațiile dezvoltate în framework-ul Angular sunt simplificate foarte mult în termeni de dezvoltare și testare pe partea cu clientul, bazându-se pe arhitecturi de tipul MVC (model-view-controller).
- Websocket este un protocol ce permite conexiuni persistente de tip client-server, fiind foarte potrivit pentru aplicații cu latență scăzută. Acest protocol este utilizat, deoarece are ca capabilități de monitorizare a comunicației. Prin intermediul Websocket aplicația are acces la statistici despre conexiunile cu serverul testat (gradul de încărcare al memoriei, microprocesorului) pe Windows, Linux sau Solaris.
- Java 2 Platform Enterprise Edition (J2EE) este o extindere pentru Java 8 dedicată organizațiilor pentru partea de calcul distribuit și servicii web. De asemenea, J2EE permite controlul cookie-urilor de sesiune. Pentru partea cu serverul a aplicațiilor, platforma J2EE suportă două tehnologii web: Java Servlets și JavaServer Pages. Servlet-urile sunt programe care permit prelucrarea cererilor primite de la serverul web în vederea obținerii unui răspuns care este trimis înapoi la serverul web și apoi la clientul care a realizat cererea. Avantajul servlet-urilor este capacitatea lor de a opera cu cereri de complexitate ridicată. JavaServer

Pages combină partea de interfață vizuală implementată în limbajul HTML cu partea de funcționalitate a servelet-urilor.

- AJAX (Asynchronous JavaScript and XML) este o variantă de implementare a paginilor web dinamice ce permite introducerea de conținut de către utilizator fără a fi necesară reîncărcarea paginii. Astfel, este permisă o interacțiune simplificată între browserul web și serverul aplicației, acesta fiind motivul pentru care utilizatorul poate face modificări asupra paginii fără reîncărcarea acesteia pentru transmiterea datelor. Interacțiunile AJAX sunt inițiate printr-un cod scris în limbajul JavaScript care manipulează codul HTML. Acțiunile AJAX sunt foarte utile în cazul validării de formulare în timp ce utilizatorul încă face completări, primirii datelor de la server și al modificării de date în mod dinamic pe pagina web.

Asemănări și deosebiri între Apache JMeter, LoadRunner și Neoload

Tabelul 1. Sinteza caracteristicilor unei aplicații de Stress Testing [5]

	Apache JMeter	**LoadRunner**	**Neoload**
Gratuitate	Software open source	Necesită licență și nu este gratuit, iar varianta full este foarte scumpă	Necesită licență și nu este gratuit
Suport pentru protocoale	Prezintă suport pentru anumite protocoale	Are cel mai mare suport pentru protocoale	Prezintă un suport pentru un număr mare de protocoale
Limbajele utilizate în implementare	Implementat 100% în limbajul Java și de aceea este portabil pe Windows, Mac, Linux	Folosește limbajul Java alături de JavaScript și Visual Basic	Partea de backend este implementată în Groovy și Java
Capacități și limitări	Are capacitate limitate. Pe o masina cu multe resurse (memorie RAM și procesor) poate simula maxim 2000 de utilizatori. O depășire a acestei valori poate conduce la excepții de memorie	Capacitatea este superioară aplicației Apache JMeter	Are o capacitate foarte mare, permițând simularea a milioane de utilizatori virtual (varianta gratuită permite maxim 50 de utilizatori)

Rulare în medii Cloud	Are anumite dificultăți în a fi rulat în Cloud	Este rulat cu ușurință într-un mediu Cloud	Soluția software rulează în Cloud, utilizând un număr mare de furnizori (Amazon EC2, Rackspace, Cloud Sigma), dar LoadRunner este mai dezvoltat pe această parte
Statistici oferite în timp real	Nu poate oferi rezultate și statistici în timp real	LoadRunner oferă rezultate destul de apropiate de timpul real	Oferă monitorizare în timp real a conexiunilor
Scopul aplicației	Nu oferă eficiența necesară utilizării în cadrul companiilor (utilizare personală)	Este conceput special pentru industrie	Conceput pentru industrie, dar nu obligatoriu
Suport pentru sisteme de certificare	Prezintă suport pentru scheme de autentificare (Basic, Message Digest, Kerberos) și certificate de clienți	Prezintă suport pentru scheme de autentificare (Basic, Message Digest, Kerberos) și certificate de clienți	Prezinta suport pentru scheme de autentificare (Basic, Message Digest, Kerberos) și certificate de clienți
Suport specific sistemelor IoT	Prezintă un plugin MQTT pentru a realiza partea de testare a performanțelor aplicațiilor IoT	Suport pentru IoT introdus recent	Suport complex pentru platforme IoT (MQTT, JSMS, SOAP, RESTAPI)

3 Implementarea unui framework de Stress Testing

3.1 Tehnologii de dezvoltare

Există un număr foarte mare de parametri care influențează rezultatele furnizate de orice aplicație de testare a performanțelor oferite de platformele web în condiții solicitante. Unul dintre aceștia îl reprezintă arhitectura sistemului informatic testat. De obicei, rețelele în care sunt integrate serverele ce urmează a fi supuse testării sunt completate de diverse echipamente hardware, precum: firewall, rutere sau load balancere. Firewall-urile pot determina apariția de blocaje, deoarece acestea trebuie să evalueze în continuu și să filtreze traficul pe care îl primește, astfel suportând un debit de cereri mai scăzut mai ales în cazul cererilor simultate generate de utilizatori, aspect luat în considerare de aplicațiile de Stress Testing prin implementarea instanțelor cunoscute sub denumirea de utilizatori virtuali. O bună practică este reprezentată de realizarea de testări pe diferite browsere, nefiind suficientă lansarea de teste pe un singur browser. Acest aspect este justificat de faptul că fiecare browser tratează în mod diferit conexiunile slabe, web cache-urile și criptarea prin protocolul SSL. [6]

În implementarea unor astfel de tool-uri de testare este necesară evaluarea instrumentelor software ce urmează a fi utilizate din prisma particularităților pe care acestea le prezintă și a specificațiilor prevăzute în cerințele funcționale. O alternativă de proiectare a unui astfel de tool este prezentată mai jos:

- utilizarea tehnologiei J2EE care este considerată o arhitectură foarte scalabilă și performantă. Aceasta prezintă capabilități de multithreading care este ideală pentru simularea de utilizatori virtuali. J2EE este implementată având un "garbage collector" care este utilizat în eliberarea automată a memoriei care deseori implicată în probleme de tipul memory leaks, memory overflow care pot genera breșe de securitate. Ca limbaj de implementare, Java se pretează foarte bine la această arhitectură și permite sincronizarea de procese pentru a evita apariția de deadlock-uri ce pot duce la indisponibilitatea sistemului.
- documentele în format XML (Extensible Markup Language) sunt foarte utilizate datorită proprietăților de stocare, procesare, extragere de date. Sunt compatibile cu o gamă de limbaje de programare precum C#, Java, Php, Python .etc și pot fi parsate prin sintaxa oferită de XPath care este foarte rapidă.
- bazele de date prezintă capabilități mai mari de stocare a datelor față de fișierele XML mai ales când se lucrează cu volume imense de date. Este o resursă indispensabilă în cadrul serverelor și aplicațiilor, dar este totodată un punct vulnerabil din punctul de vedere al securității aplicației. O soluție pentru reducerea traficului este utilizarea de cereri SQL precompilate (proceduri stocate). De asemenea, din cauza criptării și decriptării datelor în baza de date, timpului de acces la date i se adaugă o întârziere provocată de aceste măsuri de securitate.
- conexiunea la rețea este punctul central în ceea ce privește asemenea tool-uri, deoarece fără o conexiune stabilă la Internet nu se poate menține pretenția obținerii de rezultate în concordanță cu realitatea. În general pentru aplicațiile distribuite intervalul de timp dintre lansarea unei cereri și primirea unui răspuns este considerabil de ridicat. Comunicarea prin lansarea de cereri trebuie realizată printr-un protocol corespunzător. De exemplu, protocolul SOAP este mai lent decât protocolul HTTP.
- pe partea de frontend există foarte multe alternative pentru implementarea interfeței grafice cu utilizatorul. Astfel, se poate opta pentru o aplicație desktop sau o aplicație web. Pentru o aplicație desktop se pot folosi limbajele Java (framework-ul Swing sau JavaFX), C# sau chiar Python, în timp ce pentru alternativa web există framework-urile Angular (sau AngularJS),

React, Vue și Django (aparține de limbajul Python) împreună cu tehnologiile uzuale: HTML, CSS, JavaScript.

- componenta securității aplicației nu trebuie neglijată. În acest sens este necesară dotarea serverelor aplicației cu un firewall care să permită analizarea și filtrarea traficului cu potențial malițios. Pentru partea de criptare și, respectiv, decriptarea este extrem de utilizată la ora actuală algoritmul RSA bazat pe descompunerea numerelor mari în produs de două numere prime.

3.2 Structura unui framework de testare automată

Unul dintre cele mai importante obiective ale Stress Testing-ului îl constituie determinarea sarcinii necesare cu care trebuie încărcate serverele aplicației pentru a produce un blocaj în funcționarea sistemului. În acest caz performanțele software sunt forțate să atingă un prag maxim pentru a satisface numărul mare de cereri primite. Pentru a reprezenta o aplicație software este propusă o arhitectură bazată pe două nivele de funcționare, folosind modele ce fac uz de teoria cozilor pentru abstractizarea cererilor.

Modelul care este utilizat este reprezentat mai jos, iar metoda de testare este automată, bazându-se pe o buclă de feedback care ajută la generarea de trafic pe baza ieșirii modelului. Simultan sistemul testat este monitorizat prin analiza anumitor metrici de interes, iar rezultatele obținute vor ajuta la crearea unui plan de testare îmbunătățit ulterior.

Deoarece o aplicație poate deține mai multe tipuri de servicii se preferă o împărțire a acestora în clase de servicii pentru a descrie mai clar interacțiunile dintre sistem și utilizatori care generează sesiuni de conexiune. Această clasificare se realizează din punctul de vedere al comportării statistice la un volum mare de cereri și al cerințelor de funcționare. Nivelul software al aplicației (nivelul superior al diagramei de mai jos) conține resursele software, precum: serverele și baza de date, în timp ce nivelul hardware (nivelul inferior) conține microprocesoarele pe care rulează serverele și celelalte resurse fizice necesare. Acest nivel hardware poate fi interpretat drept o colecție de cozi, câte una pentru fiecare clasă de serviciu.

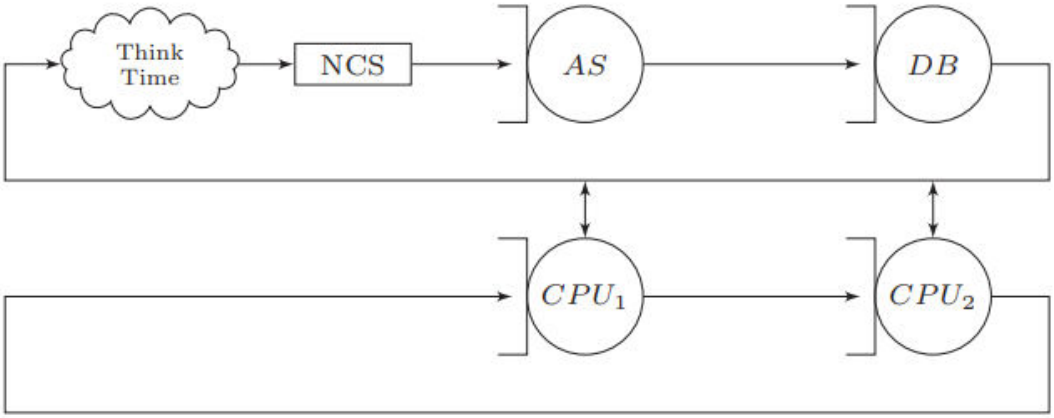

Figura 4. Reprezentarea arhitecturii “two tier” pentru o aplicație web [7]

Pe layer-ul superior există două cozi care se ocupă cu cereri atunci când nu mai există fire de execuție disponibile. Totodată, pe acest nivel este implementată o zonă non-critică ilustrată de blocul NCS care nu prezintă întârzieri la introducerea cererilor în cozile corespunzătoare claselor de servere,

deoarece nu constituie o resursă partajată. Blocul de "Think Time" modelează logica de generare a cererilor cu intervale de timp nenule între acestea, adică într-o ordine bine determinată asemănătoare procesului gândirii umane.

Funcționarea unui astfel de framework de Stress Testing este sugerată prin schema bloc prezentată mai jos. Acesta va încerca să genereze un trafic de cereri care să conducă către identificarea valorilor critice pentru anumite metrici de performanță pentru aplicații. Astfel, un controller de testare automată este componenta responsabilă de rularea unor algoritmi ce pot avea obiective diferite. De exemplu, există algoritmi implementați pentru calcularea numărului de utilizatori necesari pentru a determina sistemul să funcționeze la parametri maximi sau pentru estimarea capacității pentru fiecare clasă de resurse din sistem. În timpul testării sunt monitorizate evoluția gradului de utilizare CPU, utilizarea discului, timpul de așteptare sau debitul (la câte cereri poate sistemul să returneze un răspuns în unitatea de timp). Ulterior, datele colectate în faza de monitorizare sunt filtrate pentru corecția erorilor cu ajutorul unui estimator Kalman și apoi pe baza acestora se vor aprecia valorile de prag pentru componentele hardware testate.

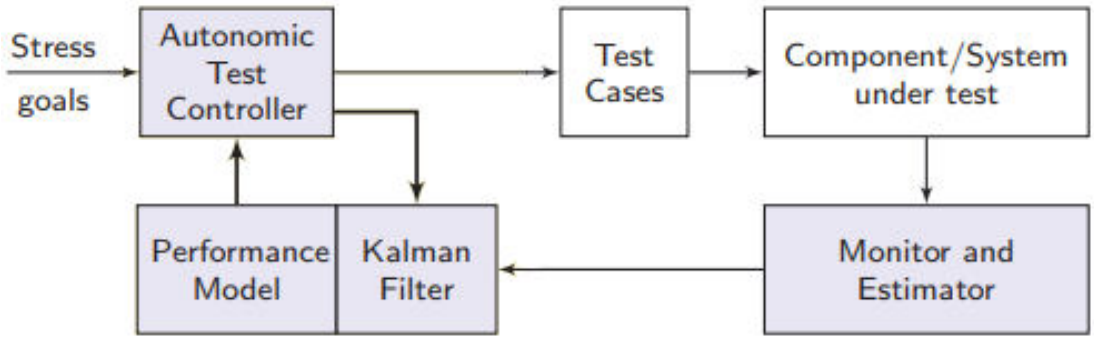

Figura 5. Arhitectura unui sistem automat de Stress Testing [7]

C. Barna, M. Litoiu și H. Ghanbari, autorii studiului cu referința numărul șapte, propun testarea acestui framework pe o aplicație implementată având următoarele resurse: trei mașini de calcul cu sistemul de operare Windows XP (deci sunt trei utilizatori ce vor genera cereri), un server de baze de date MySQL și două servere web de tip Tomcat. De asemenea aplicația prezintă și un Load Balancer Apache cu rolul distribuirii cererilor în mod optim către cele două servere Tomcat. Pe fiecare mașină de calcul au fost instalate cu aplicații de monitorizare pentru măsurarea parametrilor de performanță. Aplicația testată este de tipul unui magazin online, permițând trei categorii principale de operații care pot fi înțelese ca și clase de servicii:

- căutare - un utilizator poate cere informații despre articolele prezentate
- cumpărare - este procesul de adăugare a unui articol în coșul de cumpărături
- plătire - utilizatorul finalizează achiziția

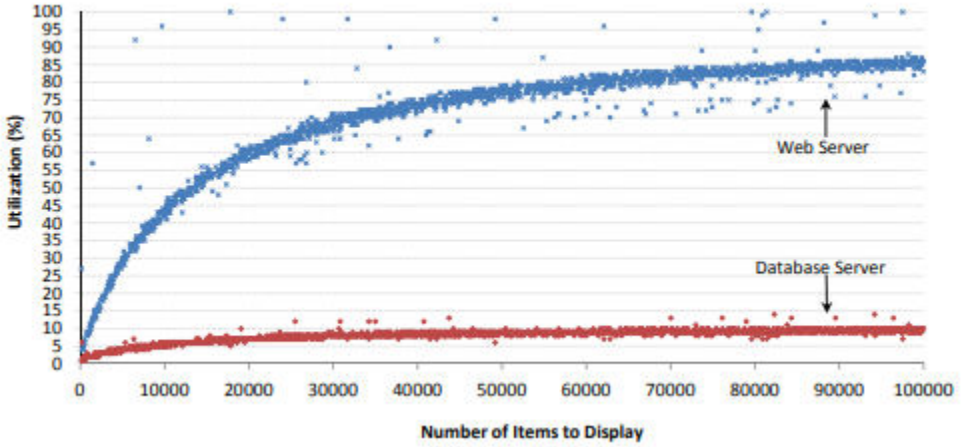

Figura 6. Graficul evoluției gradului de utilizare CPU pentru operația de căutare [7]

Operația de căutare reprezintă trimiterea unei cereri către baza de date la accesarea unei pagini ce prezintă detalii despre un articol sau mai multe. Baza de date furnizează serverului web datele cerute, iar acesta va popula pagina web cu rezultatele cererii SQL. Numărul de articole cerute va reprezenta variabila de interes pentru a cerceta gradul de utilizare CPU al cărui grafic este reprezentat mai sus.

Acest grafic ilustrează gradul de ocupare al microprocesorului pentru serverul web și, respectiv, serverul bazei de date pentru valori de până la o sută de mii de cereri. Se poate observa faptul că gradul de ocupare CPU are o creștere foarte rapidă pentru serverul web, în timp ce serverul MySQL are un grad de ocupare de sub 15% pentru valoarea de o sută de mii de articole afișate. Din cauza evoluției rapide în cazul serverului web este propusă o divizare în scenarii mai mici pe intervale ale gradului de utilizare astfel:

1. căutare0 - între 0 și 2 855 de articole (grad de ocupare de până la 25%)
2. căutare1 - între 2 856 și 11 752 de articole (grad de ocupare între 25% și 50%)
3. căutare2 - între 11 753 și 48 730 de articole (grad de ocupare între 50% și 75%)
4. căutare3 - între 48 731 și 100 000 de articole (grad de ocupare de peste 75%)

Acestor patru micro-scenarii li se adaugă clasele de servicii de cumpărare și plătire, iar ca urmare a rulării algoritmului de Stress Testing pentru această configurație de scenarii s-au obținut valori ale timpului CPU (timpul mediu de procesare al unei operații din cele șase specificate mai sus) ce sunt prezentate în tabelul de mai jos.

Tabelul 2. Rezultatele analizei de Stress Testing ale frameworkului bazat pe arhitectura "two-tier"

Operația (clasa de servicii)	Timpul CPU pentru serverul web (ms)	Timpul CPU pentru serverul bazei de date (ms)
cumpărare	5,38	5,6

plătire	5,17	5,6
căutare0	41,32	11,78
căutare1	192,84	39,91
căutare2	769,29	153,2
căutare3	1 961,82	372,3

După cum se poate observa din acest tabel evoluția celui de-al doilea parametru investigat are o creștere asemănătoare cu cea a primului parametru pentru ambele tipuri de servere. Din aceste analize nu se pot trage anumite concluzii, deoarece serverele web și cele de baze de date sunt foarte diferite. În primul rând, limbajele cu care sunt implementate aceste servere nu prezintă aspecte comune. De exemplu, pentru aplicațiile web se folosesc tehnologii precum Java, PHP, ASP .NET, JSP, în timp ce bazele de date folosesc un limbaj propriu de interogare care nu sunt utilizate în cadrul altor tipuri de aplicații sau servere (SQL / NoSQL). În funcție de volumul de informații stocat în serverele bazei de date și de gradul de ordonare al lor, performanțele serverului pot fi inferioare celor ale serverelor web, deoarece un container foarte mare de date poate necesita un timp mai mare pentru returnarea rezultatului unei cereri și un grad mai mare de utilizare al resurselor dacă se apelează la procedee de căutări mai rapide (de exemplu, multithreading). [8]

Un alt aspect foarte important care influențează rezultatele testării îl reprezintă infrastructura hardware care a fost implementată pentru platformă. Astfel, pentru a se asigura disponibilitatea resurselor care sunt puse la dispoziție de platforma web, sunt proiectate arhitecturi de tipul Load Balancer. Această arhitectură conține un echipament numit Load Balancer care are rolul de a distribui cererile venite din Internet pentru a evita suprasolicitarea serverelor care oferă serviciile. De asemenea, avantajul unei astfel de implementări o constituie rezistența în fața atacurilor DoS și DDoS. Pentru a se evita creșterea traficului prin Load Balancer se preferă conectarea serviciilor în Internet, comunicarea cu mașinile ce au făcut cererile inițiale facandu-se prin intermediul unor tunele cu ajutorul unor seturi de protocoale prevăzute în standardul IPsec (Internet Protocol Security).

Pentru filtrarea traficului din Internet se utilizează unul sau mai multe firewall-uri care protejează serverele aplicației de trafic cu potențial malițios. Firewall-urile sunt în majoritatea cazurilor implementate hardware pentru a nu fi afectate de alte aplicații și bug-uri ale acestora care pot provoca breșe de securitate în cazul în care acestea ar fi implementate în varianta software. Cererile primite de la aplicația de Stress Testing pot fi filtrate de aceste dispozitive în funcție de modul în care au fost configurate pentru rețeaua protejată și din această cauză să se obțină rezultate neconforme cu realitatea.

Un alt caz important de menționat îl reprezintă utilizarea de proxy-uri. Acestea reprezintă un intermediar în comunicarea client-server. Există multe tipuri de proxy-uri, important de punctat în cadrul subiectului dezvoltat fiind proxy-ul cache. Practic, rolul proxy-ului de tip cache este acela de a stoca informațiile de la servere care sunt cele mai cerute de către utilizatori pentru a împiedica creșterea traficului la servere și oferirea unui timp optim de răspuns la clienți. Astfel, dacă aplicația de testare generează un trafic care în mare măsură este deservit de aceste proxy-uri, atunci rezultatele nu vor fi utile pentru că cererile nu au ajuns de fapt la server.

4 Concluzii

În lucrarea de față a fost punctată importanța tehnicii de Stress Testing în asigurarea calității produselor software. Pentru exemplificarea acestui aspect a fost propusă abordarea unei situații în care s-a făcut uz de un framework de testare automată bazat pe teoria feedback-ului și teoria cozilor. Studiul de caz prezentat avea ca punct de interes testarea a două tipuri de servere: un server web Tomcat și un server de baze de date MySQL. Rezultatele analizei menționate au arătat gradul ridicat de utilizare CPU al serverului web în comparație cu cel al serverului bazei de date. Totodată, serverul web de tip Tomcat acceptă un număr de 100 000 de cereri cu un grad de ocupare CPU de aproximativ 85% ceea ce reprezintă o valoare satisfăcătoare pentru un site de comerț electronic.

Aceste testări sunt foarte importante pentru alegerea componentelor sistemului informatic în funcție de obiective și costurile dorite sau așteptate. Stress Testing-ul este o metodologie foarte utilă, dar nu este singura care trebuie luată în considerare. Există o multitudine de alte tipuri de testare din care se pot extrage detalii despre funcționarea sistemului din mai multe puncte de vedere. De exemplu, este recomandat să se verifice dacă o aplicație este compatibilă cu o versiunea curentă a unui browser pe un anumit sistem de operare, testare numită Browser Compatibility Testing. În plus, din rațiuni de protecție a datelor, se impune verificarea vulnerabilității aplicației în fața programelor malware și virușilor informatici, fiind monitorizate schimbările de comportament în cazul apariției unui incident de securitate.

Pentru activitatea de cercetare pentru următorul semestru, doresc să trec la partea de identificare a cerințelor funcționale și să încep implementarea unei astfel de aplicații de testare a performanțelor pentru diverse platforme web. Momentan ca funcționalități am în vedere următoarele:

- crearea de cereri simulate din partea unor utilizatori sub formă de fire de execuție
- crearea de planuri de testare prin specificarea mai multor informații despre platforma testată (URL-ul către server sau precizarea adresei IP împreună cu portul pe care rulează serverul)
- vizualizarea răspunsurilor venite de la server în diverse formate (JSON, de exemplu) direct în interfață
- vizualizarea istoricului pentru teste

Pentru proiectare intenționez să folosesc framework-ul Angular pentru partea de interfață cu clientul și limbajul Java pe partea de backend, deoarece oferă posibilitatea de multithreading care este ideală pentru simularea de utilizatori virtuali care generează trafic.

5 Bibliografie

[1] Ms. S. Sharmila , Dr. E. Ramadevi (March 2014) - Analysis of Performance Testing on Web Applications, International Journal of Advanced Research in Computer and Communication Engineering Vol. 3, Issue 3 (disponibil la adresa https://ijarcce.com/wp-content/uploads/2012/03/IJARCCE4H-s-sharmila-Analysis-of-Performance-Testing-on-Web-Applications.pdf)

[2] Interfața aplicației Apache JMeter: https://www.researchgate.net/figure/Specifying-Threads-Users-in-Apache-JMeter_fig1_332759795

[3] Interfața aplicației LoadRunner: https://huddle.eurostarsoftwaretesting.com/oracle-ebs-performance-testing-using-loadrunner/

[4] Interfața aplicației Neoload: https://www.aerow.group/tag/neoload-web-on-premise/

[5] Comparație între diverse implementări anterioare: https://www.ubik-ingenierie.com/blog/apache-jmeter-vs-neoload-vs-hp-load-runner/

[6] H. Sarojadevi (2011) - Performance Testing: Methodologies and Tools, Journal of Information Engineering and Applications, Vol 1, No.5 (disponibil la adresa https://core.ac.uk/download/pdf/234676929.pdf)

[7] C. Barna, M. Litoiu, and H. Ghanbari (June 2011) - Autonomic Load-Testing Framework, Department of Computer Science and Engineering York University Toronto, Canada (disponibil la adresa http://barbie.uta.edu/~jxu/Autonomic%20Load-Testing%20Framework.pdf)

[8] Andrew J. Kornecki, Nick Brixius, Ozeas Vieira Santana Filho (2005) - PERFORMANCE ANALYSIS OF WEB SERVERS - Apache and Microsoft IIS (disponibil la adresa https://www.scitepress.org/Link.aspx?doi=10.5220%2f0001231102930298)