Seminár k záverečnej práci

Milan Chrastina

4Ib, 2016-2017

Abstrakt. Nástroj na výkonnostné testovanie PerfCake v súčasnej dobe nie je možné spustiť priamo z integrovanej testovacej sady. Hlavným cieľom našej práce je preskúmať a navrhnúť možné východiská integrácie nástroja PerfCake. Ako nástroj, ktorý nám pomohol dosiahnuť cieľ sme zvolili testovací framework Arquillian. Zoznámili sme sa s hlavnými črtami Arquillianu a navrhli riešenie implementácie s nástrojom PerfCake. V práci sme implementovali do nástroja PerfCake rozšírenie, ktoré pomocou Arquillianu vloží a spustí testovací scenár ako normálny JUnit test. Ďalej sme sa zaoberali analýzou niekoľkých dostupných open-source nástrojov na výkonnostné testovanie. Pri ich skúmaní sme sa zamerali na architektúru, prevádzanie testov, podporu protokolov, definíciu pracovnej záťaže, jednoduchosť používania a monitoring.

Kľúčové slová: výkonnostné nástroje integrácia, výkonnostné testovanie, Perfcake, Arquillian

1 Úvod

V súčasnej dobe, ktorá je charakteristická prudkým vývojom v oblasti nových technológií a prechodom aplikácií na viacvrstvovú architektúru, sa začali nezanedbateľným spôsobom prejavovať výkonnostné problémy novo vzniknutých systémov. Na začiatku stačilo na testovanie úzkych miest manuálne vykonávané testy za pomoci väčšieho počtu užívateľov. No pri permanentnom zvyšovaní týchto užívateľov sa to stávalo finančne a organizačne neúnosné. Jednotlivé firmy si už dnes nemôžu dovoliť zamestnať napríklad tisíc testerov, ktorý budú využívať ich aplikáciu s cieľom nájsť jej limity. Ani za predpokladu, keby firmy najali taký veľký počet ľudí, neexistuje garancia, že by jednotlivé testy boli rovnaké. Tieto fakty dali podnet k vzniku nástrojom na výkonnostné testovanie.

Výkonnostné testy slúžia k získaniu výkonnostných charakteristík meraného systému pri rôznych typoch záťaže alebo odlišnej konfigurácii systému. Hlavným prínosom výkonnostných testov je nadobúdanie informácií o danom systéme a simulácia v reálnom nasadení, čo je samozrejme spojené s lepším plánovaním nákladov na rozširovanie daného systému.

Pri meraní výkonu sa musíme zamerať na kľúčové ukazovatele výkonnosti. Tie môžeme rozdeliť na dva základné typy: orientované na službu a orientované na efektívnosť.

Indikátory orientované na službu sú dostupnosť a časová odozva, ktoré odrážajú ako správne aplikácia poskytuje službu koncovému používateľovi. Dostupnosť

chápeme ako množstvo času počas ktorého je aplikácia dostupná pre používateľa. Z pohľadu na výkonnosť to vnímame ako kompletnú neschopnosť aplikácie, ktorú koncový používateľ nie je schopný efektívne využívať. To sa môže stať buď z dôvodu skoro žiadnej odozvy systému alebo je táto odozva degradovaná na neakceptovateľnú hranicu. Časová odozva je množstvo času poskytnutý aplikácií na odpovedanie požiadavky používateľovi. Vo výkonnostnom testovaní je to normálne meranie systémovej odozvy. Systémová odozva je čas medzi odoslaním požiadavky používateľovi a prijatím odpovede aplikácie na stanicu z ktorej bola odoslaná.

Indikátory orientované na efektívnosť sú priepustnosť a využitie, ktoré odrážajú ako dobre aplikácia využíva infraštruktúru hostingu. Priepustnosť je tempo, ktorým aplikácia vybavuje jednotlivé eventy. Využitie chápeme ako percento teoretickej kapacity zdrojov, ktoré sú používané. Ako napríklad využitie šírky pásme siete alebo využitie pamäte web servera pri 1000 aktívnych používateľoch.[2]

2 Princíp výkonnostných testov

Hlavným princípom automatizovaných výkonnostných testov je simulácia vybraných business transakcií pomocou záťažových skriptov. Každý skript by mal byť upravený tak, aby bol schopný postupne spracovávať vopred pripravené dáta. Taktiež musí v danom čase vykonať požadovaný počet transakcií s príslušným počtom virtuálnych užívateľov. Pre nahratie skriptov a realizáciu záťažových testov je nevyhnutné, aby bol daný systém bez chýb, teda funkčne otestovaný. Nástroj obvykle zaznamenáva komunikáciu medzi klientskou stanicou a serverom na úrovni príslušného komunikačného protokolu. Odozvy jednotlivých virtuálnych užívateľských krokov sú obyčajne merané na úrovni sieťového rozhrania.

2.1 Metódy testovania

Určité metódy testovania boli stanovené ako štandardy v oblasti výkonnostného testovania. Niekoľko typov výkonnostných testov sa využíva v určitých fázach vývoja na rôzne účely.

Stress test.

Je technicky kontrolovaný DoS útok smerovaný na testovaný systém, resp. DDoS útok za predpokladu, že používame viac ovládacích uzlov. Vo výkonnostnom hodnotení je to spôsob, ako nájsť horné limity testovaného systému. Dosahujeme to generovaním čo najväčšej záťaže, voči testovanému systému, až kým sa jedna z jeho častí „nezlomí“. Ako napríklad prvá odpoveď HTTP so statusom 500 namiesto 200, systém vyčerpá pamäť, diskový priestor alebo iný zdroj a preto už nemôže slúžiť svojim používateľom. Taktiež, môže úroveň služby presiahnuť hranicu stanovenú v požiadavkách na výkon - doba odozvy je vyššia ako 30 sekúnd. Stresové testy majú zvyčajne formu testu s určitým reprezentatívnym pracovným zaťažením s konštantným rozbehom profilu vkladania. Limity výkonu sú dôležitým faktorom, ktorý je potrebné mať na pamäti pri navrhovaní systému a preto by mohlo byť prospešné vykonať stresové

testovanie ešte pred nasadením výroby, aby sme sa ubezpečili, že systém spĺňa požiadavky na výkonnosť a má priestor pre budúci rast.

Load test.

Toto je pravdepodobne jeden z najbežnejších typov testu výkonnosti, ktorý sa niekedy označuje aj ako test spoľahlivosti. Zmyslom záťažového testu je preukázať, že systém môže udržať plnú prevádzku v rámci limitov dohodnutých v SLA pod vrcholom záťaže definovanej v požiadavkách. Vrchol záťažového testu je niekedy definovaný, ako určité percento bodu zlomu stanoveného počiatočným stres testom. Záťažový test sa zvyčajne realizuje s konštantným virtuálnym počtom užívateľov s voliteľnou záťažou, čo umožňuje zahriatie systému a zároveň aj sledovanie jeho reakcie na zvyšujúcu sa záťaž.

Soak test.

Tento test je technicky „load test“, niekedy používaný pod pojmom test stability. Tento typ testu overuje stabilitu systému tým, že ho podrobí dlhému testovaniu konštantnou rýchlosťou. Cieľom je odhaliť potenciálne problémy, ktoré sa ťažko objavia inými prostriedkami, ako napríklad úniky pamäte alebo aritmetické pretečenie. Takéto problémy je možné identifikovať ako pomalé a postupné vyčerpávanie systémových zdrojov (dostupná pamäť) alebo postupné zvyšovanie sledovaných parametrov, ako napríklad doba odozvy. Ďalším typom, pri ktorom je možnosť, že sa problém objaví v závislosti na dĺžke trvania testu, je samotná súbežnosť vlákien.

3 Zlá výkonnosť alebo prečo používať výkonnostné testy

Pokúsime sa uviesť niektoré príklady dobrej a zlej výkonnosti, resp. prečo veľa aplikácií skrachuje pri dosiahnutí tohto vymedzeného cieľa.

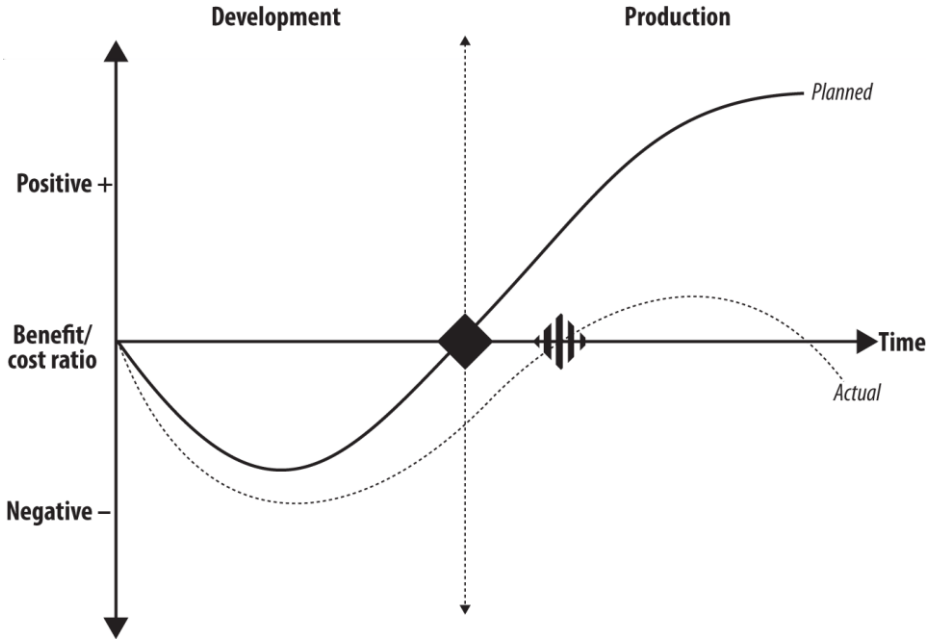
3.1 IT krivka z ekonomickej hodnoty

Výkonnostné problémy sa na nešťastie zvyknú objavovať až neskôr v životnom cykle aplikácií. Dodatočne, keď ich objavíme, stoja nemalú cenu a snahu o vyriešenie. Krivka, ktorá ilustruje tento prípad je zobrazená na Obr. 1.

Súvislá krivka (planned) na obrázku znázorňuje výsledok, kde proces vývoja aplikácie je správne navrhnutý. Príjmy projektu vznikajú v momente, kedy boli naplánované (čierny kosoštvorec). Teda aplikácia je nasadená v presne stanovenom termíne a hneď po nasadení sa z nej profituje. Vznikajú tu malé alebo žiadne výkonnostne komplikácie po nasadení.

Prerušovaná krivka (actual) nám demonštruje oveľa častejšiu realitu, kde sa nasadenie vo vývoji aplikácie natiahne (prerušovaný kosoštvorec). Výrazne neskoršie nasadenie nám zvyšuje náklady a aj čas, čo je zapríčinené opravovaním výkonnostných chýb

v produkcii. To v praxi znamená, že aplikácia, ktorá má profitovať opakovane nefunguje podľa predpokladov, čo je dosť nepriaznivá situácia pre podnikanie.



Obrázok. 1. IT krivka z ekonomickej hodnoty

3.2 Prístupy k výkonnostnému testovaniu

Obrázok 2 je postavený na zozbieraných dátach od Forrester Research zameraných na výkonnostné defekty, ktoré museli byt fixované v produkcii ako typický aplikačný vývoj.[1]

Resolving Performance Defects (2006)

Approach	*% Resolved in Production*
Firefighting	100%
Performance Validation	30%
Performance Driven	5%

Source: Forrester Research

Obrázok. 2. IT krivka z ekonomickej hodnoty

[1] Dáta zozbierané inštitútom v roku 2006 sa do teraz veľmi nezmenili [2]

Na obrázku 2 môžeme vidieť 3 úrovne prístupov k výkonnostnému testovaniu. Prvý, hasenie ohňa (firefighting), predstavuje prístup, keď žiadne alebo veľmi malé výkonnostné testovanie je zahrnuté v bežnom aplikačnom vývoji. Respektíve, všetky výkonnostné defekty musia byť ošetrované v už bežiacom prostredí. Spomínaný prístup je v súčasnej dobe síce relatívne bežný, ale zároveň najmenej preferovaný v danej oblasti. Firmy, ktoré sa nachádzajú v tomto móde, sa samy vystavujú vážnemu riziku.

Druhá úroveň (Validácia výkonnosti v procese) pokrýva firmy, ktoré majú v sebe zahrnutý proces výkonnostného testovania, avšak nie až do úplného konca životného cyklu aplikácie. Aj v tomto prípade sa vyskytuje pomerne veľký počet výkonnostných defektov po nasadení do produkcie(30%). V tejto oblasti sa nachádza najviac organizácií.

Tretia úroveň (Riadená výkonnosť) je miesto, na ktoré by sa mali organizácie zameriavať v modeli výkonnostného testovania. Predstavuje prístup, kde sa na každej úrovni životného cyklu aplikácie berie do úvahy koncept výkonnostného testovania. Výsledok tohto modelu predstavuje výskyt len malého počtu výkonnostných defektov.

3.3 Nedostatok úvah o výkonnosti v aplikačnom návrhu

Pokiaľ nezavedieme úvahy o výkonnosti do aplikačného návrhu, vystavujeme sa možným problémom. Aplikácie „zamerané na výkon“ majú dobrú výkonnosť alebo aspoň schopnosti k zmene pri potrebe vysporiadať sa s nepredvídateľnými výkonnostnými výzvami. Návrhové výkonnostné problémy, ktoré sa nepredpokladajú počas vývoja a preniknú do životného cyklu aplikácie, je veľmi ťažké prekonať. Niekedy je dokonca nutné úplné prebudovanie celej aplikácie.

Mnoho aplikácií je postavených na softvérových komponentoch, ktoré môžu byť testované osobitne a fungovať správne, avšak nie v aplikácii ako celku. Tieto komponenty musia vzájomne efektívne spolupracovať k dosiahnutiu dobrej výkonnosti.

3.4 Testovanie výkonnosti je ponechané na poslednú chvíľu

Ako už bolo spomenuté, veľa spoločností operuje vo výkonnosti na úrovni validácie výkonu. Na tejto úrovni je vykonávané testovanie len pred nasadením aplikácie, s malým ohľadom na požadovaný čas alebo následok zlyhania. Hoci je to lepšie ako „hasenie ohňa“, tento režim aj tak nesie značný risk, ktorý znemožňuje identifikovať vážny výkonnostný defekt. Ten sa objaví len v produkcii, čo neposkytuje dostatok času na správne vyriešenie problému, ktorý nebol identifikovaný pred produkciou.

Typickým výsledkom pre tento režim je oneskorenie zavedenia aplikácie, pokiaľ problém nie je ošetrený. Aplikácia, ktorá je vydaná s podstatnou výkonnostnou chybou bude vyžadovať nákladné, časovo náročné prepracovanie. V najhoršom prípade môže byť aplikácia stiahnutá z obehu, pokiaľ sa neprerobí správne.

Všetky tieto výsledky majú extrémne negatívny efekt na podnikanie a dôveru k používaniu aplikácie a jej vývojárom. Preto potrebujeme testovať výkonnostné problémy tak skoro, ako je len možné a nenechávať to na poslednú chvíľu.

4 Nástroje na výkonnostné testovanie

Nástroje na výkonnostné testovanie sú v určitej forme súčasťou praxe už minimálne 20 rokov. Za túto nemalú dobu prešli mnohými technologickými zmenami. Postupne sa menili z veľkej klientovej aplikácie na webovú a predovšetkým na mobilné sprístupnenie. Na základe všetkých týchto prispôsobení sa automatické nástroje zameriavajú na poskytovanie webového a mobilného vývoja a oveľa menej podporujú staršie technológie postavené na dvojvrstvovom aplikačnom modele. Táto zmena zamerania má pozitívny vplyv na nárast počtu automatizovaných nástrojov na trhu za primeraný rozpočet a taktiež dosť populárnych open source riešení. Všetko spomínané má dobré praktické využitie, avšak pri výkonnostnom testovaní sa niekedy potrebujeme dostať mimo web a náš výber nástrojov rapídne klesá. Technologické výzvy, ktoré pohltili množstvá automatizovaných nástrojov, sú toho len dôkazom. Taktiež sa môžu využiť open source riešenia, s čím sa môžu spájať určité problémy. Problém nie je výkon a analýza nástrojov, ale skôr schopnosť správne nahrať aktivitu aplikácie a jej následnú modifikáciu pre výsledný skript použitý vo výkonnostnom teste. Pre nástroje výkonnostného testovania, predstavujú problém aj pojmy ako šifrovanie a kompresia, ktoré pravdepodobne budú brániť nahratiu skriptu a zriedka sa dajú vypnúť. Podobne predstavujú problém aj niektoré webové aplikácie (streamovanie videa, bezpečnostné certifikáty), pričom na trhu dostupné nástroje neponúkajú všetky potrebné riešenia. Predstavíme aplikácie – Faban, Gatling, Grinder, JMeter, PerfCake. Zamerali sme sa na JVM aplikácie, ktoré podporujú súčasné vývojárske procesy novodobých webových platforiem. Opíšeme vlastností, možnosti a charakteristiky jednotlivých nástrojov, keďže každý z nich ponúka niečo iné a ich funkcie sa len čiastočne prekrývajú.

4.1 Faban

Základný modul je Faban Driver Framework – API pre kontrolu životného cyklu testu a definície pracovnej záťaže, vrátane nastavení pre náhodné modelovanie správania používateľa. Podpora pre spustenie hromadných ovládacích agentov (driver) je zabudovaná, no nie je vyžadovaná. V prípade hromadných operácií musí byť spustený RMI register manuálne na hlavnom uzle (Master). Potom na každom ovládacom uzle je spustených jeden alebo viac procesov daného agenta, v ktorých beží niekoľko vlákien, kde každé hosťuje jeden driver. Je tu zachovaný tradičný model konkurentnosti, teda jeden používateľ na jedno vlákno. Tak je možné dôkladne kontrolovať vlákna v rozsahu, ktorý je deklarovaný v testovacej konfigurácií a každý driver má vlastnú konfiguráciu vlákien v danom rozsahu. Taktiež je nakonfigurovaný počet agentov na každý uzol. Presne ako webstránka varovala, táto komplexná architektúra vytvára ťažšie podmienky pre prvé spustenie vlastného testu, no jej hodnota je v možnostiach, ktoré ponúka.

4.2 Gatling

Gatling je jednoduchý nástroj pre stresové testovanie založený na Scale. Vyniká vďaka konkurentnému modelu, ktorý nie je používateľ-na-vlákno, ako je vidieť vo väčšine ostatných nástrojov. Gatling je založený na Akka a Netty a využíva asynchrónny model, ktorý závisí od neblokujúcich implementácií transportov. Počet vlákien potrebných na simuláciu vysokého počtu virtuálnych používateľov je výrazne nižší ako v modeloch používateľov na jedno vlákno. Nižší počet použitých vlákien tiež znamená nižšie rýchlosti prepínania kontextu, ktoré zbytočne spotrebúvajú zdroje procesora, keď príliš veľa vlákien súperí o procesor. Gatling v súčasnosti nemá žiadny režim clusteru a beží iba v jednom uzle

4.3 Grinder

Grinder je nástroj založený na JVM, ktorý primárne podporuje skriptovanie v Jythone a nedávno aj Clojure. Klastrová operácia je štandardne podporovaná a agenti dokonca môžu vykonať test bez hlavnej konzoly. Procesy s viacerými agentmi sa môžu spustiť na každom stroji a každý proces agenta vykoná pracovné vlákna testov. Terminológia v Grinder sa používa mierne atypicky - prvok najvyššej úrovne je skript, ktorý inštancuje testovacie objekty, ktoré zaznamenávajú metriky.

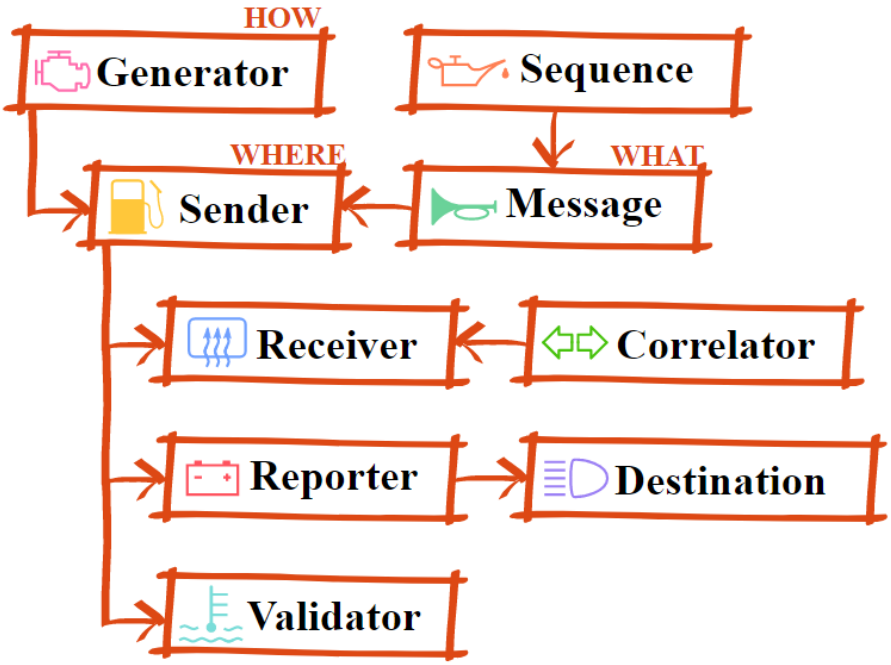
4.4 JMeter

JMeter bol pravdepodobne najdlhší čas medzi recenzovanými nástrojmi a je to zjavné v mimoriadne širokej sade funkcií. Podporuje clusterové testovanie bežným spôsobom, s trochu neobvyklou terminológiou - proces, ktorý sa spúšťa na ovládači, sa zvyčajne nazýva agent, sa nazýva, trochu zavádzajúcim spôsobom, serverom JMeter. Doplnková časť, ktorá sa nachádza na koordinačnom uzle, sa podľa očakávania nazýva klientom. Hlavnými stavebnými prvkami modelu JMeter sú: top-level ThreadGroup, ktorá obsahuje všetky ostatné prvky, samplery, ktoré generujú záťaž a listener-ov, ktorí zhromažďujú údaje o meraní.

4.5 PerfCake

Jedným z nástrojov na výkonnostné testovanie je framework PerfCake od firmy RedHat, ktorý je náš hlavný objekt, ktorému sa v tejto práci budeme venovať. PerfCake je nenáročný nástroj na výkonnostné testovanie a záťažový generátor so zameraním na minimalistickosť s jednoduchým použitím, stabilnými výsledkami, zaťažujúcimi čo najmenej meraný systém. Je nezávislý na platforme s využitím komponent a vysokou priepustnosťou. Ako sme načrtli je zložený z komponent, teda v ňom môžeme vytvárať vlastné testy pozostávajúce zo základných blokov, čo prináša vysokú flexibilitu v konfigurácií a v možnosti jednotlivo vytvorené komponenty znova využiť na odlišnú úlohu. Vďaka svojej architektúre podporuje rôzne protokoly a rozhrania ako napríklad HTTP, REST, SOAP, JDBC atď. s tým, že ich počet sa stále rozširuje a implementujú sa nové

rozhrania. Je tu jednoduchý koncept architektúry, kde sa ako jediná komunikačná jednotka vyskytuje Správa. Tieto Správy PerfCake posiela na cieľový systém a meria čas odozvy.



Obrázok. 3. Architektúra nástroja PerfCake

Na Obr. 3 môžeme vidieť schematické znázornenie architektúry. Máme tu práve jeden generátor, ktorého hlavný účel je špecifikovať, ako majú byť správy generované, koľko správ bude generovaných a ako rýchlo budú paralelne odosielané na cieľ. Štandardný generátor je implementovaný ako rad správ pripravených na odoslanie Senderom. Sender špecifikuje protokol, ktorý je použitý na odoslanie správy. To môže byť napríklad HTTP, REST, SOAP, JDBC, web soket, súbor a iné. Sender je konfigurovaný so špecifickou adresou pre daný protokol, teda hovorí kam budú správy odosielane. Čo sa posiela je obsahom Správy (Message), čo je najmenšia jednotka ktorá je odosielaná. Celý priebeh odosielanie je starostlivo monitorovaný reportovacím zariadením (Reporter), ktorý povoľuje a zabezpečuje výpis výsledkov. Na konci ostáva validátor (Validator) ktorý dokazuje, že odpoveď je správna (niektoré systémy pod vysokou záťažou môžu rýchlo odpovedať avšak len varovnými správami o preťažení).

Takže ako sme spomínali podporuje rôzne druhy generovania záťaže, kde môžeme na cieľový systém poslať niekoľko správ a sledovať maximálnu možnú priepustnosť. Výsledok testovania je možný dostať v rôznych formátoch kde sledujeme rôzne výsledky ako napríklad pamäťové nároky na Java Virtual Machine s lineárnou regresnou analýzou, priemernej priepustnosti a iné.[3]

Všetko čo používateľ potrebuje na spustenie svojho výkonnostného testu je vytvoriť tak zvané „Scenár“, ktoré predstavuje XML súbor, ktorý popisuje jednotlivé bloky ktoré by mali spolupracovať. Následne stačí spustiť scenár pomocou Maven projektu alebo predpripraveného shell skriptu.

Keďže z veľkej časti sa výkonnostné testy používajú aj pri vývoji aplikácii, možnosť spustiť PerfCake len za pomoci Maven alebo skriptu nie vždy všetkým vyhovuje. Tu narážame na hlavný bod tejto práce a to absenciu spustenia scenára a vykonanie integrovaného testu priamo z IDE. Našou úlohou je navrhnúť možné riešenie spustenia integrovaného testu z vytvoreného scenára priamo z IDE..

4.6 Porovnanie

Nasledujúca tabuľka sumarizuje subjektívne hodnotenie. Každý nástroj má v každej kategórii priradené body od 0, čo znamená, že podpora pre tento aspekt neexistuje alebo je nefunguje podľa očakávania, 4 znamená, že nástroj spĺňa resp. prekonáva očakávania.

Tab. 1 Hodnotenie výkonnostných nástrojov

	Faban	Gatling	Grinder	JMeter	PerfCake
Architektúra	●●●	●●●	●●●	●●●●	●●
Protokoly	●	●●	●	●●●●	●●●
Definícia pracovnej záťaže	●●●●	●●●●	●	●●●●	●●●●
Injection profiles	●●●	●●●●		●●	●●●
Používateľnosť	●	●●●	●●●	●●●●	●●●●
Monitoring a reporting	●●●	●●●	●●●	●●●	●●●

5 Návrh riešenia

Ako za najschodnejšie riešenie, ktoré napomôže integrácii nástroja PerfCake prichádza do úvahy framework Arquillian. Arquillian je testovacia platforma pre Javu a Java Virtual Machine ktorý umožňuje vývojárom jednoducho písať a vykonávať integrované a funkcionálne testy pre Javu middleware, od Javy EE a ďalej. Nahrádza to,

na čo už unit testy nedosahujú a zameriava sa na integráciu aplikačného kódu priamo v reálnom prostredí behu programu. Arquilian dodržiava tri základne princípy:

1. Testy by mali byť prenosné na akýkoľvek podporovaný kontajner.
   Držaním API špecifikácie kontajneru od testov vývojárom umožňuje prekontrolovávať spustené testy voči rôznym kontajnerom. Teda nenáročné kontajnery môžu byť v kľude použité a neskôr nahradené plnými počas vývoja.
2. Testy by mali byť spustiteľné z IDE ako aj z buildovacieho nástroja
   Vďaka IDE môže vývojár preskočiť build-ovanie pre rýchlejšie vybavenie úlohy a následné ladenie v známom prostredí. Tieto výhody by nemali zrušiť schopnosť spustiť testy v kontinuálnej integrácii pomocou build-ovacieho nástroja.
3. Platforma by mala rozširovať alebo integrovať existujúci test framework
   Rozšíriteľná architektúra podporuje opätovné využitie existujúceho softvéru a podporuje jednotný Java testovací ekosystém.

Arquillian prináša naše testy do behu programu teda nemusíme spravovať beh programu z testu. Všetky potrebné balíčky arquillian-u sa úhľadne pribalia k platforme ktorú rozširujú čo vyúsťuje k jednoduchému spusteniu testov cez IDE.

Ako sme teda spomenuli Arquillian sa výborne hodí na vyriešenie nášho problému, ktorý máme definovaný v cieľoch práce a to schopnosť spustiť a riadiť testy separátne od štandardných testov. Čo ale musíme implementovať je rozšírenie, ktoré by nám umožňovalo spúšťať testy a kontrolór, ktorý by spravoval naše testy v Arquilliane.

Naša predstava implementácie rozšírenia bola, že v testovacej triede ktorá bude spúšťaná s Arquillianom by bola nejaká premenná s anotáciou napríklad::

```
@PerfCake(scenario = "scenar.xml")
Private static PerfCakeController perfCake;
```

Vďaka tomu sme mohli vytvoriť rozšírenie ktoré nám nainjektovalo scenár, PerfCakeController do našej testovacej triedy ktorej použitie vyzerá:

```
@RunWith(Arquillian.class)
@RunAsClient
public class PerfControlTest {

    @PerfControl(scenario = "perfcake.xml")
    private static PerfCakeController perfCake;

    @Test
    public void assertThatPerfCakeControllerExistInTest()
    }
}
```

6 Záver

V práci sme sa venovali výkonnostnému testovaniu a open-source nástrojom, ktoré sa v daných podmienkach používajú. Vyhodnotili sme, že výkonnostné testovanie je oblasť v ktorej sa stále treba venovať a netreba podceňovať prístup k výkonnostným testom, keďže sa vždy môžu vyskytnúť problémy, ktoré môžu mať ekonomicky veľké dopady na daný projekt.

Ako potenciálne najlepšie open source nástroje sme si vybrali: Faban, Gatling, Grinder, JMeter a PerfCake. Nástroje sme skúmali z hľadiska architektúry, pracovnej záťaže, možnosti profilov vkladania ako aj z ich používateľského hľadiska. Následne sme ich aj vyhodnotili a navzájom porovnali. V našom hodnotení najlepšie vychádza JMeter, ktorý aj ale má za sebou dlhé roky vývoja a veľkú používateľskú základňu. Hneď za ním sa objavuje naopak najmladší projekt PerfCake so stále sa rozširujúcim počtom užívateľov. Tento nástroj má preto rýchle nasadzovanie nových metód a teda sa rozširuje ďaleko rýchlejšie ako ostatné nástroje.

Ďalej sme sa zoznámili s integrovaným testovaním a uviedli sme najčastejšie prístupy ktorým sa prevádza. Poznatky z toho sme uplatnili v testovacom framework-u Arquillian. Pri ňom sme si uviedli jeho základnú štruktúru a architektúru. Ďalej sme sa snažili čo najlepšie uviesť princíp jeho fungovania a jeho základov. Pomocou neho sme navrhli riešenie problému ktorý pretrvával vo výkonnostnom nástroji PerfCake (nemožnosť spúšťať testy mimo testovacej sady). A následne sme ho implementovali, pričom sme sa snažili zachovať minimalictiskosť a jednoduchosť používania nástroja PerfCake.

Poďakovanie. RNDr. Peter Gurský, PhD., Mgr. Martin Večera

Skratky

HTTP - Hypertext transfer protocol
REST – Representational State Transfer
SOAP – Simple Acces Object Protocol
JDBC – Java Database Connectivity
IDE - Integrated Development Environment
CDI - Contexts and Dependency Injection
XML - eXtensible Markup Language

Literatúra

1. Steven Haines. 2006. Pro Java EE 5 Performance Management and Optimization. 2. vyd. Apress 2006. 424 s. ISBN13: 978-1-59059-610-4.
2. Ian Molyneaux. 2009. The Art of Application Performance Testing. 2 vyd. O'Reilly Media. 2009. 158 s. ISBN: 978-0-596-52066-3.
3. John D. Ament: Arquillian Testing Guide. 1 vyd. Packt Publishing Ltd. 2013. 242 s. ISBN: 978-1-78216-070-0.
4. Mauro Andreolini, Valeria Cardellini, a Michele Colajanni. 2002. Benchmarking models and tools for distributed web-server systems.V Performance Evaluation of Complex Systems: Techniques and Tools, vol. 2459 z Lecture Notes v Computer Science, 2002, s 208–235.
5. Ron Šmeral. 2014. Modern Performance Tools Applied: doktorandská práca. Brno: Masarykova Univerzita, fakulta informatiky. 2014. 63 s.
6. Projekt PrefCake, https://www.perfcake.org
7. https://github.com/PerfCake/PerfCake
8. Projekt Arquillian http://arquillian.org
9. https://github.com/arquillian/arquillian-extension-drone
10. https://github.com/arquillian/arquillian-showcase/tree/master/extensions/arquillian-static-container-controller