Iedereen die zich op het internet begeeft – en wie doet dat niet tegenwoordig – heeft wel eens last van trage websites, pagina's die niet laden of zelfs onbereikbaar zijn. Jeroen Borgers presenteert zeven stappen om die performanceproblemen te lijf te gaan.

Web performance in zeven stappen

Tips om snelheid in webdiensten te houden

Steeds meer Internetgebruikers kopen in webwinkels. Nu en dan kom ik in een webwinkel waar ik moet wachten op pagina's. Meestal ben ik er dan snel weg: met één klik ga ik naar de concurrent. De toegenomen vergelijkings- en keuzemogelijkheden die Internet biedt, gelden niet alleen voor de producten, maar ook voor de webwinkels zelf. Het is daarmee cruciaal voor het succes van een webwinkel om een snelle website te hebben.
Met weinig bezoekers is een webwinkel meestal wel snel te krijgen. Echter, met de groeiende trend van Internetverkopen, toenemende integratie en complexiteit van achterliggende systemen en de vanuit marketing geëiste almaar toenemende rijkheid aan gebruikerservaring, wordt dit nogal eens een grote uitdaging voor IT-afdelingen. Het gevolg kan zijn dat systemen uitvallen bij grote drukte of een te trage responsetijd hebben. Hoe kunnen we dit voorkomen?
Op basis van praktijkervaring zijn wij tot een aanpak gekomen die is te omschrijven als *meten is weten in de ontwikkelcyclus en de keten*. Ze bestaat uit een zevental activiteiten.

Manifestatie van performanceproblemen

Frustraties en omzetverlies
Wanneer interne applicaties traag zijn, is dit frustrerend voor de gebruikers. Ze kunnen hun werk minder efficiënt doen en raken gedemotiveerd. Wanneer externe applicaties traag zijn, kan dit direct gevolgen hebben voor de omzet van het bedrijf. Als ik een boek wil kopen of een auto- of reisverzekering wil afsluiten, ga ik online vergelijken. Als ik lang moet wachten op een site, dan stuur ik mijn browser met één klik naar de concurrent om daar te kopen.

Verstoring van reguliere ontwikkeling
Dikwijls manifesteren traagheidproblemen zich onverwacht. Alleen aan de niet-functionele aspecten van de software wordt te vaak te weinig en te laat aandacht geschonken. De problemen die onstaan in productie zetten een grote druk op zowel beheerders als ontwikkelaars om de meestal moeilijk te vinden problemen op te lossen. Dit werkt verstorend voor de reguliere ontwikkeling van nieuwe releases: het ontwikkelteam is alleen nog maar bezig brandjes te blussen.

Extra hardware: goedkope oplossing?
Vaak wordt de oplossing gezocht in meer hardware: bijvoorbeeld load balancing over meer webservers of moderner: draaien in een Cloud met een flexibele hoeveelheid hardware. Als de bottleneck echter niet in de web tier zit maar ergens anders, dan zal de investering in meer webserver machines bijvoorbeeld weggegooid geld blijken te zijn. Daarnaast worden bijkomende kosten op de nieuwe hardware voor jaarlijks terugkerende licenties en beheer nogal eens onderschat. Als extra hardware dus al een oplossing is, is het niet altijd een goedkope.

Zeven stappen

Het kan een valide keuze zijn om het risico op performanceproblemen in productie te lopen en de problemen reactief te lijf te gaan. In de regel is het echter verstandiger om pro-actief te zijn en ze daarmee te voorkomen. Deze benadering biedt meer zekerheid, rust en is bovendien ook nog goedkoper. Ze bestaat uit de volgende zeven stappen.

Jeroen Borgers
is senior consultant bij Xebia IT-Architects waar hij de performance practice leidt.

Stap 1: Opstellen performance eisen

Goed specificeren van de performance-eisen is een ondergeschoven kindje. Meestal is de eis: het moet gewoon snel zijn of: minimaal zo snel als het vorige platform. Met zulke vage omschrijvingen begint de verwarring. Het doel is niet duidelijk en wordt typisch door de business- en IT-afdeling heel verschillend geïnterpreteerd. Om dit te voorkomen is het nodig dat de doelen SMART worden geformuleerd en geprioriteerd. Voor een homepagina zal de snelheid belangrijker zijn dan voor een pagina waar de klant zijn profiel kan aanpassen: door prioriteiten te stellen wordt dit expliciet. De R uit SMART van realistisch, haalbaar, wordt meer dan eens vergeten door de opdrachtgever. In dat geval zal een hoge geëiste snelheid geruime ontwikkeltijd of dure hardware gaan kosten. Eén halve seconde trager gedurende piektijden kan best acceptabel zijn als dit tonnen bespaart. Anderzijds kan het terugbrengen van een wachttijd van 4 naar 2 seconden op essentiële plekken aanzienlijke omzetwinst opleveren. Dus een gedegen analyse van de impact van performance op de business value is nodig om de performance-eisen SMART en geprioriteerd te kunnen opstellen en af te kunnen wegen tegen de kosten.

Stap 2: Uitvoeren proof of concept

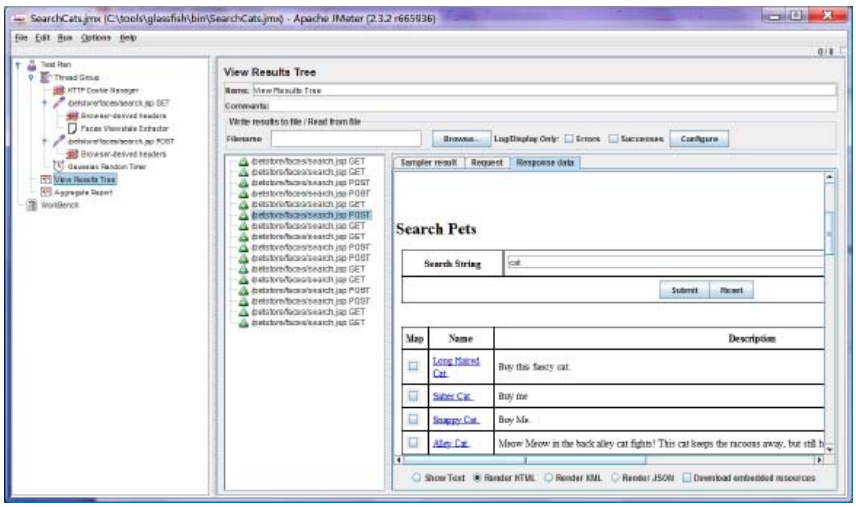
De IT-wereld is erg modegevoelig. Technologie wordt vervangen door iets dat *veel beter* is en waar iedereen gedwee achteraan loopt. Zulke modeverschijnselen zijn bijvoorbeeld CORBA, applets, EJB, Struts, Spring, Server Faces, XML, SOA en RIA. Veelal wordt er daarom nieuwe, bleeding edge, technologie toegepast in een project. Hier komt bij dat elke technologie of framework meestal met zijn eigen kinderziekten komt en meer resources gebruikt dan zijn voorganger. Doel van zo'n nieuwe technologie is doorgaans verbetering in flexibiliteit, productiviteit of onderhoudbaarheid, en performance is ook hier minder zichtbaar en veelal een ondergeschoven kindje.

Het is daarom maar de vraag of met de gekozen nieuwe technologie en architectuur de gestelde performance-eisen gehaald kunnen worden. In de praktijk wordt dit meestal pas in een laat stadium duidelijk: vlak voor de geplande in productiename. Dan kan blijken dat de gebruikte technologie of architectuur niet toereikend is. Soms is het dan nodig om deze rigoreus om te gooien met de hoge kosten die daarbij komen kijken. Het is dus zaak om vroegtijdig een proof of concept voor performance uit te voeren waarbij alle technologie- en architectuuronderdelen geraakt worden, in een vertical slice van de applicatie. Belangrijk is hierbij dat deze test voldoende representatief voor de productie situatie wordt uitgevoerd, hierover zometeen meer. Zodoende kan er op tijd bijgestuurd worden zonder hoge kosten.

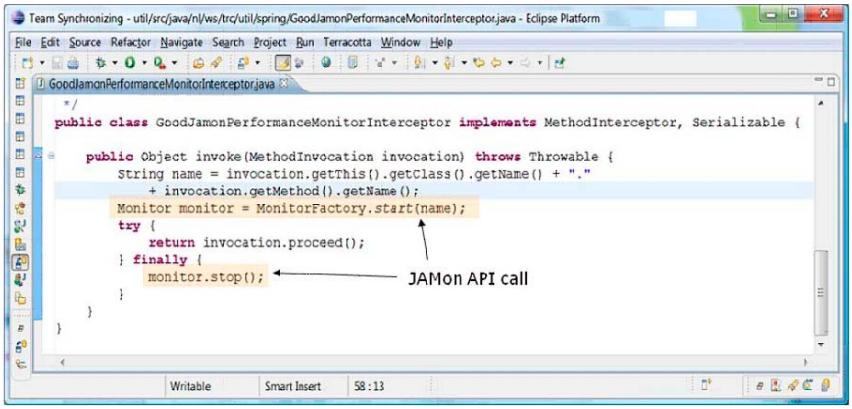
Stap 3: Representatief testen

Vaak wordt traagheid in een ontwikkelomgeving genegeerd met als argument dat de snellere hardware van de productieomgeving dit probleem wel zal oplossen. En dat is alleen te voorspellen met een test op een representatieve omgeving en op representatieve wijze. In zo'n omgeving is er meer representatief dan alleen de hardware. Ik heb dikwijls meegemaakt dat een databasequery op de testdatabase met bijvoorbeeld 1000 klanten onder de 10 ms. duurde, terwijl dit op de productie database met 100.000 klanten tientallen seconden bleek te duren. Als het ontwikkelteam dus niet test met een volledig gevulde database kan de in productiename voor verrassingen zorgen. Ook is het belangrijk dat het aantal gelijktijdige gebruikers en het gebruikersgedrag goed gesimuleerd worden in de test. Verder is het van belang dat er rekening wordt gehouden met caching effecten: als de test voortdurend dezelfde klant hetzelfde product laat opvragen, zullen de gegevens bijvoorbeeld de tweede en volgende keren in de database of query cache aanwezig zijn en zal de response aanmerkelijk sneller zijn dan de eerste keer. De test is dan dus niet representatief voor de werkelijke situatie. Een geschikte performance testtool en performance expertise is hierbij onontbeerlijk. De populairste open source performance testtool is Apache JMeter. Dit is een tool gemaakt door programmeurs voor programmeurs. Het werkt met visuele testscripts en configuratie van deze script elementen zoals een HTTP-request. Er is veel mogelijk, als deze visuele elementen tekortschieten, kun je terugvallen op in BeanShell (dynamische scripttaal met Java syntax) elementen en als dat nog tekortschiet, kun je de JMeter open sourcecode uitbreiden en je eigen elementen in Java ontwikkelen. Het is echter minder geschikt voor doorsneetesters. Bovendien is het niet een heel productieve tool. Commerciële tools zoals HP Mercury Loadrunner en Borland SilkPerformer en Neotys Neoload zijn productiever. Echter, om het tool te gebruiken is vaak wel een opleiding nodig en kan de aanschaf een flinke investering vergen.

Traagheid wordt vaak genegeerd als argument en snellere hardware dan als oplossing gezien.



Figuur 1. Uitvoeren van een performance test in Apache JMeter.



Figuur 2. JAMon API start() en stop() calls in een Spring interceptor.

Performance testen vanuit de cloud

De opkomst van cloud computing biedt nieuwe mogelijkheden voor performance testen. Een 'elastic cloud' zoals Amazon's EC2 of Google App Engine biedt de mogelijkheid om als het nodig is automatisch en snel op te schalen met het aantal servers waarop de applicatie draait en alleen te betalen voor het daadwerkelijke gebruik. Dit wordt doorgaans gedaan om een toenemende belasting aan te kunnen met behoud van goede responsetijd. Voor performance testen kan de cloud juist andersom gebruikt worden: voor het inzetten van veel test clients om de betreffende applicatie representatief te belasten. Deze test clients tesamen vereisen doorgaans veel hardware welke slechts voor een klein deel van de tijd daadwerkelijk benut wordt. Hierdoor kan de inzet van de cloud voor load testen op de momenten dat het nodig is een geschikte oplossing zijn en een goedkoper alternatief dan de hardware zelf aanschaffen en in eigen beheer nemen. Meerdere open source en commerciële performance test tools bieden inmiddels de mogelijkheid om te testen vanuit de cloud.

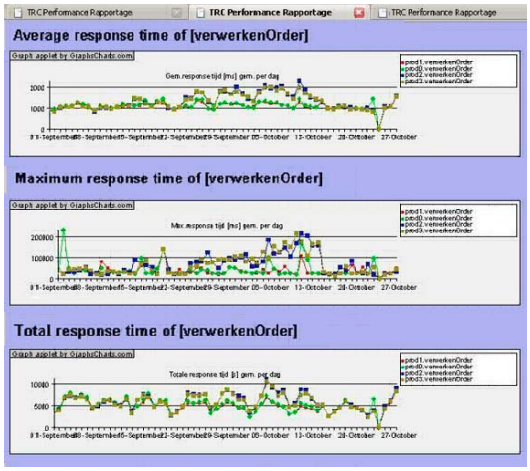
Stap 4: Continu testen

Met een representatieve test vlak voor de in productiename van een applicatie wordt voorkomen dat in productie dure verrassingen tevoorschijn komen. Er komen echter nog steeds dezelfde verrassingen tevoorschijn, alleen wat eerder en met wat minder impact. Om kosten te besparen en grote benodigde refactorings te voorkomen, is het zaak om zo vroeg mogelijk performancetesten uit te voeren. Hier geldt net als met andere softwaredefecten en Quality Assurance: hoe later in het ontwikkelproces defecten aan het licht komen, hoe meer geld het kost.

Stap 5: Monitoring & diagnosticeren

Als een nieuwe release in productie gaat, is het de vraag wat er gaat gebeuren en of het zal voldoen aan voorspellingen uit de test- en acceptatieomgevingen. Het is dus zaak om nauwgezet te monitoren wat er met de performance en beschikbaarheid gebeurt. Er zijn allerlei tools en services beschikbaar die websites monitoren op beschikbaarheid en responsetijd van pagina's zoals uptrends, site24x7, dotcom-monitor en pingdom. Ze bekijken de applicatie als black box en meten ééns in de zoveel minuten.

Om bij calamiteiten de goede maatregelen te nemen is het nodig om te kunnen achterhalen waar het probleem zit. Daarbij is het essentieel om op meerdere niveaus en op meerdere applicatieonderdelen te monitoren. Denk aan de niveaus: hardware, besturingssysteem, applicatieserver, webserver, database en applicatie. Binnen een Java applicatie kan dit met JAMon: een open source tijdsmeting API. In de basis werkt dit als een stopwatch met een start() en stop() aanroep. Elke te meten methode krijgt een eigen stopwatch of counter die statistieken bijhoudt zoals aantal, gemiddelde, maximum, standaard deviatie, etcetera en deze informatie kan vervolgens opgevraagd worden. Deze aanpak zorgt voor een lage memory en performance overhead. Het is zinvol om inkomende aanroepen zoals webrequests en uitgaande aanroepen naar bijvoorbeeld een database te meten en ook onderdelen zoals Spring-beans, EJB's en DAO's. Dit meten op onderdelen is niet alleen relevant bij nieuwe releases, maar de trends en de gebruikspieken zijn ook zinvol om te monitoren om daarmee problemen snel op te kunnen lossen en te voorkomen. Open source tool JARep biedt de mogelijkheid om de JAMon data uit een cluster in een database vast te leggen en hier trends en veranderingen grafisch mee te kunnen monitoren.



Figuur 3. JARep toont de oplopende trend tot 15 oktober op twee van de vier machines.

Stap 6: Tunen op basis van bewijs

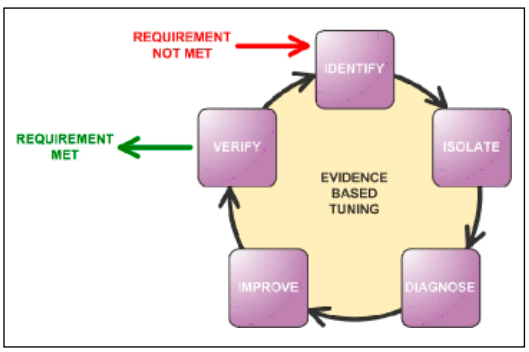
Als een applicatie niet snel genoeg blijkt te zijn, zal performance tunen een oplossing bieden. Tuning kan plaatsvinden op meerdere niveaus. Toevoegen van hardware kan een goedkope oplossing zijn. Echter, als er hardware wordt toegevoegd op de plek waar niet de bottleneck zit, heeft dit weinig zin. Belangrijke stappen in het tunen zijn daarom het identificeren welke pagina's of services niet aan de

Het is dus zaak om goed te monitoren wat er gebeurt bij een nieuwe release.

requirements voldoen en het isoleren van het probleem: waar zit het precies, in welke laag, in welke component. Dit kan met testen en monitoring op onderdelen duidelijk worden. De volgende stap is het stellen van de diagnose. Eigenlijk komt dit neer op het verzinnen van een hypothese waarom dat component zo traag is. Dit kan bijvoorbeeld zijn een missende of verkeerde index op een databasetabel of het uitvoeren van te veel kleine queries. Vervolgens wordt de verbetering op de component doorgevoerd, uitgaande van de hypothese. Tenslotte is het zaak om te verifiëren of de verbetering ook de verwachte versnelling teweeg brengt. Zo ja, dan was de gestelde hypothese juist en is er een versnelling als resultaat. Zo niet, dan is er iets mis met de hypothese en is er een alternatieve hypothese nodig. Zodra voldaan is aan de requirements, is het tunen klaar. De juiste tools zijn hierbij onontbeerlijk: performance testtool, monitoring tool, enterprise profiler, heap monitor, etcetera. In de praktijk heb ik ontwikkelaars vele dagen zien werken aan veronderstelde performanceverbeteringen die achteraf totaal niet bleken te helpen, of de applicatie zelfs tráger maakten en bovendien slecht waren voor de onderhoudbaarheid en flexibiliteit. Dit komt omdat ontwikkelaars gewend zijn om functionaliteit te maken met sourcecode en begrijpelijkerwijs vanuit sourcecode werken om performance proberen te verbeteren. Wat hier mist is meten is weten. Dit is ook wat ik ontwikkelaars bijbreng in onze performancetraining. De ervaring in de praktijk leert bovendien dat het zaak is om elke veronderstelde verbetering apart te beoordelen en alleen door te voeren zodra is bewezen dat het daadwerkelijk helpt.

Stap 7: Delen van verantwoordelijkheid over de keten

Als er een calamiteit in productie optreedt, betekent dit doorgaans stress. Bij een performanceprobleem in productie wordt er nogal eens gezwartepiet. De DBA zegt dat hij heeft gekeken en dat er niets mis is met zijn database. De netwerkbeheerder zegt hetzelfde over zijn netwerk, de applicatieserverbeheerder over zijn applicatieserver, de softwareontwikkelaar over zijn code en de backendbeheerder over het backend. Het ligt niet aan hen, maar aan de ander.



Figuur 4. Performance tune cyclus.

Dikwijls wordt de applicatie vanuit het ontwikkelteam over de muur gegooid naar de beheerorganisatie en gelden verantwoordelijkheden slechts aan één kant van die muur. Als softwareontwikkeling, testen en/of beheer is uitbesteed aan externe partijen, dan kan dit een netelige situatie opleveren. Voor je het weet wordt er met contracten en juridische procedures geschermd en is de samenwerking ver te zoeken. De hakken staan in het zand, de kosten lopen op en kostbare tijd gaat verloren.

Eilandvorming en zwartepieten draagt niet bij aan een oplossing.

Achterhalen van welk onderdeel verantwoordelijk is voor de vertraging is voor een deel op te lossen met tooling die over de gehele keten monitort en tooling die vroeg in het ontwikkelproces gebruikt wordt. Maar er komt meer bij kijken dan alleen tooling. Ervaring met en kennis van de tooling en technologieën is noodzakelijk evenals prioriteit voor de juiste inzet van de tools. Het allerbelangrijkste is echter om de eilandvorming en het zwartepieten tegen te gaan, gezamenlijk als performanceteam te opereren en verantwoordelijkheid gedeeld te krijgen.

Conclusies

In de groeiende online wereld met assertieve klanten is het cruciaal geworden dat webdiensten altijd beschikbaar en altijd snel genoeg zijn. Om de snelheid van webdiensten altijd goed te krijgen is geen sinecure, de aanpak is hierbij essentieel. Ervaring uit de praktijk heeft ons naar een aanpak geleid die bestaat uit zeven stappen en te omschrijven is met meten is weten in de ontwikkelcyclus en de keten. Deze zeven activiteiten resulteren in een pro-actieve aanpak die zich heeft bewezen in de praktijk van onze klanten. Het levert een manier van werken op die goede performance en beschikbaarheid verzekert voor een webwinkel of andere onlinedienst. «

De aanpak is essentieel om de snelheid van webdiensten altijd goed te krijgen.