Linnéuniversitetet
Kalmar Växjö

Examensarbete

Prestandajämförelse mellan olika Ramverk/språk

Författare: Josef Ottosson
Handledare: Johan Leitet
Examinator: Johan Leitet
Termin: VT13
Ämne: Datavetenskap
Nivå: B nivå
Kurskod: 1DV42E

Abstrakt

Rapporten undersöker vilket språk/ramverk som är snabbast vid olika dynamiska operationer vid webbutveckling. Rapporten grundar sig på en applikation som har tagits fram i fyra olika språk, Django, Ruby on Rails, PHP och Node.JS. De dynamiska operationer som testas är skrivning/läsning till/från fil och databas. Målet med arbetet är att förse webbutvecklare med konkret data för att underlätta vid val av språk inför kommande applikationer. Arbetet syftar också till att visa utvecklare av de olika språken var det finns rum för förbättring.

Nyckelord

Webbprogrammering, prestanda, dynamisk, ramverk

Tack till Christer, Stefan och Bob på Isotop för möjligheten att genomföra exjobbet tillsammans med Isotop. Även ett stort tack till mina opponenter Jesper och Johan för värdefull feedback.

Innehåll

1 Inledning

Det finns väldigt många olika språk och ramverk när det kommer till webbutveckling. Ett ramverk är ett bibliotek som består av olika funktioner som underlättar och snabbar upp utvecklingen av webbapplikationer[1]. Detta arbete syftar till att ge stöd och vägledning åt webbutvecklare vid val av verktyg. Det finns många fördelar med att använda sig av ett ramverk, förutom ovan nämnda så är det oftast väldigt många andra utvecklare som har använt ramverket. Detta innebär att det alltid finns hjälp att få om man väljer att använda sig utav ett ramverk. De flesta ramverk är Open Source[2], vilket innebär att ramverkets kod är helt publik och alla kan föreslå ändringar i koden. Det innebär att koden som ramverket använder är välgranskad och vältestad.

Allt med ramverk är dock inte positivt, en av de största nackdelarna är att man som utvecklare inte har full insyn i de tekniska detaljerna, vilket kan göra det svårt att felsöka sin applikation om/när eventuella fel uppstår. Om ramverket även är open source så innebär det, som tidigare nämnt, att koden är helt publik och tillgänglig för alla. En säkerhetsbrist i ett ramverk kan få förödande konskevenser för alla applikationer som bygger på det. Ett exempel på det är när wordpress[1]1 grundare Matt Mullenweg gick ut och varnade alla som använde “admin” som användarnamn att genast byta till något annat eftersom de hade upptäckt en bugg som utnyttjats av hackare[3].

Eftersom det finns många olika ramverk så är det lätt hänt att man tappar greppet om dem; vad är det här ramverket bra på? Varför ska jag välja det här?

Denna rapport undersöker vilket ramverk av Ruby on Rails[2] och Django[3] som är snabbast vid olika dynamiska operationer. Samma dynamiska operationer kommer även testas på en vanlig PHP-applikation byggd enligt MVC modellen[4] och för en Node.JS-applikation[4]. PHP-applikationen tjänar som en referens i undersökningen då språket har funnits med länge och de flesta utvecklare har någon gång använt sig av det. Motsatsen gäller Node.JS som är ett ramverk som är relativt nytt(skapades år 2009) och har fått en väldigt stor spridning på kort tid. Node.JS är intressant på grund av att det använder sig av samma språk både på klienten och servern; nämligen Javascript[5].

Alla applikationer kommer att prata med samma databas, en såkallad NoSQL-databas[5], närmare bestämt MongoDB[6].

---

[1] Wordpress är ett stort CMS/blogg-verktyg, www.wordpress.org

[2] http://rubyonrails.org/

[3] https://www.djangoproject.com/

[4] http://nodejs.org/

[5] Javascript är ett scriptspråk som används direkt i webbläsaren. https://developer.mozilla.org/en-US/docs/Web/JavaScript

2Bakgrund

Det här är ett examensarbete för Webbprogrammerarutbildningen i Kalmar.

Det utförs ihop med Webbteknikbyrån Isotop AB[6] som ligger i Stockholm. Rapporten kommer visa hur snabbt ett ramverk är vid en specifik dynamisk operation och om det finns några eventuella skillnader och vad dessa isåfall beror på. De dynamiska operationer som kommer mätas är sådanana operationer som en utvecklare normalt genomför många gånger i ett projekt:

- Läsa/skriva data ifrån en databas/fil
- Regexoperationer(Hitta diverse ord i ett längre stycke text etc)
- Modulos/divisionsberäkningar

Det har inte gjorts någon konkret mätning av dessa punkter innan. De som diskuterar detta på olika forum/bloggar[7][8] pratar ofta om vilka olika för/nackdelar det finns med ramverken och drar sedan en personlig slutsats om vilket som är bäst utan att ha några konkreta bevis för det. Det finns olika verktyg för att mäta prestanda på serversidan, ett av de mer populära är Microsoft Web Application Stress Tool[7]. Det används främst för att testa hur webbapplikationen hanterar många samtidiga anslutningar. Detta verktyg kommer inte att användas i det här testet men för vidare forskning är det ett bra alternativ. Mer om det under punkten “Framtida forskning”. Målet med rapporten är att hjälpa utvecklare vid deras val av ramverk men också att uppmärksamma skaparna av ramverken på eventuella flaskhalsar.

3Frågeställning

- Vilket språk/ramverk av Django, Ruby on Rails, PHP och NodeJS är snabbast vid dynamiska operationer såsom skrivning till databas/fil?

---

[6] www.isotop.se

[7] http://support.microsoft.com/kb/231282

4Metod

Metoddiskussion

Det finns några få saker att påpeka i metoden som kan ge ett missvisande resultat:

- **Användandet av en PHP-proxy**
I ett skarpt projekt så äger man som utvecklare (oftast) hela miljön själv och då behöver man inte använda sig utav en proxy. I det här fallet används proxyn eftersom tanken är att i framtiden testa på olika webbservrar och då kommer man vara tvungen att använda sig utav en proxy. De nackdelar som finns med proxyn är att det kan ta lite längre tid att utföra anropen men eftersom alla applikationer använder sig utav den så blir mätningen ändå rättvis.

- **Testningen sker lokalt**
Eftersom testningen sker lokalt så finns det vissa saker som kan påverka mätningen. Några av dessa saker kan exempelvis vara att operativsystemet får för sig att indexera filer, leta efter uppdateringar till program eller liknande som kan påverka prestandan vid mätningen. För att undvika detta har internet kopplats ifrån och datorn har startats om efter varje testrunda.

- **Mätvärdena är ojämna**
Varje test har körts 1000 gånger för att sedan baserat på den data som kommit fram räkna ut ett medelvärde. Vid vissa fall (särskilt gällande Ruby on Rails) har mätvärdena stuckit iväg en gång var hundrade gång ungefär. Det tas ingen hänsyn till detta utan alla värden som kommit in är med i beräkningen. Det noteras dock och visas i diagrammet.

- **Olika kunskapsnivåer i de olika språken**
Det är 4 olika språk som det skrivs applikationer i och kunskapsnivån kanske inte är lika hög i alla de språken utan det finns säkert kod som kan skrivas bättre. I de här testen har applikationerna utvecklats med best-practices i åtanke. När det kommer till kommunikation med databasen har de riktlinjer som finns i MongoDB dokumentationen följts. Om man som utvecklare är expert på ett språk så vet man vilka operationer som kan vara tidskrävande och då försöker man använda en bättre lösning. Det här kan vara ett problem om man inte använt språket förut för då riskerar man att gå i diverse fallgropar.

- **Nytta av mätningen**
De mätresultat som tas fram påvisar hur applikationen/språket beter sig vid upprepande anrop. Om man som utvecklare funderar på att bygga ett API som skall leverera data konsekvent över tid så är den här rapporten ett bra underlag. Funderar man istället på att skapa en blogg som inte har samma krav på sig gällande stabila svarstider vid upprepande förfrågningar finner man ingen större nytta av den här rapporten.

Tillvägagånssätt

För att besvara frågeställningen så kommer det att skapas en applikation för varje språk och en testklient. Applikationerna byggs som ett API[9] med följande funktioner:

- **GetAllCities** - Hämtar alla rader ifrån en databas och gör om dessa till JSON format[8] och returnerar sedan dessa.
- **GetAllCitiesWhere** - Hämtar alla rader där ett visst villkor uppfylls och gör om dessa till JSON format och returnerar sedan dessa.
- **ReadFile** - Läser in en fil från disk. Filen är 3.1 MB stor. Samma fil används av alla applikationer
- **ReadAndSaveNew** - Läser in samma fil som ovan, går igenom filen med hjälp av regex[9] och byter ut alla strängar[10] "_id" till "id" för att sedan spara som en ny fil. Den aktuella strängen förekommer en gång per rad i slutet av raden.
- **CalculateModulus** - Räknar ifrån 0 till 10000000(tio miljoner) och kollar om det aktuella talets rest är lika med noll, i så fall läggs det till i en array[11]. Arrayen med talen returneras sedan.
- **SelectAndUpdate** - Hämtar alla rader där ett visst villkor(alla städer med en befolkning under 10 000) uppfylls för att sedan kontrollera om en egenskap på objektet är skrivet med stora bokstäver. Om så är fallet görs egenskapen om till små bokstäver och sparas i databasen. Om så inte är fallet görs egenskapen om till stora bokstäver för att sedan sparas i databasen.

Ovan nämnda funktioner valdes då de är frekvent förekommande för en utvecklare i dennes vardag. Modulusberäkningen är med för att se hur snabbt språket/ramverket är på att räkna ut matematiska operationer då de tenderar att vara väldigt cpu-krävande[10] att utföra en stor mängd matematiska beräkningar.

Applikationsspecifik information

● **PHP** - Applikationen är byggd enligt ett MVC mönster där varje inkommande anrop styrs vidare till rätt metod. För att få applikationen att kunna använda MongoDB används den officiella MongoDB-drivrutinen[11]. PHP-applikationen körs på en MAMP PRO server med version 2.1.1 och PHP version 5.3.14.

---

[8] JSON används för att representera data i textform. http://www.json.org/

[9] Ett "regex"(*regular expression)* är ett textmönster som används för att matcha mot text-strängar.

[10] Strängar är en typ som består av följder av tecken. Till exempel så är "Jag heter Josef" en sträng

[11] En array är ett sammanhängande antal element av samma typ som representeras med ett variabelnamn.

- **Ruby on Rails** (Ruby version 2.0.0p0, Rails version 3.2.13) - Ett vanligt rails projekt är skapat med hjälp av new[12] kommandot i terminalen för rails. För att använda databasen används tre stycken så kallade *Ruby Gems*[13]; *Mongo*, *bson_ext* och *mongoid*. Inkommande requests routas automatiskt till rätt metod efter att ha kommenterat bort följande rad i routes.rb konfigurationsfilen i rails projektet: *match ':controller(/:action(/:id))(.:format)'*.

- **Django** (version 1.3.0.) - En specialversion av Django med stöd för No-SQL databaser används då Django från början inte har stöd för det[14]. Versionen heter Django Nonrel[12]. Django Toolbox[13] används för att få stöd för fler datatyper i modelldeklarationen. Slutligen används MongoDB-Engine för att kunna ansluta till databasen.

- **Node.JS** (version 0.10.4) **-** Inkommande anrop tas om hand och routas till rätt metod med hjälp av en requesthandler(annat namn för controller i en MVC-applikation) som har skapats efter de rekomendationer som finns[15]. MongoJS[14] används för att ansluta till databasen. Val av följande moduler/tillägg har gjorts efter att ha jämfört de alternativ som eventuellt finns och även efter test av vilken som har varit snabbast att använda. Den modul/det tillägg som har varit snabbast har också valts.

Testklient

Testklienten är byggd i Javascript ihop med biblioteket jQuery[15]. JQuery har använts eftersom det underlättar väldigt mycket vid javascriptutveckling och särskilt vid Ajaxanrop[16]. Applikationen låter användaren välja vilket språk, metod och hur många gånger som testet skall utföras. Alla tester som ligger till grund för den här rapporten har körts 1000 gånger var för att få ett så rättvisande resultat som möjligt och för att undvika slumpmässiga fel.

En mätning går till på följande sätt:

1. Användaren väljer språk, metod och antal gånger testet skall genomföras och startar testet
2. Testklienten skapar ett datumobjekt innan anropet sker till metoden med hjälp av jQuerys "beforeSend"[17].
3. Ett a*synkront*[18] Ajaxanrop sker till metoden(via en "PHP proxy"[19], mer om det nedan).
4. När anropet är klart skapas ett nytt datumobjekt i metoden "success.
5. Datumobjekten subtraheras för att få fram tidsskillnaden i millisekunder.

---

[12] http://www.allbuttonspressed.com/projects/django-nonrel

[13] Ger stöd för flera datatyper i modellfälten. https://github.com/django-nonrel/djangotoolbox

[14] En nodejs-modul som ger stöd för MongoDB. https://github.com/gett/mongojs

[15] http://jquery.com

6. Tidsskillnaden sparas i en databas ihop med vilken metod och språk som användes.
7. Testet upprepas tills det har körts så många gånger som användaren valde vid start.

Varje anrop går genom en proxy på grund utav att ajaxanrop inte är tillåtna “Cross Domain” på grund av säkerhetsskäl[20]. Därför måste man gå igenom en server som i sin tur skickar vidare anropen.

När mätningarna är färdiga skapas ett diagram med hjälp av javascriptbiblioteket Highcharts[16]. Se bild 1.

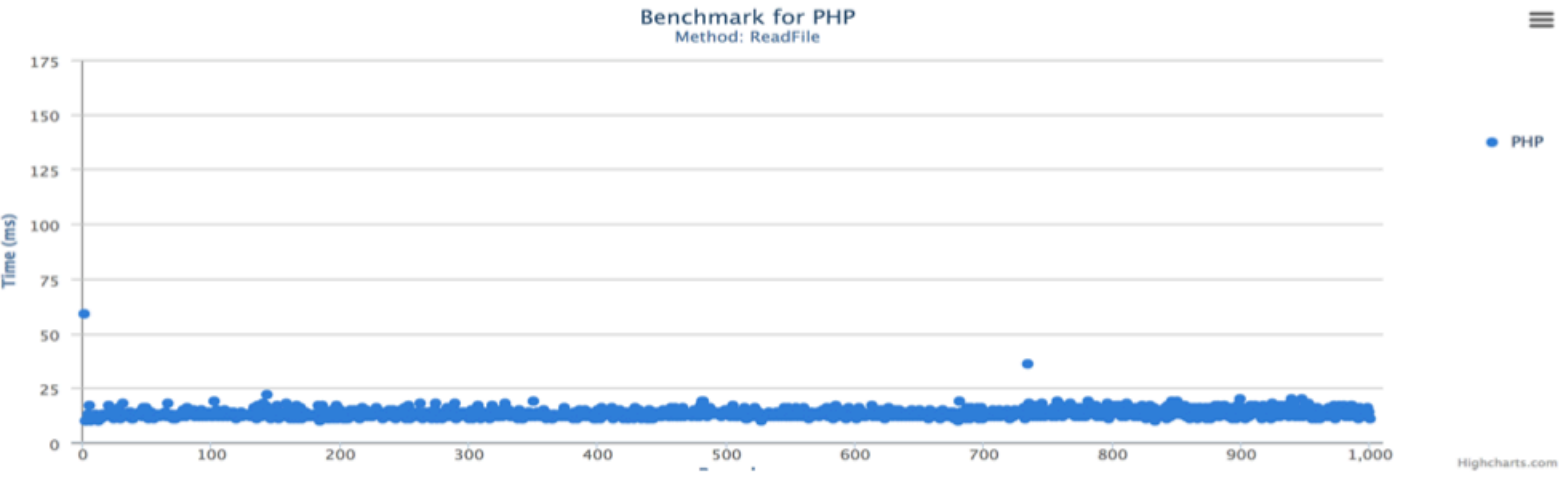


Bild 1.

Den diagrambild som visas när en mätning är klar är till för att kontrollera om det varit några stora avvikelser under testet, om så är fallet har testet gjorts om.

4.3. Testmiljö.

Samtliga tester har utförts på en Macbook Pro Retina 13 2012 med följande specifikationer:

- 8GB 1600 MHz DDR3 ram.
- Intel Core i5 2.5 GHz processor
- 256 GB SSD[21]
- Mountain Lion 10.8.3
- Webbläsare: Google Chrome 26.0.1410.65
- Databas - MongoDB innehållandes en collection, “cities”, med 29347 rader. Inga index är satta förutom på ID-fältet som sätts automatiskt vid skapandet avdatabasen. Testerna har utförts med internet avslaget(airport avstängt) och mellan varje test har

[16] www.highcharts.com

webbläsaren startats om. Språken har testats var för sig och när ett språk varit färdigtestat har datorn startats om. Alla program i autostarten har även stängts av.

Utvärderingsmetod

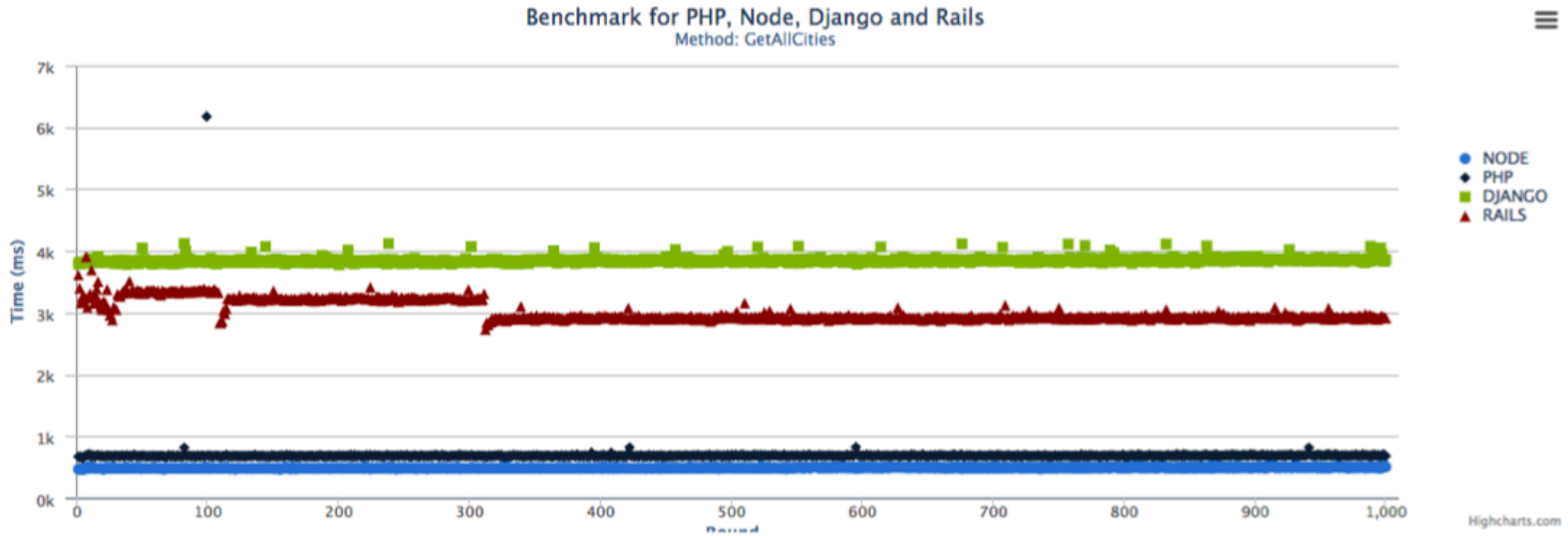
Språket/ramverket kommer att jämföras beräknat på medelvärdet av den data som testklienten samlat in vid mätning av de olika metoderna. Medelvärdet används eftersom det ger en mer rättvis bild av hur snabbt det har gått över tid. Mätningarna har visat att den långsammaste och snabbaste tiden kan variera rejält och därför skulle det vara missvisande att utgå från enbart den snabbaste respektive långsammaste tiden.

5Resultat

GetAllCities

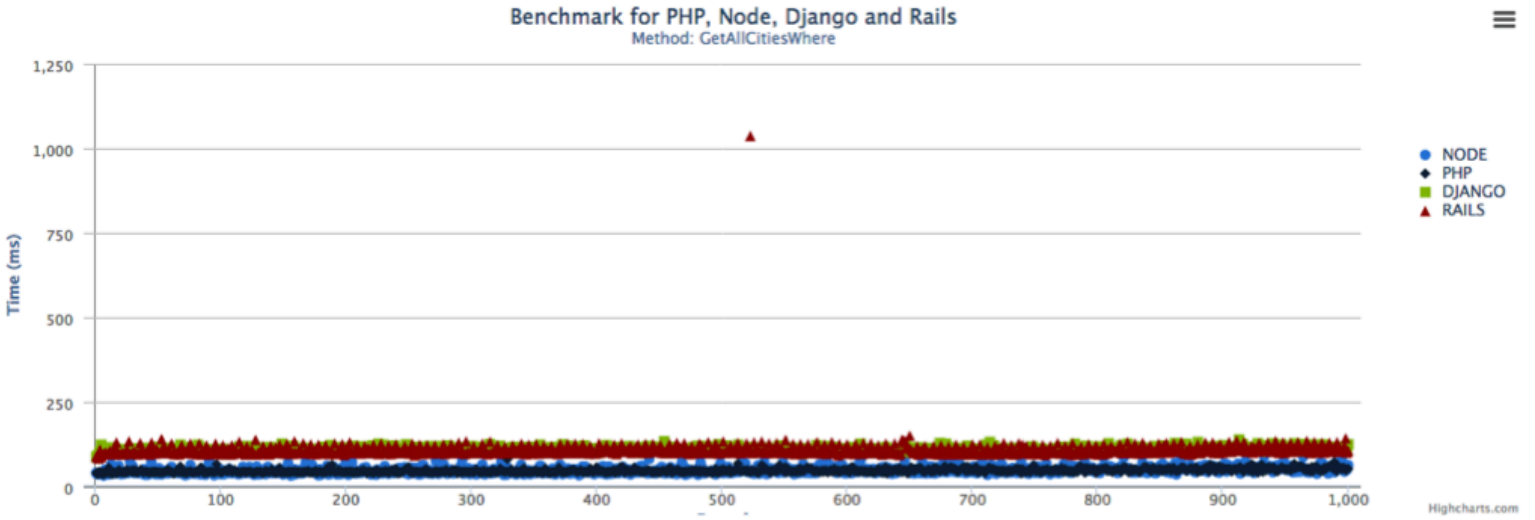
Genomsnittligtid:

- 491ms - **Node**
- 691ms - PHP
- 3022ms - Rails
- 3845ms - Django



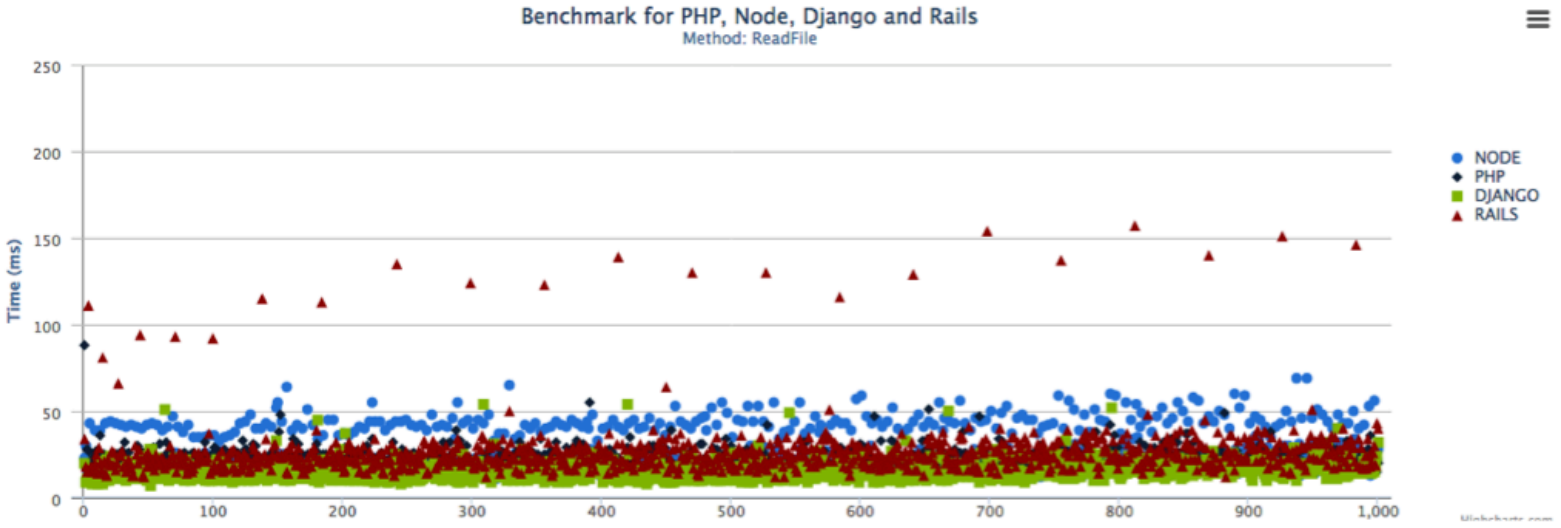
GetAllCitiesWhere
Genomsnittligtid:

- 46ms - **PHP**
- 47ms - Node
- 106ms - Rails
- 109ms - Django



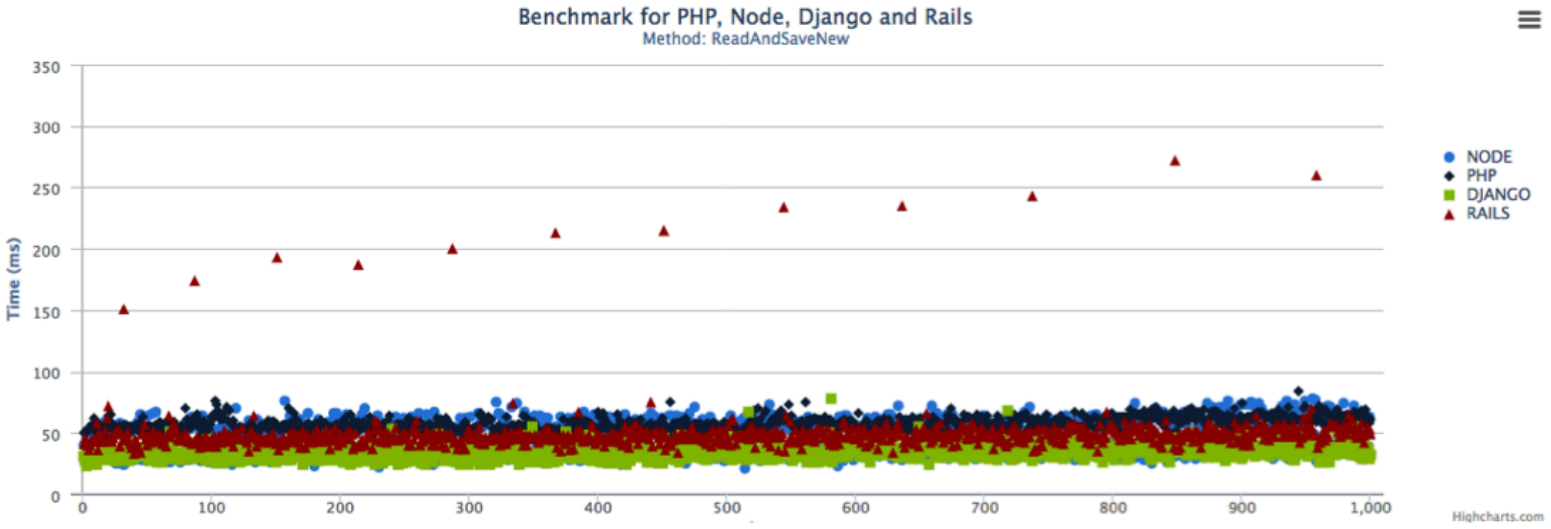
ReadFile
Genomsnittligtid:

- 15ms - **Django**
- 21ms - PHP
- 25ms - Node
- 26ms - Rails



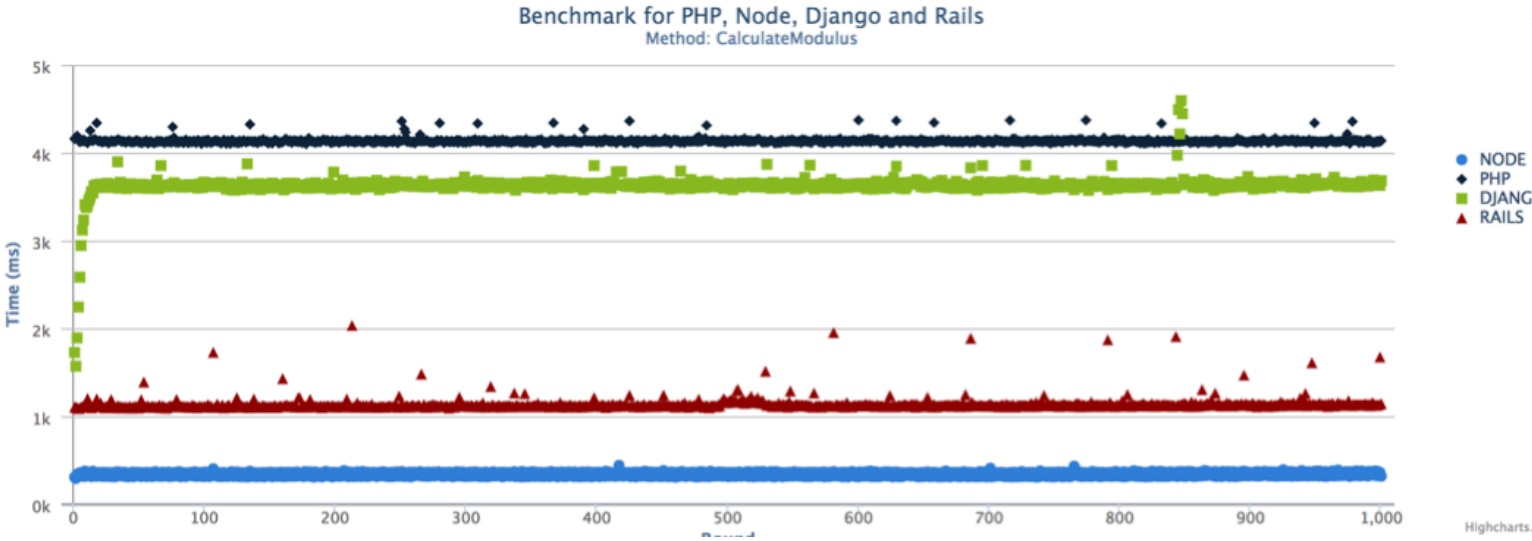
ReadAndSaveNew
Genomsnittligtid:

- 34ms - **Django**
- 45ms - Node
- 49ms - Rails
- 57ms - PHP



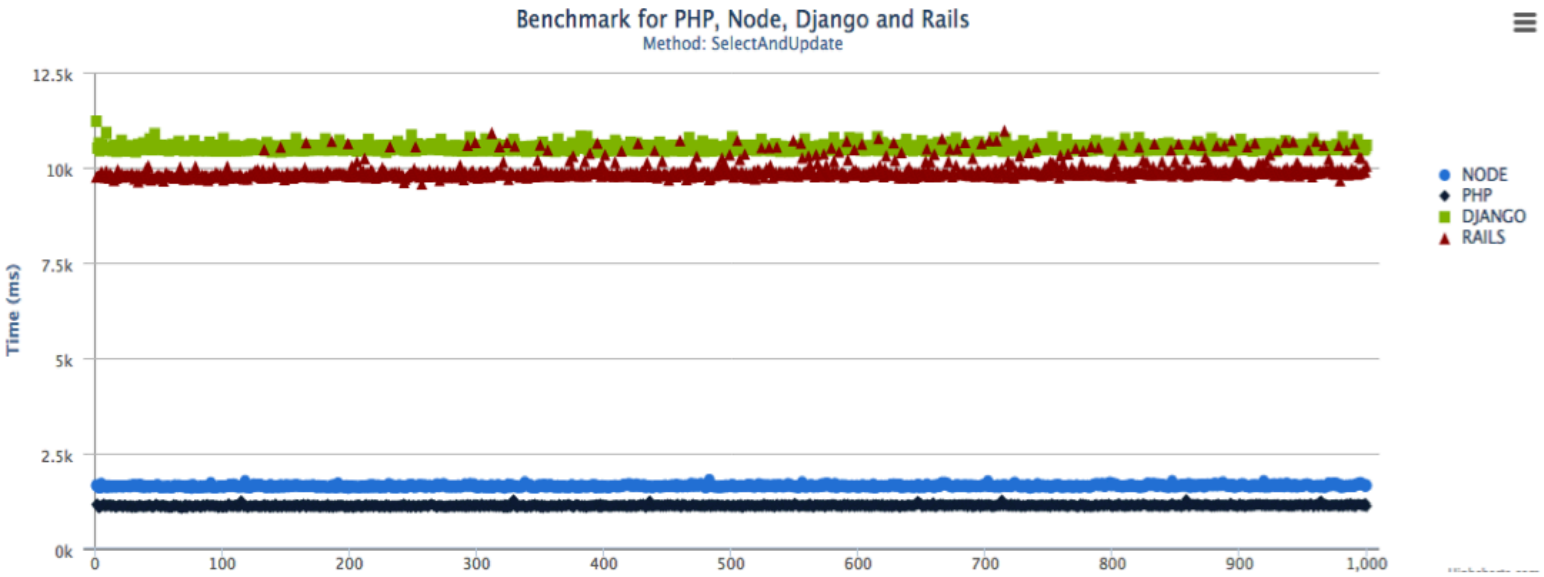
CalculateModulus
Genomsnittligtid:

- 339ms - **Node**
- 1134ms - Rails
- 3628ms - Django
- 4135ms - PHP



SelectAndUpdate
Genomsnittligtid:

- 1126ms - **PHP**
- 1646ms - Node
- 9912ms - Rails
- 10528ms - Django



6Slutsats

De resultat som har samlats in var ganska väntade(att ramverken generellt skulle vara långsammare), här följer en genomgång för varje metod och några hypoteser till resultaten.

GetAllCities
NodeJS och PHP är väldigt mycket snabbare än vad de två ramverken Django och Rails är. Det är ett förväntat resultat, då det sker en modellbindning[17] i Django och Rails vilket gör att de är långsammare. PHP och NodeJS har ingen modellbindning som fungerar ihop med MongoDB. I PHP-applikationen lagras datan som en array som sedan görs om till en JSON-sträng. NodeJS applikationen arbetar på samma sätt.

GetAllCitiesWhere
Samma resultat och anledning som ovan. Det är mycket färre rader som returneras men det syns fortfarande tydligt att Django och Rails är långsammare än PHP och NodeJS.

ReadFile
Språken/ramverken har här gemensamt att mätningarna inte är lika stabila över tid som de är vid de andra metoderna. Detta kan bero på flera saker:

[17] Modellbindning innebär att all data ifrån databasen mappas till objekt som kan användas direkt i applikationen.

- Operativsystemet hinner inte med att öppna/stänga filerna innan nästa anrop sker och detta innebär att nästa anrop får vänta tills filen är redo för att öppnas igen.

- Olika implementationer av Garbage-collection[22], orsakar en tydlig felaktighet i mätningarna av Rails applikationen, då ungefär var 50:e körning visar extrema värden. Det som sker är att servern frigör minne vilket påverkar nästkommande mätning på ett negativt sätt. Det språk/ramverk som är snabbast vid inläsning av fil är Django och det är även det språk/ramverk som är stabilast vid mätningen.

ReadAndSaveNew
Även här varierar mätvärdena väldigt mycket mellan långsamma och snabba värden och det är av samma anledningar som ovan. Här blir Rails garbage-collection ännu tydligare. Även här är Django snabbast.

CalculateModulus
NodeJS är överlägset snabbast och stabilast. Mätningarna för rails visar även här en stor differens men dock inte lika regelbundna som i de andra metoderna så det är svårt att dra en slutsats av vad det beror på. Något som är intressant här är Django. Vid de 10 första körningarna så stegrar tiden från 1500ms upp till ungefär 3600ms. Vid kontroll av cpu-användningen med hjälp av aktivitetskontrollen som är inbyggd i Mountain Lion så ser man att minnesanvändningen under de 10 första körningarna ligger runt 100mb för att sedan lägga sig stabilt på 450mb under resterande mätningar. Det som tar mest tid är att lägga till numrena i arrayen för Django, om det bortkommenteras så ligger genomsnittet på cirka 600ms. PHP är långsamt här av samma anledning som Django, array hanteringen. Kontentan av detta är alltså att NodeJS och Rails hanterar stora arrayer på ett bättre sätt än Django och PHP.

SelectAndUpdate
Den här metoden påvisar tydligast skillnaden mellan ett ramverk kontra språk. PHP och NodeJS är överlägset snabbast, PHP har ett medelvärde på 1100ms och NodeJS ligger runt 1650ms. Ramverken ligger på nästan 10000ms(10 sekunder) i snitt och Django ligger runt 10500ms(10.5 sekunder). Det är på grund av, som ovan nämnt, modellbindningen men det verkar också vara så att drivrutinerna för MongoDB är långsammare för Django/Rails när det kommer till vissa frågor. I det här fallet utförs en "Select all cities with a population less than 10 000". Om man kör den frågan i konsolfönstret för MongoDB så tar det ungefär 1000 ms att välja alla rader medans det tar cirka 3000 ms att utföra samma fråga med Rails/Django.

Sammanfattning av resultatet
NodeJS är väldigt snabbt men framförallt väldigt stabilt när det kommer till frågor mot databas, sätta in data och räkna ut modulus. Det är inte lika stabilt när det kommer till att läsa filer men felet behöver inte ligga i NodeJS grundfunktionalitet utan det kan bero på hårdvarans begränsingar, till exempel läs/skrivhastigheten på disken[18].

NodeJS är ett ungt språk som inte ens har nått version 1.0 ännu (förnärvarande på version 0.10.7) men det är redan väldigt snabbt men framförallt stabilt. Att det är stabilt

[18] http://www.overclockers.com/forums/showthread.php?t=678783

backas upp av de tester jag har genomfört men också det här testet av Jakub Skvara som har jämfört NodeJS httpserver mot en Apache server som kör PHP[23]. NodeJS vann alla de tester som han utförde. Liknande resultat har tagits fram av Matt Knight, han testade PHP mot NodeJS och kom fram till att NodeJS var ungefär 50 gånger snabbare än PHP[24].

NodeJS placerade sig antingen etta eller tvåa i alla mina test förutom ReadAndSaveNew där det kom på en tredje plats. PHP applikationen är även den väldigt snabb när det kommer till att kommunicera med databasen men när det kommer till cpu-intensiva operationer såsom CalculateModulus så är den nästan fyra sekunder långsammare än vinnaren. När det gäller läsning/skrivning till fil så är applikationen väldigt snabb på att läsa in filen men när det gäller att spara den nya filen är den långsammast i testet.

Django är det språk/ramverk där placeringarna i testen varierar mest, när det gäller att läsa/spara filer så är det snabbast men när det kommer till kommunikationen med databasen är det långsammast. Om en utvecklare skall skapa en applikation som läser/skriver till fil väldigt mycket så är Django att föredra.

Rails är det enda språket/ramverk som inte bäst mätresultat i någon kategori. Det presterar bäst i CalculateModulus där det placerar sig på en andra plats. Värt att notera är att Rails är snabbare på att kommunicera med databasen än vad Django är, vid de två tyngre operationerna, GetAllCities och SelectAndUpdate är det 800ms respektive 600ms snabbare än Django.

Om man som utvecklare är ute efter ett språk som skall kommunicera mycket med en databas och man vill ha en konsekvent prestanda så tycker jag att NodeJS är rätt val. Det är fortfarande ett nytt språk men det är såpass stabilt och framförallt snabbt och utifrån denna undersökningen är NodeJS det bästa valet.

Om det är viktigt med modellbindning, alltså att man vill använda ett ramverk, så är mitt råd att använda sig av Rails. Det är snabbare än Django förutom när det kommer till att läsa/skriva till fil. Värt att notera är dock att garbage-collection kan ställa till det när det kommer till konsekventa svarstider.

Framtida forskning

Det vore intressant att testa hur applikationerna uppför sig på andra typer av servrar, exempelvis en Apache[19] server eller en Ngingx[20]. Det vore också intressant att se hur väl applikationerna skalar vid testning med flera kilenter samtidigt. Hur applikationerna hanterar sin garbage-collection och vilka skillnader den kan göra i prestanda är också en intressant sak att kolla närmare på. Det vore också väldigt intressant att undersöka modellmappningen för PHP och NodeJS. Detta är i dags läget inte möjligt att göra ihop med en MongoDB men det fungerar ihop med en MySQL-databas, det gör även Rails och Django.

---

[19] Den populäraste webbservern. http://httpd.apache.org/

[20] http://nginx.org/en/

Källförteckning

1.Dirk Riehle, "Framework Design: A Role Modeling Approach. Ph.D. Thesis, No. 13509. Zürich,."maj 2000. [Online],
Tillgänglig: http://www.riehle.org/computer-science/research/dissertation/diss-a4.pdf [Hämtad: 2013 0429]

2.Opensource.com, "What is Open Source?" 2010. [Online],
Tillgänglig: http://opensource.com/resources/what-open-source [Hämtad: 2013 0418]

3. Mathew J. Schwartz, "WordPress Hackers Exploit Username 'Admin'" april 2013. [Online],
Tillgänglig: https://www.informationweek.com/big-data/news/security/attacks/wordpress-hackers-exploit-username-a dmi/240152864?cid=SBX_bigdata_related_slideshow_Application_Security_big_data&itc=SBX_bigdata_r elated_slideshow_Application_Security_big_data
[Hämtad: 2013 0429]

4.”rj45”, "Simple Example of MVC (Model View Controller) Design Pattern for Abstraction” 2008. [Online],
Tillgänglig: http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design [Hämtad: 2013 0418]

5. NoSQL - edlich, "Your Ultimate Guide to the Non - Relational Universe!” 2009. [Online], Tillgänglig: http://nosql-database.org/ [Hämtad: 2013 0418]

6. 10gen, Inc, "MongoDB - Agile and Scalable” 2010. [Online], Tillgänglig: http://www.mongodb.org/ [Hämtad: 2013 0418]

7. Riccardo Poggi, "My Experience of Django vs Rails”, Jul 2010. [Online],
Tillgänglig: http://riklife.herokuapp.com/blog/2012/07/10/my-experience-of-django-vs-rails/ [Hämtad: 2013 0427]

8. Max Ivak, "Python vs Ruby and Django vs Rails”, Dec 2012. [Online], Tillgänglig: http://maxivak.com/python-django-vs-ruby-on-rails/ [Hämtad: 2013 0427]

9. Dave Roos, "HowStuffWorks ‘What is an API?’”. [Online],
Tillgänglig: http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm [Hämtad: 2013 0427]

10. ‘Leonid’, "Is MOD operation more CPU intensive than multiplication?”, 2011. [Online], Tillgänglig: http://stackoverflow.com/questions/4109330/is-mod-operation-more-cpu-intensive-than-multiplication [Hämtad: 2013 0427]

11. ‘dericker’, "Mongo PHP Driver”, Apr 2010. [Online], Tillgänglig: https://github.com/mongodb/mongo-php-driver [Hämtad: 2013 0501]

12. Ruby on Rails, "Ruby on Rails Guides: Getting started with Rails”, 2011. [Online], Tillgänglig: http://guides.rubyonrails.org/getting_started.html
[Hämtad: 2013 0502]

13. RubyGems Manual, "The RubyGems bookshelf”, 2010. [Online], Tillgänglig: http://docs.rubygems.org/
[Hämtad: 2013 0502]

14. Django Software Foundation, "NoSqlSupport - Django”, 2011. [Online], Tillgänglig: https://code.djangoproject.com/wiki/NoSqlSupport
[Hämtad: 2013 0502]

15. Manuel Kjessling, "The Node Beginner Book”, 2012. [Online], Tillgänglig: http://www.nodebeginner.org/#whats-needed-to-route-requests [Hämtad: 2013 0504]

16. Eric Rowell, "The Power of jQuery with Ajax”, Jun 2010. [Online], Tillgänglig: http://sixrevisions.com/javascript/the-power-of-jquery-with-ajax/ [Hämtad: 2013 0504]

17. The jQuery Foundation, "The Power of jQuery with Ajax”, 2010. [Online], Tillgänglig: http://api.jquery.com/jQuery.ajax/
[Hämtad: 2013 0504]

18. Stephen Chapman, "Asynchronous or Synchronous” [Online], Tillgänglig: http://javascript.about.com/od/ajax/a/ajaxasyn.htm [Hämtad: 2013 0504]

19. Ben Alman, "Simple PHP Proxy: JavaScript finally "gets" cross-domain!”, Jan 2010 [Online], Tillgänglig: http://benalman.com/projects/php-simple-proxy/
[Hämtad: 2013 0505]

20. ‘Kibbee’, "Why the cross-domain Ajax is a security concern?”, Jan 2009 [Online], Tillgänglig: http://stackoverflow.com/questions/466737/why-the-cross-domain-ajax-is-a-security-concern?lq=1 [Hämtad: 2013 0505]

21. Webopedia, "solid state disk” [Online],
Tillgänglig: http://www.webopedia.com/TERM/S/solid_state_disk.html [Hämtad: 2013 0506]

22. Hongli Lai, "Saving memory in Ruby on Rails with fork() and copy-on-write", april 2007. [Online], Tillgänglig: http://izumi.plan99.net/blog/index.php/2007/04/05/saving-memory-in-ruby-on-rails/ [Hämtad: 2013 0518]

23. Jakub Škvára, "node-js-vs-apache-php-benchmark", dec 2010. [Online],

Tillgänglig: https://code.google.com/p/node-js-vs-apache-php-benchmark/wiki/Tests
[Hämtad: 2013 0518]

24. Matt Knight, "Node.js vs PHP Performance – Maths", apr 2011. [Online], Tillgänglig: http://www.matt-knight.co.uk/2011/node-js-vs-php-performance-maths/
[Hämtad: 2013 0518]