Examensarbete

HÖGSKOLAN
I SKÖVDE

JÄMFÖRELSE AV JAVASCRIPT OCH PHP

När data lagras som JSONObjekt i relationsdatabaser eller NoSql-databaser

COMPARISON OF JAVASCRIPT AND PHP

When data is stored as JSONObject in relational databases or NoSQL-databases

Examensarbete inom huvudområdet Datalogi
Grundnivå 30 högskolepoäng
Vårtermin 2016

Anton Honkavaara Dahl

Handledare: Mikael Lebram
Examinator: Henrik Gustavsson

Sammanfattning

Trafiken till webbapplikationer ökar därför ställs det högre krav på att svarstiderna för användaren hålls nere även fast det blir högre trafik och mer data som behöver laddas till webbapplikationerna. Där det vanligaste språket för att hämta data ifrån databaser är PHP som är ett språk som varit med i många år. När det nu finns möjlighet till att skiva all kod i ett och samma språk som JavaScript med Node.js är frågan hur PHP står sig emot JavaScript i svarstider för användaren. Det blir också vanligare med NoSql databaser istället för RDBMS

Detta arbete gör ett experiment där JavaScript och PHP ställs emot varandra och kollar hur dom påverkar svarstider. Där även programmeringspåken tar användning av en RDBMS och en NoSql databas. Där resultatet visar på att det inte går att skilja dom åt om man kollar på svarstider.

Nyckelord: JavaScript, PHP, RDBMS, NoSql, JSONObjekt

Innehållsförteckning

1 Introduktion

Webben växer med varje år som går där trafiken till dom populäraste webbapplikationerna bara ökar och där användarna blir allt mer vana till med att svarstiderna för att använda dessa webbapplikationer blir lägre. Där enligt Chaniotis, Kyriakou och Nikolaos (2015) är Facebook en utav dom mest använda webbapplikationen med 1.23 miljard månadsvis aktiva användare. Där teknikerna samt utvecklingsmetoder för att utveckla webbapplikationer som kan upprätthålla små svarstider för all trafik blir allt fler.

Majoriteten av webbapplikationerna har Apache/Nginx som webbservrar enligt Chaniotis, Kyriakou och Nikolaos (2015) och på dessa webbservrar för att lagra data så används i majoritet RDBMS databaser. Där PHP är det språk som oftast pratar emellan klientsidan och databasen samt webbservern. Men på klientsidan så är det med majoritet JavaScript som tar hand hur innehållet ska presenteras och pratar med PHP för att olika typer av förfrågningar till databasen.

Men med nyare tekniker som Node.js där det blir möjligt att använda sig av att använda sig av JavaScript ifrån back-end till front-end. Där Chaniotis, Kyriakou och Nikolaos (2015) utvärderar hur det påverkar serversidan. Där även på databassidan så kommer nyare tekniker för att hantera data i så kallade NoSql databaser enligt Catell (2010).

Problemet med Chaniotis, Kyriakou och Nikolaos (2015) som denna studie spinner vidare på deras arbete är att dom inte tar upp hur svarstider påverkas för användaren utan enbart kollar på hur webbservern påverkas. Samt att dom inte använder sig av databaser i sin studie vilket är ett vanligt sätt att lagra data. Då kommer också problemet vilken typ av databas ska användas för att få ner svarstiderna. Vilket Yishan och Manoharan (2013) och Cattel (2010) mäter i sina studier. Men inte hur dom databaserna påverkas när PHP eller JavaScript integrerar med dom.

Frågeställningarna i denna studie fokuserar på svarstider och hur dessa påverkas när PHP med en Nginx webbserver byts ut emot ren JavaScript med Node.js som stöd och webbserver. Men även hur svarstiderna påverkas när data hämtas ifrån en NoSql databas istället för en RDBMS. Där den data som hanteras är lagrad i form av JSONObjekt.

Den metod som används för att svara på dom frågarna som ställts är experiment samt att Chaniotis, Kyriakou och Nikolaos (2015), Yishan och Manoharan (2013) och Cattel (2010) använder sig av experiment för att utvärdera deras frågeställningar. Experiment lämpar sig också till denna studie då det är variabler som byts ut för att få svar på hur svarstiden blir påverkad när dom olika variablerna byts ut.

Webbapplikationerna som implementerades är fyra olika stycken där alla har syftet att hämta olika mängd JSONObjekt ifrån antingen en RDBMS (MySql) eller en NoSql (MongoDB) databas. Där det programmeringsspråket som användes är PHP eller Javascript. Där PHP använde sig av Nginx som webbserver och JavaScript Node.js.

2 Bakgrund

Med varje år som går så växer antalet internet sidor på World Wide Web och därmed kraven på dom tekniker som utvecklarna använder sig av för att utveckla sidorna. Samt att webbapplikationerna blir mer komplexa, får mer genomströmning och allt fler utvecklare behövs för att skapa applikationerna. Som utvecklare eller beställare av en webbapplikation kan det vara svårt att veta vilka tekniker som är bäst lämpade till sin webbapplikation. Vilka språk som applikationen ska använda sig av för att ge användarna så bra upplevelse som möjligt. Där JavaScript som den mest dominerande web teknologin för klient sidan av programmeringen och PHP den främsta för back-end till serversidan som.

PHP är ett skript språk gjord för server sidan av webbapplikationer. Som är gjord för att kunna skapa dynamiska webbapplikationer. Som oftast paras ihop med en variation av databaser och webbservrar för att få fram webbapplikationer. De två vanligaste webbservrar som används är Apache och Nginx som består utav sextio procent av marknaden (Chaniotis, Kyriakou och Nikolaos 2015). Där Nginx är en öppen source-projekt som började utvecklas 2002 av Igor Sysoev. Det första publika släppet av koden var 2004 (Nginx Inc 2016).

Men med födelsen av server sida JavaScript Node.js May 2009 som ger en klient server integrering med enbart JavaScript. Till dessa olika webbservrar används olika typer av databaser för att lagra data som används till dom olika webbapplikationerna. Där dom vanligaste typerna är relationsdatabaser och NoSql databaser. Men eftersom Node.js är ny teknik saknas det forskning om det är mer optimalt till stora webbapplikationer och vilka databaser som ska användas. Därför finns det utrymme att vidare utveckla dom beräkningar som finns kring svarstider på de olika kombinationerna av dessa tekniker.

Svarstider i detta arbete är ifrån att användaren gör en HTTP förfrågan till att användarens sida med HTML, JavaScript och CSS är klar laddat. Enligt Mickens (2010) så förväntar sig en användare att en sida laddas klart på två sekunder eller mindre och fyrtio procent av användarna är inte villiga att vänta mer än tre sekunder innan dom lämnar sidan.

2.1 JavaScript

Enligt Richards et al. (2010) är JavaScript är ett objekt orienterat programmeringsspråk med en liknande syntax som Java. Men tillskillnad ifrån Javans objektorienterande tänk så byggs JavaScript på en prototyp baserad objekt system. Vilket enligt Richards et al. (2010) gör JavaScripts objekt system till väldigt flexibelt som resulterar i att det ger en svårighets grad i att få en bra restriktion på beteendet av varje givet objekt. Richards et al. (2010) ger en exemplifiering på JavaScript där dom beskriver hur ett innehåll i en egenskap kan modifieras vid varje given tidpunkt eller byta ut en egendoms fält helt. I JavaScript kan vilken funktion som helst bli konstruerad för en ”klass” objekt och innehålla en prototyps fält vilket initialt refererar till ett tomt objekt.

2.2 Node.js

Enligt Ojamaa och Düüna (2012) är Node.js en öppen source plattform för nätverks applikationer som siktar in sig på att förenkla implementationen för snabba och skalbara nätverks tjänster, så som t.ex. webbapplikationer. Node.js är enligt Ojamaa och Düüna (2012) en event baserad nätverks applikation som gör att utvecklare använder sig av en asynkron programmerings interface för I/O operationer. Där programmeringsspråket i utvecklingsmiljön är JavaScript. Node.js använder sig av en tråd för att exekvera sin applikations kod. Vilket resulterar i att all kod och förfrågningar som kommer in hanteras av en och samma tråd enligt Ojamaa och Düüna (2012). Node.js är implementerat i C++ och är baserat på Google V8 JavaScript motor som var utvecklat för Google Chrome webbläsare. Enligt Ojamaa och Düüna (2012) så är V8 en snabb motor som har ett bra minnes hantering och hanterar kör tiden på event tråden.

2.3 Databaser

Databaser är en förvaringsplats med organiserad och strukturerad data enligt Abramova och Bernardino (2013). För att få tillgång till data är det vanligt enligt Abramova och Bernardino (2013) att använda sig av ett ”DataBase Management System” även förkortat till DBMS. DBMS definierar Abramova och Bernardino (2013) som en samling av mekanismer som möjliggör lagring, editering och hämtning av data. Dom beskriver också att DBMS har under de senaste åren blivit en synonym med databaser.

För att kunna hantera data på ett enkelt sätt utan att få omvälvande förändringar i representationen av data föreslog Codd (1970) att data ska bli representerad i en relationsmodell. Där hans förslag lämnas öppen för vilket språk som relationerna ska skrivas i.
Där utifrån Codds (1970) teori utvecklades enligt Abramova och Bernardino (2013) språket ”Structured Query Language” förkortat till SQL. Där SQL har blivit en standard för att hantera data i relationsdatabaser även kallat ”Relational DataBase Management System” förkortat till RDBMS. Där relationerna representeras som uppsättningar av tabeller. Där all data är relaterat och kan bli tillträdes samtidigt. Där komplexa databaser innehåller många tabeller och för att kunna få fram relationer emellan tabellerna använder sig relationsdatabaser sig av två olika nycklar. En primär nyckel som alltid är unik för att identifierar en tabell. Samt en främmande nyckel som används för att korsreferera mellan tabeller. Främmande nycklar i en tabell kan också vara en del av en primär nyckel i en annan tabell. Men en primärnyckel behöver inte vara unik. Det är bara när den är ihop parad med en primärnyckel och bildar en primärnyckel den måste vara unik.

Under dom senaste åren har trafiken på webbapplikationer blivit större och utvecklingen av nya system som kan hantera en god skalbarhet för enkla läs/skriva databas operationer över många servrar. Där dom nya databas systemen för att hantera dessa krav är så kallade ”Not Only SQL” förkortat till NoSql enligt Catell (2010). Där NoSql databaser till skillnad ifrån RDBMS använder sig inte av relationer. Enligt Catell (2010) så har NoSql generellt sex stycken funktioner.

1. Förmågan att skalas horisontellt med användning av ”enkla operationer” över många servrar med genomströmning.
2. Förmågan att replikera och distribuera data över många servrar
3. En enkel nivå på gränssnitt och protokoll (tillskillnad ifrån en SQl bindning).

4. En svagare samverkande modell än ACID överföring av dom flesta RDBMS.
5. En effektiv användning av distribuerade index och ram för data lagring.
6. Förmågan att dynamiskt lägga till nya attribut till data uppgifter.

Där Catell (2010) beskriver olika data lagrings kategorier för NoSql modeller, Lagring med Nyckelvärden, Dokument lagring och utdragbar register lagring även kallat ”kolumn lagring”.

Nyckelvärde – All data är lagrad med en nyckel som index för att hitta data. Men detta sätt att lagra data erbjuder inget sekundärt index eller nycklar som en relationsdatabas.

Dokument – Data lagras i dokument där dokumenten är indexerade och en enkel frågeställning mekanism finns att tillförfogande för att få ut data. Tillskillnad ifrån nyckelvärden så kan dokument sättet lagra lite mer komplex data. Där dokument generellt kan stödja sekundär indexering och flertal olika dokument för databaser. Där dokumenten kan vara nästlade dokument och listor.

Utdragbar registrering (kolumn lagring) – Lagrar utdragbara uppgifter som kan bli partitionerade både vertikalt och horisontellt över noder. Där data modellen är rader och kolumner där skalbarhets modeller är delar upp både rader och kolumner över flertal noder.

Rader bli delade över noder med en primär nyckel.

Kolumner av tabeller är distribuerade över flertal noder och använder sig av ”grupper av kolumner”.

Flesta kolumn lagringen är inspirerat av Goggles BigTable. Dom är alla väldigt lika men skiljer sig lite i ”concurrenct mechanism” och andra funktioner.

2.4 Tidigare forskning

2.4.1 Chaniotis, Kyriakou och Nikolaos (2015)

Är end-to-end JavaScript ett hållbart alternativ för att bygga moderna webbapplikationer? Enligt dom testarna som Chaniotis, Kyriakou och Nikolaos (2015) har gjort så är det så och också högt rekommenderat. I deras experiment testar dom om Node.js är bättre än PHP/Nginx och PHP/Apache. Vilket dom drar slutsatsen att PHP/Apache är uråldrigt och inte är lika effektivt att använda sig av servens minne samt CPU eller lika skalbart som dom andra två. Enligt Chaniotis, Kyriakou och Nikolaos (2015) så presterar Nginx bättre än Apache i alla tester som dom gör. Det enda användningsområdet som Nginx och Apache presterar bättre än Node.js i deras tester är att skicka statiska filer mellan server och klienter. I alla dom andra testerna så presterar enligt Chaniotis, Kyriakou och Nikolaos (2015) Node.js bäst i att vara mest effektiv i att använda minnet samt att utnyttja den processkraft som fanns tillgängligt.

Utifrån Chaniotis, Kyriakou och Nikolaos (2015) tester så rekommenderar dom att Nginx är mycket bättre som en statisk fil server än Node.js. Men Node.js har mycket bättre datorprestanda än PHP vilket gör att den utnyttjar hårdvarans processkraft och minneseffektivitet så skulle Node.js vara mycket mer lämpad till dynamiska

webbapplikationer. Men eftersom Nginx var mycket mer effektiv i att skicka statiska filer rekommenderar Chaniotis, Kyriakou och Nikolaos (2015) att en mix mellan Node.js och Nginx används för att optimera en webbapplikation med en webbserver bakom. Där en kombination med Nginx server framför som tar hand om statiska filer och sedan låta Node.js ta hand om all dynamiskt innehåll på en webbapplikation. Tillslut så skulle denna kombination kunna byta ut åldrandes kombination av PHP/Apache.

Problemet med Chaniotis, Kyriakou och Nikolaos (2015) experiment och rekommendation är att dom utvärderar bara serversidan av en applikation och inte tar med en verkligare webbapplikation som används i ett test. Dom testar statiska filer på 13.5 mb som representerar en webbapplikation samt en webbapplikation som skriver ut ”hello world”. Detta problem tar dom även upp i sin slutsats att det borde varit bra att göra en fallstudie på ett par riktiga webbapplikationer som använder sig av dessa olika kombinationer av tekniker. Som också använder sig av data ifrån databaser som hämtar data dynamiskt in till en webbapplikation. För att kunna få fram mer data om vad som är bäst i praktiken.

3 Problemformulering

Chaniotis, Kyriakou och Nikolaos (2015) drar en slutsats Nginx tydligt presterar bättre än Apache och samtidigt är mer effektivt än Node.js server del. Däremot så säger dom att det är tydligt att Node.js överträffar PHP i datorprestanda genom att vara mer minne effektiv och utnyttja all tillgänglig processkraft.

Problemet med deras studie är att deras fokus ligger på hur serversidan av Apache, Nginx och Node.js presterar gentemot varandra. Där testerna inte har en riktig front-end utan bara en statisk fil som skickas i representation av en hemsida. Dom har heller inte en databas där data hämtas med hjälp av JavaScript eller PHP. Vilket Chaniotis, Kyriakou och Nikolaos (2015) också tar upp i sin studie. Dom nämner också att det hade varit intressant att göra en fallstudie på verkliga system som har alla dessa variabler.

För en webbutvecklare kan det vara svårt att göra valet med vilken databas man ska använda sig av. Ska man ta en RDBMS som är den vanligaste eller den nyare tekniken som är på uppgång med NoSql databas.

Problemet med val av en RDBMS är att dom är långsammare enligt Yishan och Manoharan (2013) mätningar jämfört med NoSql databaser. Vilket Cattel (2010) också stödjer.

Då kommer problemet om man som webbutvecklare förlorar tillräckligt med svarstid till klienten när man använder sig av en RDBMS som är dom vanligaste databaserna som flest utvecklare känner till och är bekväma att använda sig av. Eller om det är så stor skillnad i svarstiderna att man behöver använda sig av en NoSql databas. Detta är i det fallet då enbart enklare frågeoperationer ställs mot databaser som NoSql databaser vilket också RDBMS databaser klarar av samt mycket svårare frågeoperationer.

Detta lämnar ett antal frågor som en webbutvecklare som utvecklar kompletta webbapplikationer med en webbserver, back-end med databas PHP/Node.js och en front-end. Där svarstider för användarna kan avgöra om dom vill använda sig av utvecklarens webbapplikation.
Frågor som ska besvaras:

- Hur påverkas svarstiden när applikationen hämtar ett JSONObjekt ifrån en databas för att skriva ut det till klientsidan av applikationen.
- Ger det någon markant skillnad om applikationen använder sig av PHP eller JavaScript mellan databas och front-end.
- Ger det någon markant skillnad om det är en NoSql databas eller en RDBMS som lagrar JSONObjekten.
- Ger det någon markant skillnad när webbservern är Nginx för PHP eller när webbservern är Node.js för JavaScript.

I frågeställningen utlämnas sådana variabler som kan påverka svarstider så som CPU användning och minnes hantering. Eftersom Chaniotis, Kyriakou och Nikolaos (2015) redan har ett väl genomfört experiment på dessa variabler som kan påverka en webbapplikation. Samt att hårdvara i dagens läge är mycket billigare att köpa in än en utvecklare med kompetens i olika utvecklingstekniker.

3.1 Hypotes

Hypotesen är att JavaScript kommer att ge lägre svarstider än PHP när ett JSONObjekt skickas till klient och det kommer inte göra någon skillnad om det används en NoSql databas eller en RDBMS.

3.2 Metodbeskrivning

När man utvärderar olika tekniker inom mjukvaruutveckling så använder man sig enligt Wohlin et al. (2012) utav tre olika tekniker. Dessa är undersökning t.ex. enkät som samlar in data ifrån eller om människor, fallstudie som utvärderar data i en redan implementerad metod eller verktyg i en industri och experiment som används i en kontrollerad miljö där en faktor eller variabel manipuleras av den studerade inställningen.

Den metoden som kommer att användas för att kunna besvara på hypotesen och dom frågeställningarna som tagits upp är experiment. Det lämpar sig bäst eftersom både Chaniotis, Kyriakou och Nikolaos (2015) samt Yishan och Sathiamoothy (2013) använde sig av experiment när dom gjorde sina tester. Styrkan med experiment beskriver Wohlin et al. (2012) är att det går att undersöka i vilka situationer som hypotesen eller när frågeställningarna är sanna. Samt att ett experiment används när man vill ha kontroll över en situation och vill manipulera beteendet direkt, snabbt och systematiskt. Ett experiment kan ge kontext när en standard, metod eller verktyg kan rekommenderas för användning. Vilket passar in för den hypotesen och frågeställningarna som finns i denna avhandling vill ge i slutändan en rekommendation vilket språk eller databas som kan tänkas användas i webbapplikations skapande.

Experimentet som ska utföras är inspirerat av Chaniotis, Kyriakou och Nikolaos (2015) där dom mäter ifrån en http förfrågning är förfrågad emot serven tills att en fil har blivit överförd till klienten. Men i detta arbete så ska filen representeras av en webbapplikation som ska laddas klar för användaren. Där istället för att mäta prestandan på serven som Chaniotis, Kyriakou och Nikolaos (2015) gör så är det i detta arbete fokus på svarstiderna för klienten som blir det slutgiltiga data som utvärderas. Då svarstiderna mäts ifrån när en användare gör en förfrågning till serven fram tills att den informationen som hämtas ifrån databasen har skrivits ut med den JavaScript, HTML och CSS för att visa upp webbapplikationen. Detta kommer att ge en total svarstid för en klient sidan.

För detta experiment så finns det delar av den totala svarstiden som är intressanta att ta ut mer mätpunkter för att få ut data om vilken teknik som påverkar svarstiden. Eftersom experimentet inte bara ska kolla på totala svarstiden utan även också skillnaden mellan RDBMS/NoSql och JavaScript/PHP.

Den delen av experimentet som gäller för databas delen så kommer svarstiden till de olika databaserna mätas ifrån att webbapplikationen kallar på databasen det vill säga ifrån att JavaScript/PHP koden gör en förfrågan till databasen tills det att JavaScript/PHP får tillbaka data ifrån databasen. Där detta experiment är inspirerat av Yishan och Sathiamoothy (2013) som när dom gjorde sina experiment på databaser använde sig av fem olika operationer för att få fram tider om hur olika operationer fungerade på RDBMS och NoSql databaser. Då att utvärdera alla fem operationer dom gjorde så kommer i detta experiment använda sig av dom två vanligaste som används vid användandet av webbapplikationer. Då Catell (2010) gör ett liknande experiment för utvärdering av RDBMS och NoSql databaser

använder sig av två enkla operationer för att utvärdera databaser vilket är operationerna läs/skriv. Där läs/skriv används också två utav dom fem som Yishan och Sathiamoothy (2013) använder kommer detta experiment använda sig utav den vanligaste av dessa två operationerna som är läs.

Sista delen av den totala svarstiden som är intressant att ta fram data på för att få en utvärdering på PHP/JavaScript är ifrån att PHP koden börja köras tills den är slut exekverat. För att få fram hur lång tid som PHP koden tar upp av den totalasvarstiden. När det gäller Node.js som är motsvarande så är det den kod som körs i experimentet istället för PHP koden. D.v.s. den kod biten/filen som körs istället för PHP.

I detta experiment så är det ännu en aspekt av webbapplikationens samarbete med webbservern som är ska utvärderas. Även detta är inspirerat av Chaniotis, Kyriakou och Nikolaos (2015) experiment för webbservrar. Där det experimentet ska samla in data för hur många samtidiga anslutningar det kan vara innan den totala svarstiden för användaren blir förlångsamt. D.v.s. att den totala svarstiden ska vara under tre sekunder som enligt Mickens (2010) är då en stor andel av användarna lämnar webbapplikationen.

3.2.1 Alternativa metoder

Alternativa metoder som finns till experiment enligt Wohlin et al. (2012) är enkät och fallstudie.

Där en enkät är en samling av och om människors beskrivning, jämförelse, attityd och beteende kring en mjukvara/teknik. Där en enkät oftast används för att undersöka i efterhand när en teknik eller verktyg har varit i användning under ett tag. Den primära insamlingen av data i en enkät är intervjuer eller frågeformulär. Där resultatet kan bli analyserat till en härledande beskrivning och förklarande slutsats (Wohlin, et al. 2012).

En fallstudie enligt Wohlin et al. (2012) genomförs för att undersöka en entitet eller ett fenomen inom ett verkligt sammanhang inom en specifik tidsram. Där fallstudie passar sig bäst för industriella utvärderingar av metoder och verktyg. Där skillnaden ifrån ett experiment är att experiment samlar in data för variabler som blir manipulerade. En fallstudie väljer ifrån dom variabler som representeras i en typisk situation. Enligt Wohlin et al. (2012) finns det nackdelar och fördelar med fallstudier. Fördelen är att den är enklare att planera och mer realistisk. Nackdelen är att resultaten är mer svåra att generalisera och svårare att tolka. D.v.s. att det går att visa en effekt på en typisk situation men den kräver mer analysering för att generalisera till andra situationer.

Utifrån ovanstående så går det att göra en enkät för att se vad en användare tycker om svarstider samt när en användare inte längre tycker att en total svarstid är förlångsam för att få fram en hållbar webbapplikation. Som Mickens (2010) redan har testat med en enkät för användare för att få fram acceptabla svarstider. Så skulle en del av studien kunna vara en enkät. Dock blir det svårt att få tag i tillräckligt med många användare för att få ett generellt statistiskt resultat för att få fram en bra slutsats. Samt att det är logistiskt krävande att få in dom i en labb sal där experimentet äger rum.

Det skulle varit lämpat att göra en fallstudie av på realistiska implementationer som redan använder sig av dessa olika verktyg som kommer användas i experimentet. Som även Chaniotis, Kyriakou och Nikolaos (2015) har med i sin slutsats att för framtida forskning kan det vara bra att göra en fallstudie på realistiska implementationer av dessa tekniker. Men

eftersom flera variabler ska undersökas i en kontrollerad miljö så är experiment att föredra i detta fall jämfört med fallstudie.

3.2.2 Forskningsetiska aspekter

Dom etiska problem som kan uppstå är om det data set som används i exkrementet kan spåras till någon individ eller vara stötande för den som läser denna avhandling. Det är också en etnisk fråga om vem data som produceras gynnar. Något företag som pushar för den gamla tekniken eller någon som tar fördel för att Node.js visar sig vara den mer optimala tekniken. Kommer experimentet ge en sanningsenlig bild när man använder en stor webbapplikation kommer denna applikation spegla verklighetens applikationer vad det rätt val av metod eller skulle det göras en fallstudie.

4 Implementation och datainsamling

Under detta kapitel implementerades dom grundläggande delarna som är tvungna att finnas i webbapplikationen för att svara på dom frågeställningarna och hypotesen som är ställda i denna studie. De grundläggande delarna som implementerades för att få fram en webbapplikation som kan svara på hypotesen och frågeställningarna är:

- Webbserver med Node.js samt en motsvarande med Nginx.
- En NoSql databas som lagrar JSON objekt i denna studie används MongoDB.
- En RDBMS som lagrar JSON objekt i denna studie används MySql.
- Ett dataset som finns lagrat i JSON struktur som går att importera till de olika databaserna.
- Ett PHP script som hämtar ut JSON objektet ifrån databaserna samt en motsvarande fil i JavaScript som kan hämta ut liknande JSON objekt.
- En front-end med HTML fil som parsar ut JSON objekten till på ett läsbart sätt i en tabell.
- En JavaScript fil som kallar gör ett anrop till PHP/JavaScript filen som hämtar JSON objektet ifrån databasen. Där enda skillnaden i filen är just att den kallar på en PHP fil eller en JavaScript fil.
- Applikationen ska lagra de svarstider som behövs för att svara på hypotesen det vill säga, en total tid, en tid för PHP/JavaScript som hämtar JSON objektet tar. Hur lång tid anropet till databaserna tar samt vilken totaltiden blir.

4.1 Förstudie

För att få en bra förståelse hur en utvecklare skriver en dynamisk applikation så är Nixon(2009) en väldigt bra källa för att få en bra grund. Han beskriver också i sin bok hur grunderna för JavaScript, PHP och MySql ska implementeras. Vilket är väldigt ger en bra grund för att kunna förstå hur dom olika applikationerna har byggts i denna studie. Vilket passar in då alla dessa tre språken är med i denna studie. Att studera denna bok ger en bra grund, förståelse och uppfattning om det som undersöks i denna studie.

En inspirationskälla för att få så bra optimering på den kod som är skriven i JavaScript när data skall skrivas ut till HTML för användaren var Powel(2009). Där han beskriver och ger mätningar på vilket sätt som är mest effektivt enligt hans mätningar för att skriva ut stora arrayer i JavaScript. För att mätningarna ska bli så rättvisa som möjligt är det viktigt att den kod som används är så optimerade som möjligt. Därför var hans blogg inlägg en bra grund för att kunna ge JavaScript en sån bra optimering som möjligt.

4.2 Implementering grunden

4.2.1 Dataset

Datasetet är hämtat från Usgs(2016) och är ett stort dataset som visar på koordinater till mineraler depåer i världen. Där även information till vad det är för mineraler och massor av data på dessa depåer finns i datasetet. Valet varför just detta datasetet används för att mäta på JSON objekt genom PHP/JavaScript är för att det är väldigt stor data att jobba med. Det

gör det enklare att få fram JSON objekt som är stora nog att jobba med och det behövs inte tillverka eller hitta på ny data för att mäta på tillräckligt stora JSON objekt. Om det visar sig att JSON objekten blir för stora så är det enklare att skala av JSON objektet.

```
1  dep_id,mrds_id,mas_id,site_name,latitude,longitude,region,country,state,county,com_type,commod1,commod2,commod3,oper_type,dep_type,prod_size,
   dev_stat,ore,gangue,other_mat1,orebody_fm,work_type,model,alteration,conc_proc,names,ore_ctrl,reporter,hrock_unit,hrock_type,arock_unit,arock
   _type,structure,tectonic,ref,yfp_ba,yr_fst_prd,ylp_ba,yr_lst_prd,dy_ba,disc_yr,prod_yrs,discr
2  10000001,A010000,,"Lookout Prospect",55.05612,-132.14344,NA,"United States",Alaska,,M,Copper,"Silver,
   Gold",,Unknown,,N,Occurrence,"Chalcopyrite, Covellite, Pyrite","Quartz, Sericite",,,,,,,"Wakefield Minerals Co., Mammoth,
   Conundrum",,"Hirschmann, M. M. (Elliott, R. L.)",,Schist,,,"Schist Strikes N65w, Dips 70sw",,"USGS PROFESSIONAL PAPER 1, P. 75-77.USGS BULL
   347, P. 131.USGS BULL 1246, P. 174USGS MF 433USGS OF 78-869, P. 117",,,,,,,,
3  10000002,A010001,,"Lucky Find Prospect",55.52751,-132.68514,NA,"United States",Alaska,,M,Copper,Gold,,Unknown,,N,Occurrence,"Chalcopyrite,
   Pyrite","Calcite, Quartz, Siderite",,,Underground,,,,,"Vein Follows Contact","Hirschmann, M. M. (Elliott, R. L.), Mosier,
   Dan",,Diabase,,,,,"USGS BULL 347, P. 165.USGS MF 433USGS OF 78-869, P. 120.",,,,,,,,
```

Figur 1 Visar utdrag ifrån CSV fil med datasetet

Som visas på figur 1 så visas en bit av datasetet ifrån den CSV fil som hämtats ifrån Usgs(2016). Där rad ett är alla nycklar som finns och rad två samt tre visar data till två stycken mineral platser. Något att notera är att det finns många ”,,” som visar på att det finns en del kolumner utan data som är ifyllt.

4.2.2 JSON objekt

Eftersom det som ska utvärderas är ett JSON objekt som skickas och att det webbapplikationer kommer att använda sig av MongoDB samt MySql som båda lagrar JSON objekt i sig. Så när datasetet som ligger i CSV format importeras in i MongoDB sparas data undan i JSON objekt i dokument struktur. Samt när CVS filen importeras till MySql så kommer data också sparas undan i JSON objekt.

```
25 {
26     "_id" : "56f2bfbe8154d3ef8a0d8a44",
27     "dep_id" : 10000001, "mrds_id" : "A010000",
28     "mas_id" : "", "latitude" : 55.05612,
29     "longitude" : -132.14344, "region" : "NA",
30     "country" : "United States",
31     "state" : "Alaska",
32     "com_type" : "M",
33     "commod1" : "Copper",
34     "commod2" : "Silver, Gold",
35     "prod_size" : "N",
36     "dev_stat" : "Occurrence",
37     "ore" : "Chalcopyrite, Covellite, Pyrite",
38     "gangue" : "Quartz, Sericite",
39     "work_type" : "",
40     "hrock_type" : "Schist",
41     "structure" : "Schist Strikes N65w, Dips 70sw",
42     "ref" : "USGS PROFESSIONAL PAPER 1, P. 75-77.USGS BULL 347, P. 131.USGS BULL 1246, P. 174USGS MF 433USGS OF 78-869, P. 117"
43 }
```

Figur 2 JSON exempel som används i webbapplikationen

I figur 2 så visas data som sparats undan i ett JSON objekt. I bilden visas ett objekt men i själva databasen finns hundratusentals sådana objekt som kommer att skickas fram och tillbaka för att få utvärdering av webbapplikationen. Till skillnad ifrån figur 1 så finns det mindre nycklar i JSON objektet som har blivit bortskalat för att inte skicka nycklar med tomma värden fram i JSON objektet.

4.2.3 HTML med JavaScript

För att enbart påverka webbapplikationerna på så lite som möjligt i mätningarna som ska göras kommer dom den HTML kod som JSON objekten skrivas ut se exakt lika dana ut det vill säga, dom delar samma fil. Detta gäller också den JavaScript filen som anropar PHP/JavaScript filen som hämtar JSON objektet ifrån databasen som är vald. För kod på

JavaScript filen se Appendix B - global.js. Där den enda som skiljer är vilken fil som JSON objektet hämtas.

_id	dep_id	mrds_id	mas_id	latitude	longitude	region	country	state	com_type	commod1	commod2	commod3
56f2bfec8154d3ef8a107d86		NM STATE HIGHWAY DEPT INDEX		New Mexico	10199000		350190032	NA	Sand and Gravel, Construction			United States
56f2bfec8154d3ef8a107d87		USGS TOPO MAP 1978		New Mexico	10199001		350410057	NA	Sand and Gravel, Construction			United States
56f2bfec8154d3ef8a107d88		RECORDS OF N.M. STATE HWY. DEPT		New Mexico	10199002		350390439	NA	Sand and Gravel, Construction			United States
56f2bfec8154d3ef8a107d89		WILLIAMS F E ETAL 1964 NMBM CIRC 76P31CLAIMS EXTEND INTO SEC 7 NW QUARTER		New Mexico	10199004		350530206	NA	Fluorine-Fluorite, Quartz			United States

Figur 3 Klient vyns table

När den gemensamma HTML JavaScript filerna körs så visas exakt lika HTML table för klienten som visas i figur 3. Där koden till HTML filen finns att se i Appendix A - Index.html.

4.2.4 PHP/JavaScript fil

För PHP och JavaScript filerna som hämtar JSON objektet så ska dom ha exakt samma funktion eller så snarlikt som möjligt. Där det enda som skiljer sig åt är programmeringsspråket men själva funktionen av koden ska vara så snarlik som möjligt. Den ska göra en anslutning till databasen få ut JSON objektet och skicka fram det objektet till front-end JavaScripts fil. För att sedan kunna skrivas ut till användaren.

```php
<?php
    $start_time = microtime(true);

    $m = new MongoClient();

    $db = $m->nodetest1;

    $data = array('infojson' => array(), 'timeer' => array());

    $collection = $db->coords;

    $cursor = $collection->find()->limit(10000);
    $end_time = microtime(true);
    foreach ($cursor as $document => $row){
        array_push($data['infojson'], $row);

    }
    $total_time = ($end_time - $start_time);

    array_push($data['timeer'], $total_time);
    print_r(json_encode($data));
```

Figur 4 Exempel PHP fil för att hämta JSON objekt ifrån databasen.

I figur 4 så visas den kod som hämtar ut ett JSON objekt med data ifrån databasen för att kunna skrivas ut till användaren. Den funktionen filen har är att den öppnar en anslutning till databasen, hämtar ut det JSON objekt som frågas efter till databasen och sedan gör det tillgängligt för den JavaScripts fil som kallar på PHP filen att använda sig av JSON objektet.

```
14  router.get('/getdata', function(req,res){
15      var db=req.db;
16      var collection = db.get('coords');
17
18      var end = null;
19      var dbConnectTime = null;
20      var start = new Date().getTime();
21      var data = {
22          infojson:[],
23          timeer:[]
24      };
25
26      collection.find({},{limit:1000},function(e,docs){
27
28          data.infojson = docs;
29
30          end = new Date().getTime();
31          dbConnectTime = (end - start)/1000;
32          data.timeer = dbConnectTime;
33          res.send(JSON.stringify(data,null,3));
34
35      });
```

Figur 5 Exempel på Node.js fil för att hämta JSON objekt ifrån databas.

I figur 5 visas en router fil i Node.js som hämtar ut ett JSON objekt ifrån en databas för att front-end JavaScripts fil ska kunna skriva ut data till användaren. Den funktionen filen har är att den öppnar en anslutning till databasen, hämtar ut det JSON objekt som frågas efter till databasen och sedan gör JSON objektet tillgängligt för den JavaScripts fil som kallar på denna router fil för att kunna använda sig av JSON objektet.

4.2.5 Svarstider

För att kunna få ut data om hur webbapplikationerna presterar emot varandra eller emot sig själva behövs tidtagning om hur lång tid det tar för olika kod bitar samt hela koden tar.

I JavaScript så kommer funktionen *performence.now()* att användas som returnerar millisekunder, exakt till en tusendels millisekunder. Vilket ger en mätning med hög precision.

I PHP så kommer funktionen *microtime()* att användas för att mäta tidpunkter. Funktionen *microtime(true)* att användas som ger ett returvärd i mikrosekunder. Vilket resulterar i att det blir två olika tidformat på dom olika språken därav måste *microtime(true)* funktionen divideras med ettusen för att bli konverterat till millisekunder som JavaScript funktionen *performence.now()* returnerar.

```
93  var start_time = window.performance.now();
94
95  //Excecute the cod you want to time
96
97  var end_time = window.performance.now();
98  var total_time = end_time - start_time;
```

Figur 6 Exempel på tidmätning i JavaScript

I figur 6 visas ett exempel på hur det ser ut när kod mäts i JavaScript. Där *start_time* tar en tidpunkt med *window.performence.now()*. Därefter kan kod köras där kommentaren visas i figur 6. Tillslut definieras en till tidpunkt med *end_time* och för att få ut mellanskillnaden subtraheras *end_time* med *start_time* för att få en mellanskillnad på hur lång tid det tog att köra koden.

```
20  $start_time = microtime(true);
21
22  //excecute the kod you want to time
23
24  $end_time = microtime(true);
25  $total_time = $end_time - $start_time;
```

Figur 7 Exempel på tidmätning i PHP

I figur 7 så visas ett exempel på hur det sur ut när kod mäts i PHP. Där *start_time* definierar en tidpunkt med *microtime()*. Sen kan kod köras där kommentaren visas i figur 7 som tillexempel den koden som visas i figur 5. Tillslut definieras en till tidpunkt med *end_time* och för att få ut mellanskillnaden subtraheras *end_time* med *start_time* för att få en mellan skillnad på hur lång tid det tog att köra koden.

4.3 Progression

När JavaScripts fil som skulle användas för att skriva ut JSON objektet i HTML så ifrån början så användes ett bibliotek som är byggt på JavaScript som heter JQUERY. Som gör det enklare att skriva just kod i JavaScripts filer. Då JQUERY laddar extra JavaScripts kod när den körs så får det konsekvenser när man ska iterera igenom JSON objekt. Vilket leder till att för varje iteration igenom JSON objektet så laddas det extra JavaScripts kod gentemot om koden bara är skriven i JavaScript.

```
51      $.each(data['infojson'],function(key,val){
52          var tableInputTd = "<tr>";
53          $.each(val,function(index,data){
54              tableInputTd += "<td>" + data + "</td>";
55          });
56          tableInputTd += "</tr>";
57          $('#resultdata').append(tableInputTd);
58      });
```

Figur 8 Utskrivning av JSON objekt in i HTML fil med JQUERY

```
25          var arr = data['infojson'];
26          var length = Object.keys(data['infojson']).length;
27
28          var values = data['infojson'];
29          var contentInsert = [];
30          var i = 1;
31          for (var a = 0; a < length; a++){
32              contentInsert[i++] = '<tr>';
33              for(var key in values[a]){
34                  contentInsert[i++] = '<td>';
35                  contentInsert[i++] = values[a][key];
36                  contentInsert[i++] = '</td>';
37              }
38              contentInsert[i++] = '</tr>';
39          }
40          $('#resultdata').append(contentInsert.join(''));
```

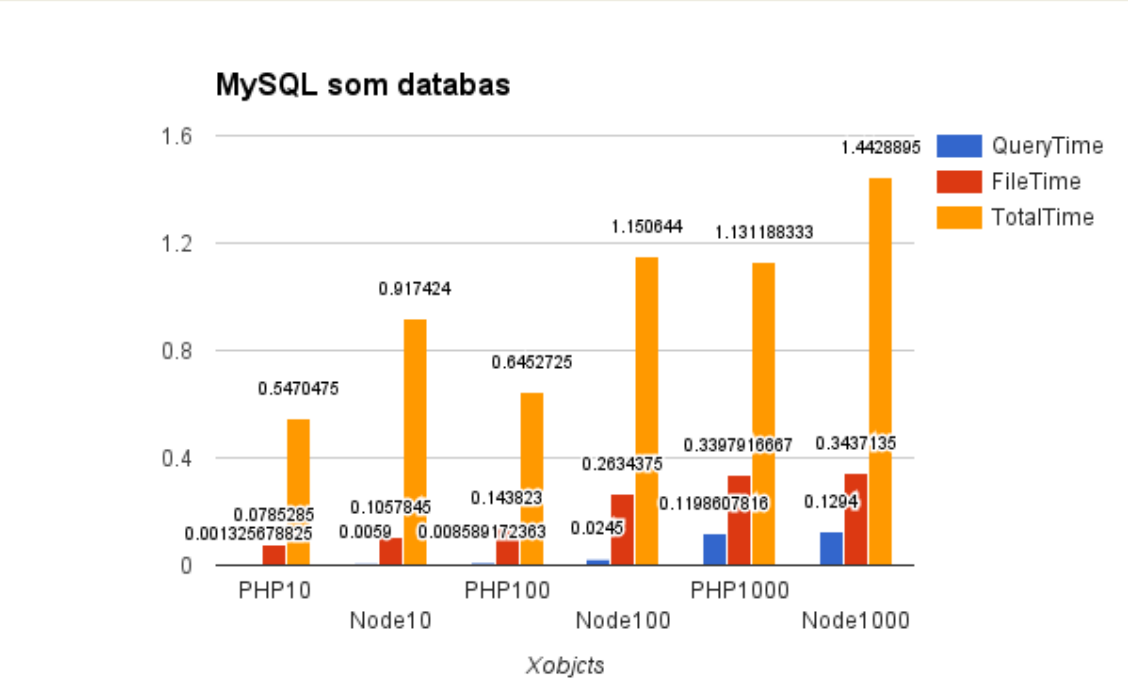
Figur 9 Utskrivning av JSON objekt in i HTML fil JavaScript

I figur 8 visas kod för hur JSON objektet skrivs ut till klienten vilket är skrivet med väldigt lite kod. I figur 9 så visas hur JSON objektet skrivs ut till klienten men i JavaScript. Som visas i dessa bilder så är det olika stora i mängden kod för att göra exakt samma funktion. Men eftersom JQUERY är ett biliotek så laddar den in mer JavaScript kod i bakgrunden utav funktionerna. Detta visar Powel(2009) i sin blogg inlägg vilket inspirerade till att använda så ren JavaScript som möjligt i detta experiment.

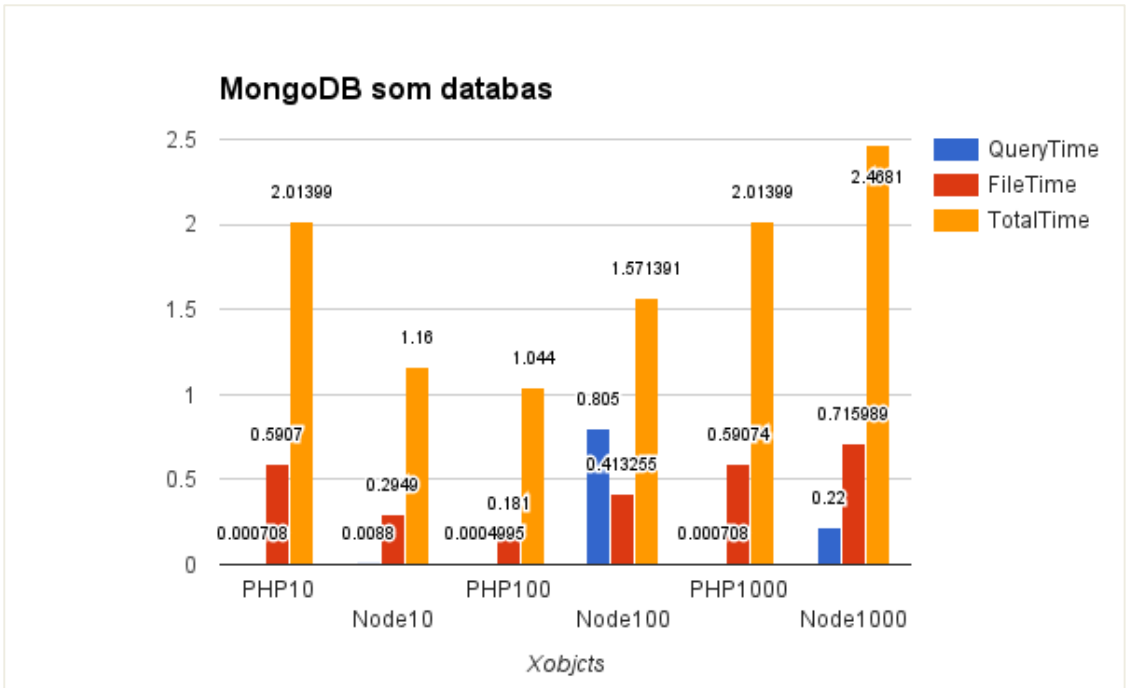
4.4 Pilotstudie

En pilotstudie gjordes för att få fram en baslinje för att mäta emot samt tre andra applikationer där olika variabler har bytts ut. Baslinjen i denna studie är en webbapplikation som använder sig av PHP för att skicka JSON objektet till klienten ifrån en MySql databas som lagrar JSON objekt. Den webbserver som används är Nginx till baslinje webbapplikationen. Dom andra webbapplikationerna som används är. En andra med Node.js som webbserver, JavaScript för att skicka applikationen ifrån en MySql databas. Den tredje varianten av webbapplikationen använder sig av Nginx, PHP samt en MongoDB databas istället för en MySql databas. Den fjärde varianten använder sig av Node.js, JavaScript och en MongoDB databas istället för en MySql databas.

I denna pilotstudie kördes alla fyra webbapplikationer tio gånger per mätpunkt och mätpunkterna så skickades tio, hundra och tusen JSON objekt igenom applikationerna.



Figur 10 Tio mätningar med MySql



Figur 11 Tio mätningar med MongoDB

I Figur 10 visas dom mätningar som gjordes för de webbapplikationerna som använde sig av MySql som en RDBMS. Samt i Figur 11 visas dom mätningar som gjordes för de webbapplikationerna som har en NoSql databas i form utav MongoDB. Där den mängd data som skickats i form utav JSONObjekt. Där JSONObjekten börjar med 1000, sen 5000 och till sist 10000. Där man också kan se i figurerna 10,11 att det sker en ökning beroende på mängd JSONObjekt så att mättningspunkterna inte går in i varandra.

Det som är noterbart med denna pilotstudie är att dom mätningarna som är genomförda har med all data som kom ut ifrån att applikationerna kördes. Det vill säga att spikar är med och det är inte tillräckligt många repetitioner för att få ut tillräckligt många iterationer av experimentet för att analysera. Det enda mätningarna visar är att det går att få ut data till experimentet. Det är även värt att visa på att visa iterationer av mätningarna som tex med tio JSON objekt får frågeställnings tid till databasen en väldigt liten tid. Där denna graf formatet inte kommer att användas till resultatet.

5 Utvärdering

5.1 Hårdvaruspecifikationer, versions specifikation för plattformar och test parametrar

I detta kapitel visas dom specifikationer på den dator som användes som server för att göra alla tester. Det visas också alla versioner av mjukvaror som användes under testerna. Samt att det presenteras vilka testparametrar som användes.

Modell	ASUS VivoBook S551LB 15.6" HD touch
Operativsystem	Microsoft Windows 10 x64-bit
Processor	Intel Core i5 4:e gen. 4200U / 1.6 GHz
Minne	8,00 GB DDR3 SDRAM 1600 MHz
Grafik	NVIDIA GeForce GT 740M - 2 GB DDR3 SDRAM
Lagring	750 GB HDD / 5400 rpm, 24 GB SSD-cache

Tabell 1 Hårdvaro specifikation för testdator

Mjukvara	**Version**
Node.js	5.0
Nginx	1.9.13
PHP	5.6.20
MySql	5.7
Google Chrome	51.0.2704.84
MongoDB	3.2

Tabell 2 Specifikationer för mjukvaran

I Tabell 1 visas den hårdvara som använts i detta experiment där det val som gjorts är en pc för att sätta upp miljön. Sen visas det även i Tabell 2 vilka versioner av mjukvara som använts till experimentet. Det är värt att notera att dessa verisioner inte är dom senaste. Utan är utvalda för att passa in till frågeställningarna och dom inspirationskällor som finns till studien.

Testfalls id	programmeringsspråk	Webbserver	Databas
T1	PHP	Nginx	MySql
T2	PHP	Nginx	MongoDB
T3	JavaScript	Node.js	MySql
T4	JavaScript	Node.js	MongoDB

Tabell 3 Kombinationer av programmeringsspråk, webbserver och databas för tester.

I Tabell 3 visas dom fyra olika kombinationer av databaser, webbservrar och programmeringsspråk som användes för testerna för att kunna svara på de frågeställningar som har ställts. Dom testfalls id som finns i denna tabell kommer att refereras till senare i studien då det är svårt att få plats med dom variabler som finns. Då dom bara är fyra stycken så är det enkelt att referera tillbaka.

Testfall	Test 1	Test 2	Test 3
T1	1000	5000	10000
T2	1000	5000	10000
T3	1000	5000	10000
T4	1000	5000	10000

Tabell 4 Olika testfall med hur många JSONObjekt som hämtades vid dom olika testerna.

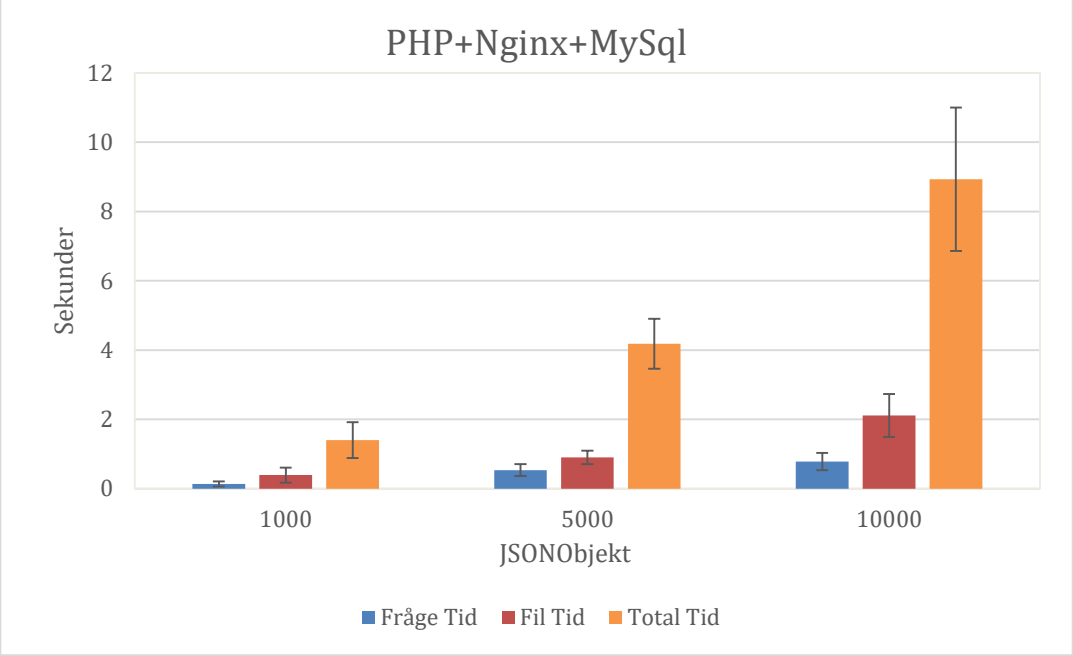
I Tabell 4 visas dom olika testfallen som gjordes. Med tre tester per testfall som upprepades ettusen gånger per test. Där för varje test så hämtades olika många JSONObjekt med en bas på ettusen JSONObjekt som stegrades till femtusen och till sist tiotusen.

5.2 Resultat

I detta kapitel visas det resultat för dom mätningarna som gjorts. Där det finns fyra olika kombinationer av webbapplikationer som mäts på. Där alla fyra utav dessa webbapplikationer har blivit testad på tre stycken olika punkter. Där varje punkt har olika mängder av JSONObjekt som skickats igenom webbapplikationerna. På varje mätpunkt har testerna gjorts ettusen repetitioner av.

5.2.1 T1 (Baslinje) PHP + Nginx + MySql

Här visas de resultat för mätningarna av baslinjen för testerna även kallat T1. Det resultat som visas är för dom mätningarna som gjordes med PHP som programmeringsspråk för att hämta och skicka fram JSONObjekten till klienten med Nginx som webbserver och MySql som lagrade JSONObjekten som databas. Det är även denna mätning som kommer att vara den bas som kommer att jämföras emot för att utvärdera hypotesen och frågeställningarna.

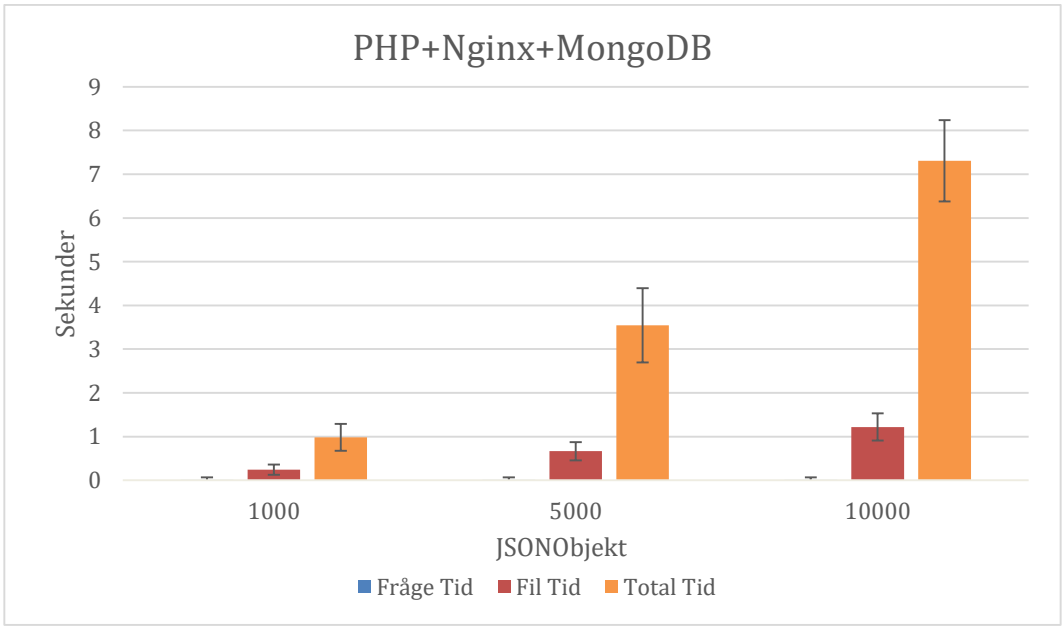


Figur 12 Visar medelvärden med standardavvikelser för T1.

I Figur 12 visas de medelvärden för mätningen av T1 (baslinjen) där man kan se att för varje mätpunkt så stiger de tre mätvärdena som mäts på. Där grafen börjar lågt för frågetiden, fil tiden och totaltiden sen stiger för varje stegring utav JSONObjekt.

5.2.2 T2 PHP + Nginx + MongoDB

Här visas det resultat för testerna för T2. Där PHP är programmeringsspråket som hämtar JSONObjekten till klienten ifrån en MongoDB databas och där hela webbapplikationen ligger på en Nginx webbserver.

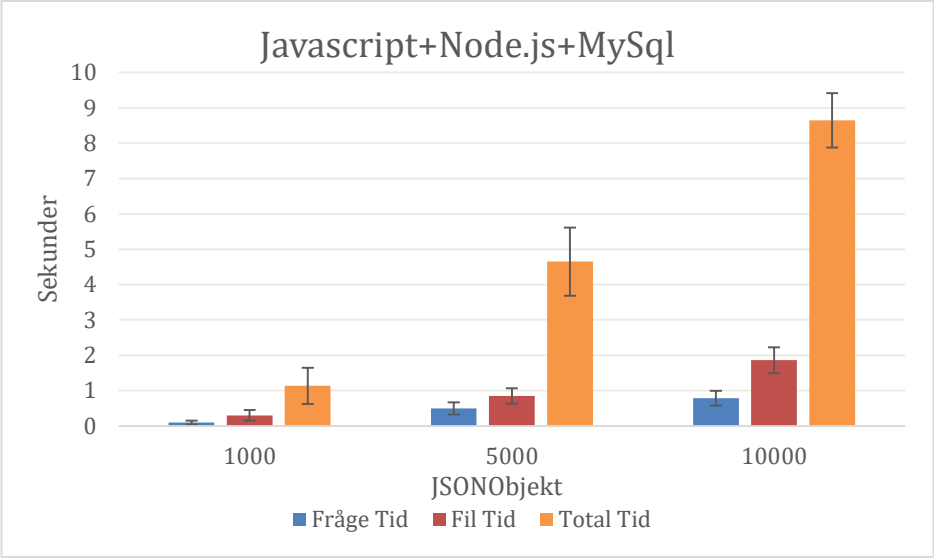


Figur 13 Visar medelvärden med standardavvikelser för T2.

I Figur 13 visas de medelvärdena för frågetid, fil tid och totaltid under mätningen av T2. Där man kan se att fil tiden och total tiden stiger ganska kontinuerligt när det skickas mer JSONObjekt. Vad som är värt att ta med sig till analysen är frågetiderna eftersom dom är så små jämfört med fil tiden samt om man jämför med fråge tiden ifrån T1(baslinjen) som visas i Figur 12.

5.2.3 T3 JavaScript + Node.js + MySql

Här visas resultatet för T3. Där JavaScript är det språk som används för att hämta JSONObjekt ifrån en MySql databas som ligger på en Node.js webbserver.

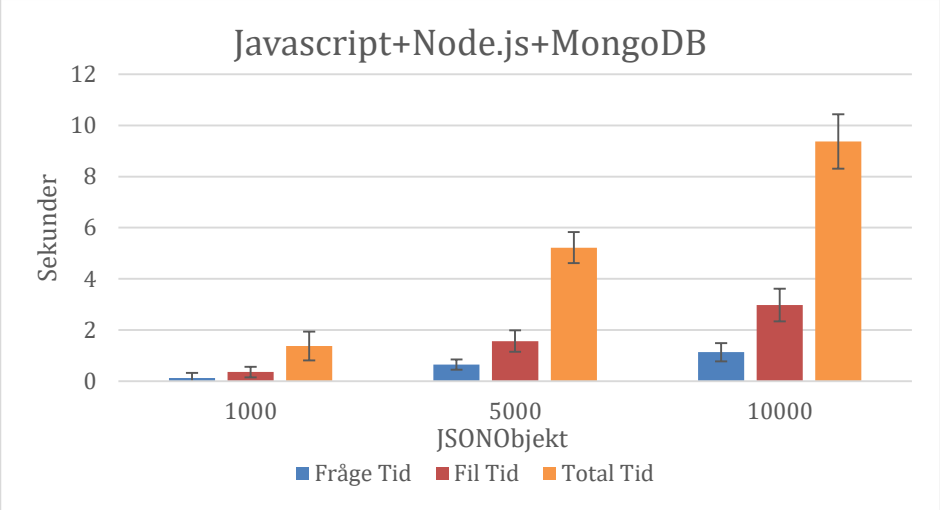


Figur 14 Visar medelvärden med standardavvikelser för T3.

I Figur 14 visas dom medelvärdena för hur långtid det tar för webbapplikationen med testfall T3 att köras. Där det visar på hur lång tid det tar för fråge tid, fil tid och den totala tiden för att visa webbapplikationen för klienten. Även i T3 testfallet ser det jämt ut med en stegring av tider för varje mätpunkt där JSONObjekten ökar.

5.2.4 T4 JavaScript + Node.js + MongoDB

Här visas resultatet för T4. Där JavaScript är det språk som används för att hämta JSONObjekt ifrån en MongoDB databas som ligger på en Node.js server.

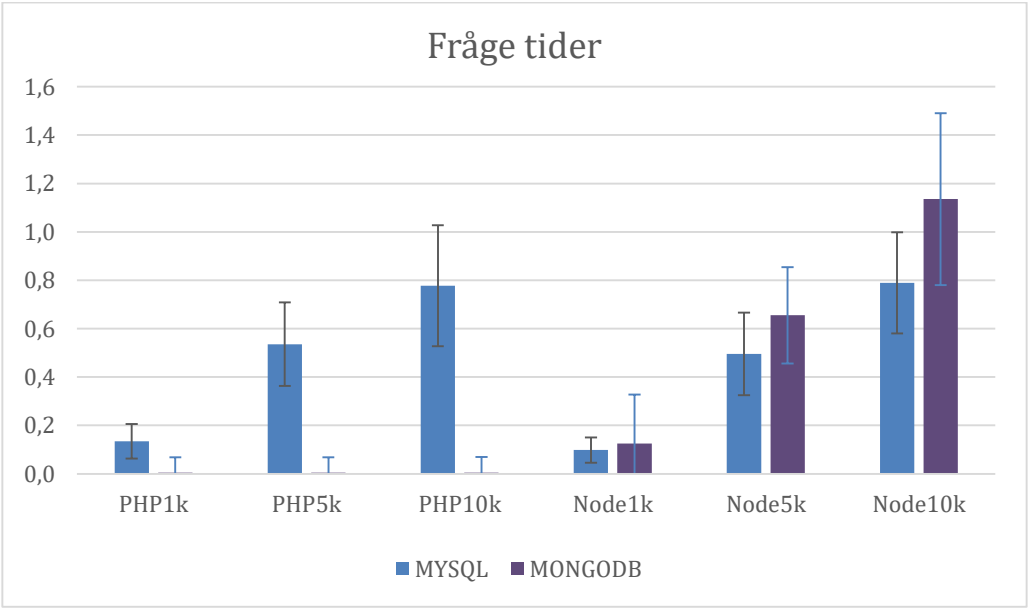


Figur 15 Visar medelvärden med standardavvikelser för T4

I Figur 15 visas dom medelvärdena för webbapplikationen med variablerna T4 att köra. Där det visas medelvärde på hur lång tid det tar att ställa en fråga ifrån databasen och få tillbaka JSONObjekten. Där det också visas hur lång tid JavaScript filen som hämtar JSONObjekten tar. Samt det medelvärdet på den totala svarstiden för klienten. Där även i detta testfall är en jämn stegring på medelvärdena när storleken på hur många JSONObjekt hämtas.

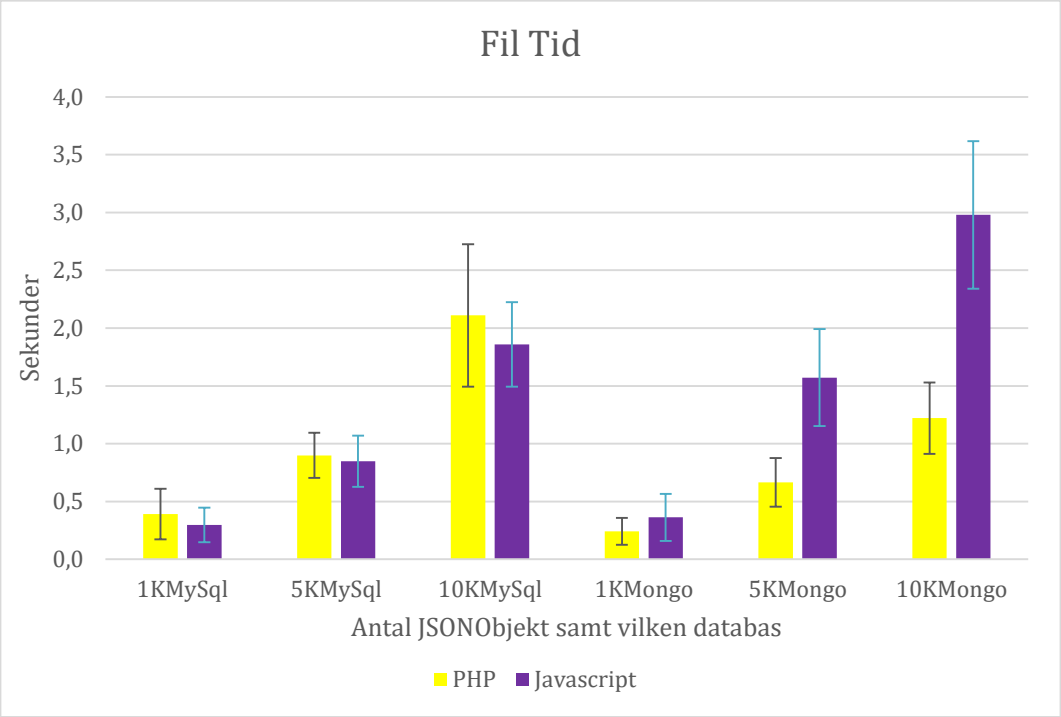
5.3 Analys

Det som kommer analyseras av resultatet är dom olika medelvärdena emot varandra. Där frågetiderna, totaltiderna och fil tiderna ställs upp i grafer emot varandra för att se på ett djupare sätt vad som har hänt i mätningarna.



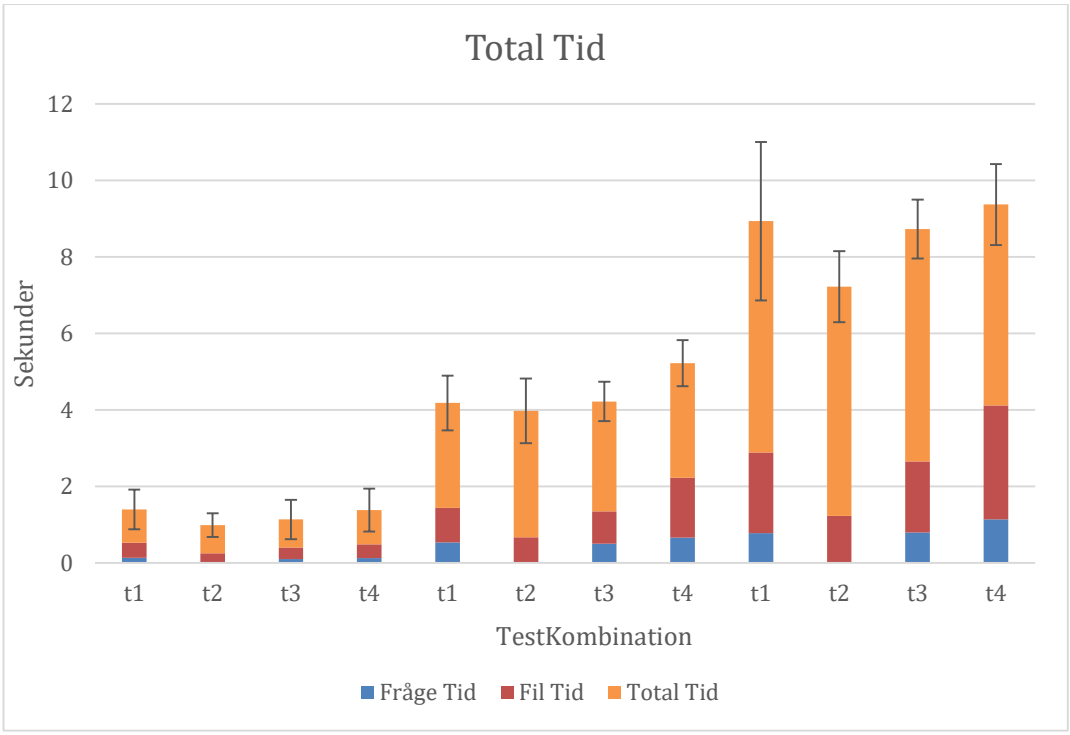
Figur 16 Medelvärden på alla frågetider som ställts med urskiljning på databas typ.

Det som är värt att notera när frågeställningarna körs emot dom olika databaserna som visas i Figur 16 är att när Node.js använder sig av JavaScript för att ställa frågan är det en skillnad som visas i staplarna men eftersom standardavvikelserna är parallella med varandra kommer ett framtida T-test med stor sannolikhet inte kunna ge ett statistiskt säkerställt resultat att dom skiljer sig åt. Sen visar Figur 16 när det är PHP som ställer frågeställningar mot MongoDB databasen så ser det ut på staplarna att det inte är någon stegring i hur lång tid det tar att ställa frågan även om frågeställningen frågar efter mer data. Men när PHP koden frågar emot en MySql databas så visar Figur 16 att staplarna stegrar efter hur mycket data som frågas efter.



Figur 17 Medelvärden med standardavvikelse för fil tiden urskiljning på programmeringsspråk

Det som kan urskiljas i Figur 17 är att på dom mätningarna som har MySql som databas så finns det ingen skillnad emellan språken då standardavvikelserna ligger parallellt med varandra. Det syns även på att medelvärdena ligger väldigt nära varandra. Men med dom som har MongoDB som databas går det att urskilja en större skillnad i medelvärdena där det med stor sannolikhet finns en statistik skillnad emellan språken då det skickas lika med eller mer än femtusen JSONObjekt. Vilket kan bero på att själva fråge tiden som visas i Figur 16 inte ökar när MongoDB används som databas när fler JSONObjekt används.



Figur 18 Total tid för mätningar med alla test kombinationer stegrandes med antalet hämtade JSONObjekt.

En utav dom mest intressanta delarna av resultatet att analysera är den som visar totala tider. Då det ger en bild på dom olika testkombinationerna står sig emot varandra när alla tider läggs ihop. Då det är enklare att få en bild över om det är någon mätpunkt som sticker ut eller en test kombination.

Som visas i Figur 18 där mätningarna med ettusen JSONObjekt kommer först till vänster med dom fyra testkombinationerna. Sen kommer dom med femtusen och till sist dom med tiotusen JSONObjekt. Det som visas i Figur 18 är att total tiderna ligger väldigt nära varandra och att det ser ut som att det inte gör någon skillnad vilket testkombination som används. Då grafen även visar hur stor del fråge tid och fil tid tar upp. Så går det att se även på T2 att även fast fråge tiden är minimal och tar upp väldigt liten del utan total tiden så ligger även T2 testerna väldigt nära dom andra testernas tider.

5.4 Slutsatser

Dom slutsatser som går att dra utifrån det resultat och den analys som gjorts så är det att mätningarna över dom fyra test kombinationerna är alldeles för lika för att kunna med säkerhet säga att dom är med skilda då man kan se att standardavvikelserna går in i varandra på det horisontella. Som visas i Figur 18 så ser man när man jämför testfallen med varandra på dom olika mängden JSONObjekt som hämtas att dom är väldigt lika. Även om det ska dras en slutsats om skillnaden mellan språken som användes där det som mäts är skillnaderna på tiderna när dom olika filerna laddas. Så går det att se i Figur 17 även här så ligger standardavvikelserna även här när dom olika svarstiderna står emot varandra på den olika mängden JSONObjekt har hämtats väldigt nära varandra. Vilket kan indikera på att det inte finns någon skillnad i filerna. Då kan slutsatsen dras att det inte är någon skillnad mellan PHP och JavaScript.

Den enda svarstiden som går att se att den skiljer sig åt vid att hämtning av olika mängd JSONObjekt är när PHP paras ihop med MongoDB. Om man jämför med MySql med PHP samt JavaScript med både MongoDB och MySql. Detta går tydligt att se i Figur 16 vilket gör att slutsatsen kan dras att när PHP är i hopparat med MongoDB så blir det skillnad i förfrågningen till databasen dock så gör detta ingen skillnad i den totalasvarstiden som visas i Figur 18.

Den slutsatsen som går att dra om den hypotes som är satt. Är att den inte håller då det inte alls gör någon skillnad på den totala svarstiden om man byter ut JavaScript emot PHP. Men den stämmer om man ser till vilken typ av databas om används då det inte gjorde någon som helst skillnad vilken databas som implementerades för den totala svarstiden.

6 Avslutande diskussion

6.1 Sammanfattning

Dom frågeställningar som undersöks i dennas studie undersöker om hur svarstider påverkas på webbapplikationer när Nginx eller Node.js är webbservrar. Där Node.js använder sig av JavaScript för att prata med dom olika databastyperna och Nginx använder sig av PHP för att prata med databastyperna. Där databastyperna är NoSql i form av MongoDB och MySql som en RDBMS. Där frågeställningen ställs på hur svarstiderna påverkas när PHP/JavaScript ställer en fråga till databaserna, hur lång tid det tar att ladda PHP/JavaScripts fil och tillslut hur stor skillnad det är på den totalatiden. När olika stor mängd data skickas igenom webbapplikationerna i form av JSONObjekt.

Där metoden som använts för att få fram svarstider att utvärdera emot frågeställningarna är ett experiment inspirerat av Chaniotis, Kyriakou och Nikolaos (2015), Yishan och Manoharan (2013) och Cattel (2010). Där resultatet är svarstider med olika testfalls kombinationer. Där första testfallet är Nginx/PHP/MySql, andra är Nginx/PHP/MongoDB, tredje är Node.js/JavaScript/MongoDB och fjärde är Node.js/JavaScript/MySql. Där resultatet för denna studie inte kan visa på någon skillnad i totaltiden utifrån testfallen som körts. Det ses heller ingen skillnad i om filerna med PHP och JavaScripts jämförts med varandra i sätt till svarstider. Den enda förbättringen sätt till tider som går att se är när PHP gör förfrågningar till MongoDB då det visar på en markant skillnad i svarstider jämfört med det första, tredje och fjärde testfallen.

6.2 Diskussion

Om man skall diskutera problemet som är beskrivet i problembeskrivningen och om resultatet har besvarat hypotesen som är framlagd. Så borde först frågeställningarna diskuteras i om dom har några tydliga svar utifrån presenterat resultat. Samt om det finns några brister i hur resultatet har tagits fram och presenterats. Det är också intressant om det finns några samhälleliga bidrag eller risker med studien. Det är även så att det forskningsetiska med analys av resultatet och om det harpresenterats på ett korrekt forskningsetiskt sätt.

Det första som borde diskuteras är det problem som Chaniotis, Kyriakou och Nikolaos (2015) gav ifrån sig för framtida studier om att PHP är ett språk som inte håller för nya moderna webbapplikationer för att skicka stor mängd data. Om man ser till det resultatet som visas i denna studie så visas ingen signifikant förbättring sätt till svarstider om man skickar mycket data med JavaScript gentemot PHP. Men eftersom denna studie inte mäter på svarstider istället för CPU och minnes hantering. Så är det mycket möjligt att deras frågeställning fortfarande inte är besvarad. Det är ju också så att det inte är tillräckligt mycket data eller kod i dom olika språken i denna studie som använts för att få svar på deras frågeställning. Men om frågeställningen som finns i denna studie om det finns någon markant skillnad emellan PHP och JavaScript så visar resultatet på att det inte finns någon.

Det andra som borde diskuteras är lagringen av data som i detta fall är JSONObjekt och dom olika typer av databaser som använts är en RDBMS i form av MySql och en NoSql databas i form av MongoDB där båda kan lagra rena JSONObjekt. Där frågeställningen undersöker om det ger någon markant skillnad beroende på vilken typ av databas. Samt där även

hypotesen som finns tror att det inte gör någon skillnad. Som visas i resultatet så finns det enbart en signifikant skillnad när PHP gör förfrågningar emot MongoDB vilket är lite konstigt att tiderna är så mycket mindre än dom andra förfrågningstiderna och att samma låga resultat inte kommer upp när Node.js med JavaScript gör förfrågningar emot MongoDB. Men det är den enda skillnaden som går att se och visa i resultatet. Men det är också intressant att diskutera hur databastyperna utvärderas och definieras. Det är frågan om den metod som användes för att utvärdera databas typerna är tillräckligt bred för att ge ett ordentligt svar. Då Catell (2010) gjorde två stycken förfrågningar för att utvärdera databastyperna emot varandra och för visa är inte ens två stycken tillräckligt då Yishan och Sathiamoothy (2013) använde sig utav fem stycken olika förfrågningar på databastyperna för att få fram en utvärdering av dom. Då är frågan om det är tillräckligt med en typ av förfrågning räcker för att utvärdera dessa databastyper och om det skulle köras upp till fem olika kanske resultatet hade sätt annorlunda ut i fördel för någon databastyp eller PHP/JavaScript.

När det gäller RDBMS så är dom uppbyggda på relationer och frågan är om när MySql fortfarande fungerar ordentligt som en RDBMS när rena JSONObjekt lagras i den. Vi ser i resultatet att det inte gjorde någon skillnad i den förfrågningen som gjordes i denna studie samt det sättet data lagrades i form av JSONObjekt. Då är en viktig sak att diskutera om det sättet denna MySql databasen som användes för JSONObjekten strider emot Codds (1970) teori om hur en relationsdatabas definieras. Samt om det skulle göras upp till fem olika förfrågningar på databastyperna skulle MySql fallera. Men i denna studie görs en förfrågning och den visar inte på någon direkt vinning i att använda sig av någon specifik databastyp.

Sett ur ett samhälleligt perspektiv så finns det både nyttor och risker med resultatet av denna studie. Då det ger en bild av att det inte är någon riktig skillnad på dessa två språk i att utveckla webbapplikationer. Det betyder mer val möjligheter till den som ska utveckla eller beställa webbapplikationer. Det visar också på att det går att göra ett val att utveckla allt i ett och samma språk exempel JavaScript då tekniken finns i Node.js. Där det kan finnas andra egenskaper i Node.js vilket kan resulterar i att det börjar utvecklas mer system och webbapplikationer i just Node.js. Där utvecklare kan fokusera på ett språk att lära sig för att kunna utveckla webbapplikationer vilket kan leda till att utbilda utvecklare kan bli kortare och där med fler utvecklare på marknaden. Desto mer som utvecklare något desto större konkurrens om jobb vilket leder till att utvecklingen går framåt mycket snabbare i samhället. Vilket i sin tur kan leda till att vi har varutransporter som sköter sig själva ute på vägarna eller tillochmed drönare som levererar produkter till dörrkanten automatiskt. Dom risker som ligger närmast i framtiden är att dom nyare teknikerna och språken inte har utforskats tillräckligt där det kan finnas buggar eller utvecklingssätt att utnyttja för en obehörig person kommer in i systemen. Där om ett system med drönare som flyger runt utan pilot och en obehörig person kan ta över systemet som är uppbyggt i ett system där som bara är utvecklat i ett språk till exempel JavaScript kan leda till oönskade konsekvenser. Sen finns det med buggar där saker och ting kraschar för att utvecklingssättet eller språket havererar. Vilket kan leda till i exemplet med drönare som flyger runt att dom helt plötsligt levererar till fel ställen. Men det värsta som kan hända i om det inte finns tillräckligt med riskhantering för buggarna är att det kommer finnas drönare som störtar lite var stans. Eller att utvecklingen tar stopp av tekniken för att den anses som farlig eller samhälles risk.

Det forskningsetiska perspektivet är något som bör diskuteras, finns det några brister i hur resultatet presenterats. Resultatet presenteras med standardavvikelser som gör det forskningsetiskt där genom att kolla på standardavvikelserna kan slutsatser dras. Men för att få det statistiskt säkerställt skulle det behövas T-tester emellan resultaten för att med säkerhet påpeka skillnader eller icke skillnader i resultatet. Detta kan ses som ett forskningsetiskt problem att det inte finns i denna studie. En annan sak som kan ses som ett forskningsetiskt problem är att det är tusen mätningar per testfall vilket skulle behöva gå upp emot fler iterationer. Det visas heller inte det finns spikar i mätningarna vilket också kan vara ett forskningsetiskt problem. Det är också en fråga om experimentet har hög återupprepbarhet vilket inte ses som ett forskningsetiskt problem så länge hårdvaran finns och även fast hårdvaran inte finns går det att köra på nyare hårdvara. Så länge dom versionerna av mjukvara samt att koden finns så är experimentet fullt återupprepar. Men en notis om att hårdvaran ändrats borde finnas i om studien ska återupprepas.

Det sista som kan diskuteras är om det finns något verklig användbarhet eller en produkt utifrån denna studie. Det som kan tänkas eftersom mer och mer data skickas emellan servrar till användare. Är en förfinad lösning till webbapplikationer som använder sig av mycket data och trafik. Där dom måste göra ett val av vilket utveckligs sätt med programmerings språk. Eftersom denna studie använder sig av en stor databas som många har nytta av att ta del av så är det intressant hur man får ut data till så många användare som möjligt. Ett förfinat sätt att få ut data.

6.3 Framtida arbete

Framtida arbete som är intressant i det kortsiktiga perspektivet som är statistiskt signifikant är att öka dom rader kod som mäta på i dom två olika språken för att få fram större skillnader emellan dom. Där mer funktionalitet kan undersökas och mer frågor uppstår om i vilka sammanhang PHP eller JavaScript är ett bättre val för att få ner svarstiderna hos webbapplikationerna. Det skulle även vara intressant att testa språken hur som står sig emot olika webbläsare. Är det så att använda sig av mer PHP eller JavaScript för att få ner svarstider på olika typer av webbläsare. Då kan frågan ställas om vilket språk ska användas till vilken webbläsare man utvecklar till eller tar man det språket som ger minst svarstider överlag emot dom vanligaste webbläsarna. Även i det kortsiktiga perspektivet är det intressant att göra stress tester på dom olika språken/testfall som togs upp i denna studie. Då Chaniotis, Kyriakou och Nikolaos (2015) gjorde stress tester med samtidiga anslutningar och anslutningar per sekund. Vilket också i dagens högtrafikerade webbapplikationer vill man ha ner svarstider även fast det är hög trafik eller om det är många som ansluter på en och samma gång. Detta är också en intressant del att ta som nästa steg i forskningen.

I det långa perspektivet kan man tänka sig att virtuella verkligheter letar sig in på webbapplikationer vilket kommer leda till att mer kod samt mer data kommer vara tvunget att användas i större skala. Då blir det intressant ifrån denna studie om antal rader kod och mer data skalas upp samt skickas mellan server och webbapplikationen. Då kan framtida arbete bli högst relevant om PHP fortfarande håller eller ska webbapplikationerna köra ren JavaScript med Node.js för att kunna skapa en virtuell verklighet för användaren. Det är också intressant att fortsätta arbetet med vilken databas som ska användas. Kommer en RDBMS kunna ge ifrån sig data tillräckligt snabbt för den virtuella verkligheten eller blir

man tvungen att använda sig av en NoSql som i teorin ger ifrån sig data snabbare. Kommer en NoSql databas inte hålla då den inte kan sortera all information lika bra som en RDBMS och om RDBMS blir ett måste då relationer av data kanske blir mycket viktigare om man ska skapa en avancerad virtuell verklighet.

Ett framtida arbete om JSONObjekt är ifall det är nödvändigt att få ut ett JSONObjekt ifrån en databas för att spara tid eller om det räcker med att applikationerna tar hand om det själva så länge databasen ger ifrån sig en string som går att göra till ett JSONObjekt. Kommer JSON strukturen hålla säkerhetsmässigt eller kommer den gå att infektera. Går det verkligen långsammare att göra om strukturen i applikationen. Är det så att applikationen vinner på att ha en RDBMS som oftast ger ifrån sig en string men kan ta emot mer komplexa frågeoperationer. Eller räcker det med en NoSql för dom flesta applikationerna som oftast kan ge ifrån sig klara JSONObjekt. Samt vid hur stor mängd data tar det alldeles för lång tid för att använda sig av strukturerade relationer med en RDBMS och man måste gå över till en NoSql.

Referenser

Abramova, V. & Bernardino, J. (2013) NoSQL Databases: MongoDB vs Cassandra. Proceedings of the International C* Conference on Computer Science and Software Engineering. C3S2E '13. New York, NY, USA, ACM. s. 14–22.

Cattell, R. (2010). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*. New York. USA. December, 2010. Vol 30, Issue 4, ss. 12-27.

Chaniotis, I, Kyriakou, K-I & Tselikas, N. (2015) Is Node.js a viable option for building modern web applications? A performance evaluation study. *Springer Journal of Computing*. 97 (9), s.1023-1044.

Codd. E. F. (1970). A relational model of data for large shared data banks. Communications of ACM, 13(6), 377-387.

Li, Y. & Manoharan, S. (2013) A performance comparison of SQL and NoSQL databases. 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM). August s. 15–19.

Mickens, J (2010) Exploiting Java Scriptand DOM Storage for Faster Page Loads. *Proceedings of WebApps*. 2010 Unisex. June.

Usgs (2016) Mineral Resources Data System. Tillgänglig på Internet: http://mrdata.usgs.gov/mrds/ [Hämtad Mars 30, 2016].

Nixon, R. (2009) Learning PHP, MySQL and JavaScript: *A Step-by-Step Guide to Creating Dynamic Websites*. O'Reilly Media.

Nginx Inc (2016).
https://www.nginx.com/company/[2016-02-10].

Powel, J (2009). 43,439 reasons to use append() correctly. Tillgänligt på Internet: http://www.learningjquery.com/2009/03/43439-reasons-to-use-append-correctly [Hämtad April 15, 2016].

Richards, G., Lebresne, S., Burg, B & Vitek, J. (2010) An analysis of the dynamic behaviour of JavaScript programs. *10 Proceeding of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, USA 2010, ss. 1-12. DOI: 10.1145/1806596.1806598.

Ojamaa, A. & Düüna, K. (2012) Assessing the security of Node.js platform. Internet Technology And Secured Transactions, 2012 International Conference for. December s. 348–355.

Sangeeta, G & Sangeeta, N. (2015) Performance Evaluation of Nosql-Cassandra over Relational Data Store-Mysql for Bigdata. *Journal of Technology*. 6 (4), s. 640-649.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. & Wesslén, A (2012) Experiment in Software Engineering. Berlin Heidelberg: Springer. ISBN 978-3-642- 29043-5.

Appendix A - Index.html.

```
<!DOCTYPE html>
<html>
	<head>
		<title>load my data plx</title>
		<link rel="stylesheet" href="./stylesheets/style.css">
	</head>
	<body>
		<!-- the table which the data is loaded into. -->
		<table id="resultdata">
			<tr><th>_id</th>
				<th>dep_id</th>
				<th>mrds_id</th>
				<th>mas_id</th>
				<th>latitude</th>
				<th>longitude</th>
				<th>region</th>
				<th>country</th>
				<th>state</th>
				<th>com_type</th>
				<th>commod1</th>
				<th>commod2</th>
				<th>commod3</th>
				<th>dep_type</th>
				<th>prod_size</th>
				<th>dev_stat</th>
				<th>ore</th>
				<th>gangue</th>
				<th>orebody_fm</th>
				<th>work_type</th>
				<th>model</th>
				<th>alrernation</th>
				<th>conc_proc</th>
				<th>ore_ctrl</th>
				<th>htock_unit</th>
				<th>hrock_type</th>
				<th>arock_type</th>
				<th>structure</th>
				<th>tectonic</th>
				<th>ref</th>
			</tr>
		</table>
		<script src=" ./javascripts/jquery.js"></script>
		<script src="./javascripts/global.js"></script>
	</body>
</html>
```

Appendix B - Global.js

```
$(document).ready(function(){
	var htttprequest = $("#start").text();
	var documentReadyTime = performance.now();
	var dataFileLoadTime = 0;
	var htmlTime = 0;
	var queryTimer = 0;
	var table = document.getElementById("resultdata");
	var jsonObject='php10000';

	/*
	 ./getdataMysql.php for php with mysql
	./getdataMongoDB for php with MongoDB
	 /getdataMysql for JavaScript in Node.js with MySql
	/getdataMongoDB for JavaScript in node.js with MongoDB
	*/
	$.getJSON('./getdata.php', function(data){
	}).done(function(data){

		var timerDone = performance.now();
		dataFileLoadTime = ((timerDone - documentReadyTime)/1000)%60;

		var tableinput = "";
		var tableInputTd = "";
		var oneTimeDeal= 0;
		var talbeInputTh= "<tr>";
		var countTd = 0;
		var insertTd = [];
		var insertTh = [];
		queryTimer = data['timeer'];

		var length = Object.keys(data['infojson']).length;

		var values = data['infojson'];

		var contentInsert = [];
		var i = 1;
		for (var a = 0; a < length; a++){
			contentInsert[i++] = '<tr>';
			var temp = JSON.parse(values[a]);
			for(var key in temp){
				contentInsert[i++] = '<td>';
				contentInsert[i++] = temp[key];
				contentInsert[i++] = '</td>';
			}
			contentInsert[i++] = '</tr>';
		}
```

```
            $('#resultdata').append(contentInsert.join(''));

            htmlTime = (performance.now()/1000)%60;

            var logData = {
                'dataFileLoadTime' : dataFileLoadTime,
                'htmlTime' : htmlTime,
                'queryTimer' : queryTimer,
            };

            $.ajax({
                type: "POST",
                data:{
                    jsonObject:jsonObject,
                    dataFileLoadTime: dataFileLoadTime,
                    htmlTime: htmlTime,
                    queryTimer:queryTimer
                },
                url: './savedata.php'

            }).done(function(){
                console.log(' Data Saved!!!!!!');
                if(localStorage.getItem('counter') < 1000){

                    location.reload();
                }else{
                    console.log('experiment done');
                    localStorage.setItem('counter',-1);
                }
                var i = localStorage.getItem('counter');
                i++;
                console.log(i);
                localStorage.setItem('counter', i);
            }).fail(function(){
                console.log('Save data failed');
            });

        }).fail(function(data){
            console.log("im in fail");
        });


});
```

Appendix C - App.js in Node.js environment

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var mongodb = require('mongodb');
var monk = require('monk');
var db = monk('localhost:27017/nodetest1');

var mysql = require('mysql');
var connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'guest',
  database: 'coordsDb',
});

var routes = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));


//mongodb accesible to router

app.use(function(req,res,next){
  req.db = db;
  next();

});

app.use(function(req,res,next){
```

```
  req.connection = connection;
  next();

});

app.use('/', routes);
app.use('/users', users);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handlers

// development error handler
// will print stacktrace
if (app.get('env') === 'development') {
  app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.render('error', {
      message: err.message,
      error: err
    });
  });
}

// production error handler
// no stacktraces leaked to user
app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.render('error', {
    message: err.message,
    error: {}
  });
});


module.exports = app;
```

Appendix D - Index.js in Node.js environment

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

router.get('/getdataMysql', function(req,res){

        var connection = req.connection;
        var sqlquery = "SELECT * FROM coords limit 10000;";
        var end = null;
        var dbconnecttime = null;
        var start = new Date().getTime();
        var data = {
                infojson:[],
                timeer:[]
        };

  connection.query(sqlquery,function(err,rows,fields){
    if(rows.lenght != 0){
        data.infojson = rows;

                end = new Date().getTime();
                dbconnecttime = (end - start)/1000;//1000
                data.timeer = dbconnecttime;
                res.json(data);
    }else{
      data.infojson = 'connection deosent work!';
      res.json(data);

    }

  });

});

router.get('/getdataMongoDB', function(req,res){
        var db=req.db;
        var collection = db.get('coords');

        var end = null;
        var dbconnecttime = null;
        var start = new Date().getTime();
        var data = {
```

```
            infojson:[],
            timeer:[]
        };

        collection.find({},{limit:1000},function(e,docs){

            data.infojson = docs;

            end = new Date().getTime();
            dbconnecttime = (end - start)/1000;
            data.timeer = dbconnecttime;
            res.send(JSON.stringify(data,null,3));

        });

});

router.get('/loaddata',function(req,res){
        var loadDataStart = new Date().getTime();
        //console.log("this is le starttime" + loadDataStart);
        res.render('loaddata', {startTimer: loadDataStart});
});

router.post('/savedata',function(req,res,next){
        var connection = req.connection;

        var post = {
            jsonObjekt: req.body.jsonObject,
            querytime: req.body.queryTimer,
            filetime: req.body.dataFileLoadTime,
            totaltime: req.body.htmlTime
        };

        var query = connection.query('INSERT INTO extimes SET
?',post,function(err,result){
        });

});


module.exports = router;
```

Appendix E - getdataMysql.php on the Nginx server

```
<?php
        $servername = "127.0.0.1:3306";
        $username = "root";
        $password = "guest";

        $conn = new mysqli($servername, $username, $password,"coordsDb");

        $query = "SELECT * FROM coords limit 1000";

        $data = array('infojson' => array(), 'timeer' => array());
        $start=microtime(true);
        $returndata = mysqli_query($conn,$query);

        $stopTime= (microtime(true) - $start);
        while($r = mysqli_fetch_assoc($returndata)) {
        array_push($data['infojson'], $r['jdoc']);
        }
        $data['timeer'] = $stopTime;
        print_r(json_encode($data));

        mysqli_close($conn);
```

Appendix F - getdataMongoDB.php on Nginx server

```
<?php
	$start_time = microtime(true);

	$m = new MongoClient();

	$db = $m->nodetest1;

	$data = array('infojson' => array(), 'timeer' => array());

	$collection = $db->coords;

	$cursor = $collection->find()->limit(10000);
	$end_time = microtime(true);
	foreach ($cursor as $document => $row){
		array_push($data['infojson'], $row);

	}
	$total_time = ($end_time - $start_time)/1000;

	array_push($data['timeer'], $total_time);
	print_r(json_encode($data));
```