# Introduktion

Denne performance test rapport har til formål at belyse evt. performance problemer og forbedringer med TM12 projektet.

Rapporten er en initial analyse og afviklingen er derfor foretaget på en udviklings-PC. Både SUT og selve tist-miljøet bliver eksekveret på samme PC. Dette er naturligvis ikke optimalt, da belastningen fra test-miljø’et vil reducere i reduceret performance i SUT.

Der bliver udført tre test.

1. Ffør-test til at vise hvordan belastningen er når TM12 er i hvile.
   1. Dette vil vise om systemet allokerer data eller anvender betydelig CPU-kraft på at lave ingenting. Der skal naturligvis anvendes lidt på evt. problemer eller ineffektiv design, da
2. Load test
   1. Disse test vil vise hvordan system fungerer under realistisk belastning. Der vil blive eksekveret flere test for at se hvordan systemet reagere alt efter hvor hårdt det belastes, of dermed også at finde max user count.
3. Stress test
   1. Denne test vil vise hvordan systemet reagere når det belastes med et stort antal read og write tråde.
4. Efter-test
   1. Dette vil vise om systemet er i stand til at vende tilbage til hvile-tilstand, eller om der er ressourcer der ikke korrekt ryddes op.

For at gøre dokumentet mere overskueligt er test resultaterne flyttet til separate foldere og dokumenter, og dette dokument vil kun indeholde overvejelser, analyse og konklusion.

Balanceret load

# Test mål

En performance test uden noget mål er ikke meget værd. I dette tilfælde er målet at se efter muligt performance forbedringer. Da vi ikke kender det forventede load, er det naturligvis svært at sige med sikkerhed hvor det er bedst at sætte ind. VI kan dog gøre visse antagelser:

1. Der er tale om en webservice, og en gennemsnitlig svartid under 1 sekund er derfor forventeligt.
2. Det er en service til et mindre udsnit af befolkningen, og der er derfor ikke tale om åben web service. Antallet af simultane brugere er derfor relativt lille. Det præcise tal får vi fra performance testen, der rent faktisk vil kunne sige hvor mange simultane brugere vi har.
3. Som de fleste webservices er det sandsynligt at der vil være betydeligt flere read requests end write request.
4. Da DB server og application server er separeret er det et mål i sig selv at få deres load til at være balanceret, da det altid giver den bedste performance.
5. En webservice til lægeligt personale vil være belastet 24 timer i døgnet, men vil have spidsbelastning fra 8-10 og med et højere belastningsniveau end gennemsnittet fra 10 til 16. Det er derfor vigtigt at webservicen kan håndtere ”kortere” peaks i aktivitetsniveauet.

# Test setup

## Hardware

CPU: Intel® Core™ i7-2640M CPU @2,80GHz

RAM: 4GB

OS: Windows 7 Professional SP1 – 64bit

HDD: SSD

Test eksekvering

### Programmer

Java v1.7.0\_02

JMeter v2.8

JMeter plugin server agent v0.5.5

Mongodb v2.2.1

Eclipse v3.7.2 (anvendt udelukkende til at eksekvere fra. Ant anvendet til start og det er et Run ikke Debug).

Ant v1.8.4

Jvisualvm v1.7.0\_02

### Eksekvering

1. Først startes Jvisualvm.
2. Dernæst JMeter Plugin server agent.
3. Så Mongodb.
4. Så JMeter og test scriptet indlæses.
5. Der udvælges det antal reader og writer threads der vil bruges og der sættes den ramp-up tid hvor testene skal startes.
6. Inde fra eclipse startes cs.saip.main.JettyServerMain med argumenterne ” mongod 8080”
7. Dernæst startes JMeter testen.
8. Der overvåges kontinuerligt at der ikke opstår fejl under testen.
9. Ved testens afslutning tages screenshot fra CPU og Memory forbrug fra Jvisualvm
10. Testen stoppes og resultatet uploades til loadsophia hvor en PDF genereres som gemmes sammen med screenshots til senere analyse.
11. Alle åbne programmer lukkes og genstartesved næste testkørsel.

## TM12

Mongodb enabled

# Før test

Til denne test anvendes jvisualvm til at beregne hvor meget RAM og CPU systemet anvender. Samtidig vil der blive evalueret på hvordan ressourceforbruget er og om der allokeres nogle ressourcer. Da systemet er i hvile bør det være muligt at allokere meget få ressourcer løbende. Hvis systemet er designet til lange perioder med ingen aktivitet, og ressourcerne i hvile skal minimeres, vil det være muligt at allokere de nødvendige ressourcer én gang, i stedet for at oprette og nedlægge dem. Det vil dog være helt op til hvad der er målet for performance.

# Load test

Load testen baserer sig på et test script for vores test-afdeling, og det er designet til at udøve en realistisk belastning af systemet.

Vi har udført flere test for at identificere en godt antal samtidige read og write tråde. Vi kan se at selv med 10 read og 3 write tråde degrader applicationen over tid (response tid stiger). Dette betyder at applikationen ikke kan følge med.

Vi er dog ikke i en situation hvor applikationen fejler. Der har vi kørt en stress test med 700 read tråde og 5 write tråde. I det tilfælde kan applikationen ikke håndtere request, selv med forøget response tid.

Hvis vi anvender en endnu mindre belastning med 5 read og 1 write tråd …

Vi kan se at TPS peaker ved ca. 12, og med list til variation skal vi holde os under 10 tråde for at systemet kan følge med.

Som nævnt tidligere er dette et relativt lukket system til en kendt (mindre) brugergruppe, og det er derfor ikke usandsynligt at dette er et acceptabelt antal simultane read/write. Det er noget der vil skulle diskuteres med brugergruppen. At response tiden stiger i en kort periode er sandsynligvis også acceptabelt, og det antal brugere der skal til før systemet bliver non-responsive er ret højt for denne form for brug, hvilket sandsynligvis også er OK.

For at give mulighed for flere simultane brugere kan vi se at det vil være en fordel at fordele belastningen mellem DB server og application server bedre, da mongodb er betydeligt minre belastet end Jetty, og der er altid en fordel at have et balanceret system. Dette kan gøres ved at lægge mere af logikken ned i DB-requestet (eller skalere HW for at opnå balance).

Det er også muligt at anvende mere caching og færre temporere objekter for at opnå mindre service time, hvilket vil give systemet mulighed for at håndtere flere simultane brugere (ved at flytte response time ned vil der gå længere før systemet ikke kan følge med, som specificeret i kø-teorien).

# Efter test

GC manuelt eksekveret for at se hvile efter test. Dette er gjort efter 10min med hvile og er tydelig i resultatet. Det kan her ses at systemet vender tilbage til sin før-state.

# Konklusion

Uden at kende de konkrete performance krav til systemet er det naturligvis umuligt at give nogle rigtig fornuftige anbefalinger, da det alt sammen vil være under forudsætning af det HW der skal eksekvere systemet og den load som systemet skal kunne håndtere, inklusiv den peak den forventes at kunne håndtere og den evt. forøgelse af load med tiden.

Hvis vi laver følgende antagelser:

1. Test afdeling har afleveret et test-script der repræsenterer normalsituationen.
2. Server PC’en vil være mindst ligeså kraftig som udviklings-PC’en på alle parametre
3. Server PC’en eksekverer både application og DB (ligesom udviklings PC’en) og er dedikeret (ingen andre applikationer eller requests skal serviceres)).

Det er meget normalt at database server og application server er to forskellige servere med en meget hurtig I/O forbindelse. Vi har desværre ikke separeret