
**TEKNIIKKA JA LIIKENNE**

**Tietotekniikka**

**Tietoliikennetekniikka**

**OPINNÄYTETYÖ**

# ERÄÄN WEB-POHJAISEN JÄRJESTELMÄN SUORITUSKYKYTESTAUS

**Työn tekijä: Mikko Järvinen**
**Työn ohjaaja: Auvo Häkkinen**
**Työn ohjaaja: Sonja Pietilä**

**Työ hyväksytty: __. __. 2008**

**Auvo Häkkinen**
**yliopettaja**


**ALKULAUSE**

Tämä opinnäytetyö tehtiin Logica Suomi Oy:lle valtionpalveluiden osastolle.

Kiitän työn projektipäällikkö Sonja Pietilää työn ohjauksesta ja avusta. Kiitos kuuluu myös koulun puolesta työtä ohjanneelle yliopettaja Auvo Häkkiselle. Heidän antamistaan vinkeistä ja ohjeista oli tässä työssä suuri apu. Kiitän myös kaikkia niitä Logican työntekijöitä, jotka olivat auttamassa minua tämän työn tekemisessä.

Lisäksi kiitän Elina ja Jimi Järvistä kaikesta heiltä saamastani tuesta ja ymmärryksestä.

Helsingissä 11.11.2008

Mikko Järvinen

**OPINNÄYTETYÖN TIIVISTELMÄ**

| | |
|---|---|
| **Työn tekijä:** Mikko Järvinen | |
| **Työn nimi:** Erään Web-pohjaisen järjestelmän suorituskykytestaus | |
| **Päivämäärä:** 11.11.2008 | **Sivumäärä:** 43 s. + 2 liitettä |
| **Koulutusohjelma:**<br>Tietotekniikka | **Ammatillinen suuntautuminen:**<br>Tietoliikennetekniikka |
| **Työn ohjaaja:** Yliopettaja Auvo Häkkinen, Metropolia Amk<br><br>**Työn ohjaaja:** Projektipäällikkö Sonja Pietilä, Logica Suomi Oy | |


Tämä opinnäytetyö tehtiin Logica Suomi Oy:n Valtionpalveluiden osastolle. Työn tavoitteena oli suunnitella, valmistella ja toteuttaa suorituskykytestit web-pohjaiseen järjestelmään ja raportoida mahdollisista virheistä, hitauksista tai puutteista sovelluksen kehittäjille, projektipäällikölle ja sovelluksen tilanneelle asiakkaalle.

Työ aloitettiin selvittämällä ja perehtymällä web-sovelluksen suorituskykytestauksen teoriaan ja perusperiaatteisiin yleisellä tasolla. Tämän jälkeen suunniteltiin, valmisteltiin ja toteutettiin suorituskykytestit Logica Suomi Oy:n toimittamaan web-sovellukseen. Testauksessa käytettiin apuvälineenä LoadRunner-ohjelmaa, joka on suorituskykytestaukseen tarkoitettu työkalu.

Testaus on tärkeä osa ohjelmistojen toteutusta, ja suorituskykytestaus on tärkeä osa ohjelmistojen testausta. Varsinkin Web-sovelluksien suorituskykytestaus on tärkeää, jotta saadaan selville sovelluksen pullonkaulat ja suorituskykyongelmat ennen kuin sovellus julkaistaan tai päästetään markkinoille.

Tämän opinnäytetyön tuloksena syntyi kirjalliset raportit testin tuloksista. Raportit toimitettiin sovelluksen projektipäällikölle, sovelluksen kehittäjille ja sovelluksen tilanneelle asiakkaalle. Näitä raportteja voivat sovelluksen toteuttajat käyttää sovelluksen korjaukseen, projektipäällikkö sovelluksen tilanteen kartoitukseen ja asiakas voi vertailla tuloksia edellisiin suorituskykytesteihin.

Testattavan sovelluksen tilannut asiakas toivoi, että heidän yrityksensä, eikä testattavan sovelluksen nimeä julkistettaisi tässä opinnäytetyössä. Tämän vuoksi tässä opinnäytetyössä ei tulla käsittelemään suoraan kumpaakaan edellä mainituista. Työn tuloksena syntyvät testausraportit on myös määritelty ei-julkisiksi.



**Avainsanat:** Web-sovelluksen suorituskykytestaus, LoadRunner

Metropolia

**ABSTRACT**

**Name:** Mikko Järvinen

**Title:** One Web-Based system performance testing

**Date:** 11.11.2008 **Number of pages:** 43 pages + 2 attachements

**Department:** Information technology

**Study Programme:** Telecommunications technology

**Instructor:** Senior teacher Auvo Häkkinen, Metropolia polytechnic

**Supervisor:** Prjoject manager Sonja Pietilä, Logica Suomi Oy

This scholarly thesis was made for the Logica Suomi OY Government services department. The jobs goal was to plan, prepare and follow through performance tests for a web based system and report possible errors, slowness or deficiencies to the software developers, project manager and to the customer who purchased the software.

The work was started by researching and getting to know web-software preformance testing theory and basic principles on a common level. Next was planning, preparing and following through the tests for the software provided by Logica Suomi OY. The actual testing was made with using a Loadrunner program which has been created for performance testing.

Testing is an important part of software developing and the performance testing is very important part of the whole testing process. Especially in web based software the performance testing is important so that the bottlenecks and other performance problems of the software will be found before publishing and releasing the product.

As the result of this engineering job there are written reports of the testing findings. The reports were taken to the software project manager, developers and the actual customer. These reports can be used by the developers to repair and improve the software, by the project manager to map the status of the product, and by the customer to compare it to previous test results.

Because the customer did not want their name or the products name published, the reports are defined not public.

**Keywords:** Performance testing for web-application, LoadRunner

**SISÄLLYS**

# 1 JOHDANTO

Tässä työssä suunniteltiin ja toteutettiin erään web-pohjaisen sovelluksen suorituskykytestit. Testit toteutettiin, koska muutaman vuoden tuotannossa olleesta sovelluksesta oltiin julkaisemassa uusi versio. Tuli siis selvittää, että sovellukseen toteutetut uudistukset ja muutokset toimivat moitteettomasti sovelluksen tuotannossa kohtaaman kuormituksen kanssa.

Testitulokset analysoitiin ja niistä muodostettiin testausraportit. Analysoinnin tuloksena syntyi kaksi erilaista testausraporttia, joista kattavampi raportti toimitettiin sovelluksen kehittäjille ja projektipäällikölle ja toinen, vähän suppeampi raportti, toimitettiin sovelluksen tilanneelle asiakkaalle.

Työn tuloksena syntyneistä raporteista selviää testattavan sovelluksen

- suorituskyky
- mahdolliset pullonkaulat
- mahdolliset ei-toimivat ohjelman osat
- mahdolliset parannus- ja korjausehdotukset.

Työn alussa perehdytään suorituskykytestauksen teoriaan ja tarkastellaan suorituskykytestausta yleisellä tasolla. Lisäksi tarkastellaan suorituskykytestauksen hyötyjä ja millaisia eri suorituskykytestauksen tapoja on olemassa. Työn teoriaosuudessa käsitellään myös suorituskykytestaus-prosessia ja mitä kaikkia toimintoja siihen kuuluu.

Työssä on myös yleiset ohjeet, kuinka testiskriptit nauhoitetaan LoadRunner-työkalulla.

Tämän työn päähuomio on kuitenkin erään web-pohjaisen järjestelmän suorituskykytestaus ja siitä syntyvien tuloksien analysointi ja raportointi. Testit toteutettiin suorituskykytestaussovellus LoadRunnerilla ja erillisellä XML-viestitesterillä. Työssä on kerrottu kuinka testit suunniteltiin ja toteutettiin.

Työn lopussa testitulokset analysoitiin LoadRunnerin Analysis-ohjelmalla. Analysoiduista testituloksista koostettiin kaksi erilaista raporttia. Toinen raportti toimitettiin asiakkaalle ja toinen Logicalle.

Testausraporteista selviää, että sovelluksesta löytyi muutamia liian hitaita komponentteja. Toinen näistä koski XML-liikennettä ja toinen lupahakemuksen muodostamista. Näihin hitauksiin pyritään löytämään korjauksia ennen sovelluksen toimitusta asiakkaalle. Muutamaa hidasta komponenttia lukuunottamatta testattu sovellus toimi moitteettomasti.

Testattavaa sovellusta ei tässä työssä käsitellä kovinkaan tarkasti, koska sovelluksen Logicalta tilannut yritys ei halunnut omaa nimeään, eikä sovelluksen nimeä julkaistavaksi. Tämän takia työssä esiintyviä sovelluksen näyttöjä ja liitetiedostoja on jouduttu karsimaan tai niitä on jätetty ei-julkisiksi.

## 2 SUORITUSKYKYTESTAUS

Suorituskykytestauksen (engl. Performance testing) tavoitteena on saada selville tietokoneen, verkon, sovelluksen tai jonkun laitteen nopeus ja toimivuus kovan rasituksen alaisuudessa. Tämä prosessi voi sisältää testejä testiympäristössä, esimerkiksi vasteaikojen mittaamista testattavasta sovelluksesta tai testattavan sovelluksen suorituskyvyn mittaamista apuvälineitä käyttäen. Tällaisia apuvälineitä löytyy useita. On niin sanottuja open source -sovelluksia ja maksullisia lisenssipohjaisia sovelluksia. [7]

Suorituskykytestaus todentaa, että testattu kohde toimii sille asetettujen raja-arvojen puitteissa. Yleensä nämä raja-arvot asettaa testattavan kohteen tilaaja ja/tai sen toimittaja. Raja-arvoja asetetaan kohteen nopeudelle, datan siirtonopeudelle, siirtokaistalle, kapasiteetille, hyötysuhteelle ja luotettavuudelle.

Suorituskykytestausta käytetään myös tunnistamaan ja etsimään kohteen pullonkaulat. Nopeakin kohde voi toimia hitaasti, jos sillä on huonosti tai väärin toteutettu komponentti tai järjestelmän osa. Sanonta ”ketju on yhtä vahva kuin sen heikoin lenkki” toimii siis omalla tavallaan myös suorituskykytestauksessakin.

Suorituskykytestauksessa kiinnitetään huomiota seuraaviin seikkoihin: [5]

- Nopeus - vastaako testattavan sovelluksen nopeus määrittelyjä?
- Kapasiteetti – vastaako laitteisto määrittelyjä?
- Skaalautuvuus - pystyykö sovellus esimerkiksi hallitsemaan mahdollisen käyttäjämäärien kasvun?

- Vakaus - käyttäytyykö testattu sovellus oikein kovan kuormituksen alaisuudessa?

## 2.1 Suorituskykytestauksen tarkoitus ja hyödyt

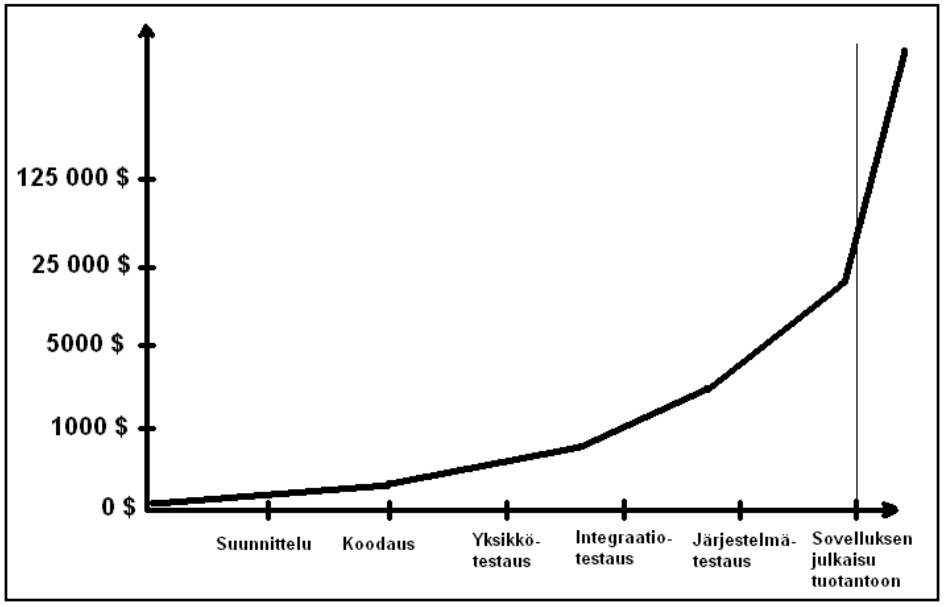
Suurin syy sovelluksen suorituskykytestauksen toteuttamiselle löytyy yleensä kustannuksien hallinnasta, tuotteen jatkuvuudesta, yrityksen maineesta ja riskien hallinnasta.

Yrityksille on yleensä todella tärkeää säilyttää yrityksen maine. Huono ohjelmisto väärässä ja tärkeässä paikassa voi tuhota yhtiön maineen, olkoon sitten kyseessä sovelluksen toimittaja tai sovelluksen tilannut asiakas.

Myös monia muita syitä suorituskykytestaukselle on olemassa. Tällainen syy on esimerkiksi varmistaa sovelluksen julkaisu-valmiustila. Suorituskykytestejä toteuttamalla varmistetaan sovelluksen toiminta asiakkaan antamien raja-arvojen puitteissa. Näin ollaan varmempia siitä, että sovellus tulee olemaan toimiva ja vakaa tulevassa tuotannossa. [1, s. 4.]

Suorituskykytestauksella pystytään myös varautumaan mahdolliseen sovelluksen tulevaisuuden kasvuun, esimerkiksi käyttäjämäärien räjähdysmäiseen nousuun. Nykypäivänä monet sovellukset ovat kaatuneet yllättäviin käyttäjämäärien kasvuun, kun ne eivät ole osanneet varautua tarvittavin keinoin. Tästä esimerkkinä on Internet-sivusto Facebook, joka joutui kasvattamaan laitteistojaan ja tietokantojaan rajusti, kun sen sivustoista tuli maailmanlaajuinen hitti.

Suurin yksittäinen syy suorituskykytestauksen toteuttamiselle lienee rahan kuluminen tai oikeastaan sen säästäminen. Seuraavassa kuvassa on havainnoillistettu sovelluksesta löytyvien virheiden  korjauskustannusten kehittymistä sovelluksen kehityksen eri vaiheissa.




*Kuva 1. Virheen löytymisen kustannukset sovelluksen eri kehitysvaiheissa (HUOM! Kuvassa esiintyvät rahamäärät ovat arvioita) [3, mukailemalla s. 9 kuvaa.]*

Kuten kuvasta 1 huomataan, jos sovelluksen virhe havaitaan vasta sovelluksen julkaisun jälkeen, niin kustannukset ovat moninkymmenkertaiset verrattuna siihen, jos virhe olisi havaittu jo testausvaiheessa. Tämän vuoksi on tärkeää toteuttaa suorituskykytestejä jo varhaisessa vaiheessa sovelluksen kehitystä.

Niiden syiden takia monet ohjelmistoalan yritykset ovat ottaneet suorituskykytestauksen osaksi testaussuunnitelmiaan.

## 2.2 Suorituskykytestauksen muodot

Suorituskykytesteillä voidaan testata koko sovelluksen suorituskykyä. Testit voidaan kohdistaa tiettyyn osaan testattavaa sovellusta tai niillä voidaan selvittää mikä on sovelluksen hajoamispiste, eli missä pisteessä sovellus ei pysty enää käsittelemään enempää kuormitusta. Lisäksi voidaan testata myös sitä, kuinka sovellus käyttäytyy pitkän ja kovan kuormituksen alaisuudessa.

Suorituskykytestauksesta käytetään usein ainoastaan nimeä suorituskykytestaus, vaikka sillä onkin useita muotoja. Tällaisia ovat muun muassa kapasiteettitestaus, komponenttitestaus, kestävyystestaus, kuormitustestaus, savutestaus ja stressitestaus.

*Kapasiteettitestaus*

Kapasiteettitestauksella (engl. capacity testing) saadaan selville, millaisella kuormituksella sovellus pysyy vielä sille määritettyjen raja-arvojen sisäpuolella. Tällä testausmuodolla selvitetään myös se, pystyykö sovellus nykyisillä laitteistoilla täyttämään sille asetetut vaatimukset. [1, s. 10-11, 17.]

Sovelluksen suunnittelijat ja määrittelijät saavat kapasiteettitestauksesta saatavista tuloksista tärkeää informaatiota sovelluksen mahdollisista parannus- ja päivitystoimenpiteistä.

*Komponenttitestaus*

Komponenttitestauksella (engl. component testing) saadaan selville, minkälainen kuormitus voidaan kohdistaa sovelluksen arkkitehtuaalisiin osiin, kuten esimerkiksi tietokantoihin, servereihin, verkkoihin, palomuureihin ja muisteihin. [1, s. 11.]

*Kestävyystestaus*

Kestävyystestauksella (engl. endurance testing) pyritään selvittämään, miten testattavan sovelluksen suorituskyky käyttäytyy pitkän ajan kuluessa. Tällä testausmuodolla selvitetään esimerkiksi muistin täyttymisongelmia, joita ei lyhytkestoisilla testeillä saada selville. [1, s. 11.]

*Kuormitustestaus*

Kuormitustestauksella (engl. load testing) testataan, miten sovellus toimii määriteltyjen raja-arvojen ylärajoilla. Kuormitustestauksella pyritään myös selvittämään sovelluksen laitteiston kokoonpanon riittävyys tai päivittämistarve. [1, s. 8, 16.]

Kuormitustestauksen tarkoitus ei ole jumiuttaa sovellusta, mutta se helpottaa löytämään sovelluksen niin sanotut kipupisteet, eli toisin sanoen kuinka monta samanaikaista käyttäjää sovellus pystyy käsittelemään, ennen kuin se joutuu hylkäämään käyttäjiä.

Kuormituskykytestauksella pystytään myös keräämään tärkeitä tietoja, jotka auttavat vakaus- ja kapasiteettisuunnittelussa.

*Savutestaus*

Savutestaus (engl. Smoke testing) ajetaan yleensä ennen suorituskykytestauksen alkamista ja sillä todetaan, että sovellus toimii normaalilla kuormituksella täysin normaalisti. Tällä varmistetaan se, että ei aloiteta suorituskykytestauksia, ellei sovellus pysty käsittelemään edes normaalia kuormitusta. [1, s. 13.]

*Stressitestaus*

Stressitestauksella (engl. stress testing) saadaan selville milloin testattava sovellus lakkaa toimimasta oikein, kun sille määritellyt raja-arvot ylittyvät. Stressitestauksella siis testataan, miten testattava sovellus kestää niin sanotun ”worst case”- tapauksen. Eli tapauksen, joka olisi pahin mahdollinen, jonka sovellus voisi tuotannossa kohdata. [1, s. 8-9, 181.]

Stressitestauksella saadaan selville, kuinka paljon yli raja-arvojen sovellus pystyy, yleisen hidastumisen lisäksi, menemään ilman, että se vahingoittaa dataa ja aiheuttaa virheitä.

Stressitestauksella saadaan myös selville, heikkeneekö sovelluksen tietoturva raskaan kuormituksen aikana. Palomuuri ja tietoturva-sovellukset voivat nimittäin kaatua kovan rasituksen jälkeen, tai sitten ne eivät palaudu oikealla tavalla sovelluksen kaatumisen jäljiltä ja näin tietoturvaan voi jäädä aukkoja.

Seuraavassa taulukossa on esitelty suorituskykytestauksen muodoittain ne riskit ja riskitekijät, joita kuvatulla testillä pystytään ja pyritään selvittämään.

*Taulukko 1. Suorituskykytestauksella etsittävät riskitekijät [1, s. 22-23.]*

| **Suorituskykytestin alalaji** | **selvitettävät riskit ja riskitekijät** |
|---|---|
| Kapasiteettitestaus | -Vastaako sovelluksen kapasiteetti sovellukselle annettuja raja-arvoja?<br>-Kuinka sovellus toimii kun kapasiteetti on ylä-rajoilla? |
| Komponenttitestaus | -Vastaako komponentti odotuksia?<br>-Onko komponentti oikein optimoitu?<br>-Onko huono suorituskyky juuri tämän komponentin syytä? |
| Kestävyystestaus | -Jatkuuko suorituskyky samana ajan kuluessa?<br>-Onko sovelluksessa sellaisia ongelmia, joita lyhyemmillä testeillä ei ole saatu ilmi?<br>-Onko sovelluksessa sisäisiä häiriöitä, joita ei ole aiemmin otettu huomioon? |
| Kuormitustestaus | -Kuinka monta yht'aikasta käyttäjää sovellus pystyy hallitsemaan?<br>-Kuinka paljon dataa palvelin / tietokanta pystyy käsittelemään?<br>-Onko verkon osat oikein mitoitettuja |
| Savutestaus | -Onko sovellus valmis suorituskykytestaukseen?<br>-Minkä tyyppisiä suorituskykytestejä sovellus tarvitsee?<br>-Onko tämä sovelluksen versio parempi kuin edeltäjänsä? |
| Stressitestaus | -Mitä sovellukselle tapahtuu, kun raja-arvot ylittyvät?<br>-Minkälaisiin virheisiin tulee varautua?<br>-Tuleeko kehitellä jonkinlaisia ilmaisimia estämään tulevat virheet? |

Sovelluksen suorituskykyyn liittyy useita riskejä tai riskitekijöitä. Oikein toteutetulla suorituskykytestauksella saadaan lähes kaikki sovellukseen liittyvät riskitekijät selville ja korjattua ne ennen sovelluksen julkaisua.

### 2.3 Suorituskykytestauksessa käytettäviä sovelluksia

Suorituskykytestauksen apuna käytetään usein apuvälineitä. Nämä apuvälineet ovat usein sovelluksia, joilla voi nauhoittaa testattavan sovelluksen käyttöä ja tiettyjä toimintoja, jonka jälkeen käytettävä apuväline kirjoittaa toiminnosta skriptin. Skriptillä tarkoitetaan virtuaalikäyttäjän sovelluksessa tekemiä toimintoja. Myöhemmin tässä työssä puhutaan virtuaalikäyttäjistä. Sitten itse ajoja ajaessa skriptejä voidaan laittaa ajoon useita kerrallaan ja niille voi antaa jokaiselle eri toimintakäskyt.

Suorituskykytestaukseen on saatavilla useita  apuvälineitä ja sovelluksia. Niin sanottuja ’open source’ –sovelluksia suorituskykytestaukseen on viimeaikoina tullut entistä enemmän markkinoille. Nämä ovat ilmaisohjelmia, joita voi ladata internetistä. Tällaisia sovelluksia ovat muun muassa Hammerhead, Pylot, vPerformer ja Curl-Loader.

Saatavilla on myös lisenssipohjaisia, eli maksullisia suorituskykytestaus–työkaluja. Tällaisia sovelluksia ovat muun muassa Mercuryn LoadRunner ja Compuwaren QALoad. Tässä työssä tutustuttiin tarkemmin Mercuryn (nykyisin HP:n) LoadRunner –työkaluun. Tämän sovellus valittiin, koska Logicalla oli käytössä tämä sovellus ja sitä on käytetty myös sovellukselle toteutetuissa aikaisemmissa suorituskykytesteissä.

Lisää LoadRunner-testityökalusta on luvassa luvussa 4.

# 3 WEB-SOVELLUKSEN SUORITUSKYKYTESTAUS

Tehdessä suorituskykytestejä web-sovellukselle, on hyvä noudattaa seuraavaa asialistaa, jotta kaikki asiat tulisi varmasti mukaan testiin. [1, s. 32-42.]

- Tutkitaan ja määritellään testattavan sovelluksen testiympäristö.
- Määritellään sovelluksen suorituskyvylle asetetut raja-arvot ja kriteerit.
- Hahmotellaan ja suunnitellaan tulevat testit.
- Konfiguroidaan testiympäristö.
- Toteutetaan testisuunnitelma, eli nauhoitetaan skriptit.
- Toteutetaan testit.
- Analysoidaan ja raportoidaan testitulokset ja mahdollisesti testataan uudelleen.

Näin toimittaessa suorituskykytestauksessa tulee otettua kaikki tarvittava huomioon, niin ennen kuin jälkeenkin testien. On nimittäin harmillista, kun testaaja on aloittamassa testejä ja sitten joku pieni yksityiskohta onkin jäänyt huomaamatta, niin testejä pitää pahimmassa tapauksessa siirtää jopa päiviä eteenpäin.

Seuraavissa luvuissa käsitellään jokaista yllämainittua kohtaa tarkemmin ja kerrotaan, mitä toimenpiteitä ne sisältävät.

## 3.1 Testattavan sovelluksen testiympäristön tutkiminen

Ensimmäiseksi suorituskykytestauksessa tulee tutkia ja hahmotella testattavan sovelluksen fyysinen testiympäristö. Fyysisellä testiympäristöllä tarkoitetaan laitteistoa, sovelluksia ja verkon asetuksia. Testiympäristön hahmottelu ja tutkiminen on tärkeää, koska jokainen testiympäristö eroaa toisistaan tavalla tai toisella.

Testattavan sovelluksen testiympäristön tulee muistuttaa sovelluksen tulevaa tuotantoympäristöä, sillä muuten suorituskykytestien suorittaminen voi johtaa vääriin testituloksiin. Sovellus voi toimia testiympäristössä moitteettomasti, mutta tuotannossa voi esiintyä odottamattomia ongelmia, jos ympäristö onkin erilainen.

Suorituskykytestejä tehdessä on myös sovelluksen tietokantaan kiinnitettävä erityistä huomiota. Yleensä sovelluksen suorituskyky laskee, kun tietokanta

suurenee, mutta sovelluksen optimointi-tekniikoilla voidaan parantaa suorituskykyä huomattavasti. Testattavan sovelluksen tietokanta tulisi olla samankaltainen tuotannossa olevan tietokannan kanssa. Näin saadaan varmuus siitä, että testattava sovellus pystyy käsittelemään sen kohtaaman kuorman realistisessa tuotantoympäristössä. Testattaessa realistisilla tietokannoilla voidaan varmistua muun muassa siitä, että kovalevytila riittää, prosessorin teho riittää ja haut tietokannasta eivät kestä liian kauan. [8, s. 217-218.]

Seuraavassa on muutamia asioita, joita testaajan tulee kiinnittää erityistä huomiota testiympäristöä tutkiessa ja sitä hahmottaessa.

- Laitteistoon ja sen asetuksiin tulee kiinnittää erityistä huomiota. Myös laitteiston fyysisiin ominaisuuksiin, eli muistiin, prosessoriin ja kovalevyyn tulee kiinnittää huomiota testiympäristöä tutkiessa.
- Verkon asetukset ovat tärkeitä. Verkon arkkitehtuurin ja loppukäyttäjän sijainnin tunteminen on arvokasta. Myös klusterin ja DNS:n asetukset pitää tutkia. On myös tiedettävä miten kuorma tasaantuu verkon eri osissa.
- Suorituskykytestaustyökalun rajoitukset (esim. lisenssin tiedot, kuinka monta virtuaalikäyttäjää, yms.) on hyvä tuntea. Myös muiden valvontatyökalujen käyttöä tulee miettiä, sillä esimerkiksi muistin käyttäytymistä ei monikaan suoritustestaustyökalu pysty käsittelemään.
- Testiympäristön muiden sovelluksien toiminta, eli jotka on asennettu tai toimivat jaetussa tai virtuaalisessa ympäristössä, on tunnettava. Sovelluksien lisenssit ja niiden rajoitukset tulee tietää ja tunnistaa.
- Verkon kuormitus tai ylimääräinen liikenne verkossa on huomioitava. Kannan varmuuskopionti-aikataulu ja sovelluksien päivitykset on huomioitava kuten myös vuorovaikutus muiden laitteistojen kanssa.

Kun testaajalla on ymmärrys testattavasta testiympäristöstä, testauksen suunnittelu ja testien toteuttaminen on paljon helpompaa. Testeistä tulee myös tehokkaampia, kun testaaja tuntee testattavan ympäristön.

Joissain tapauksissa testiympäristöön tulee tutustua jaksoittain koko sovelluksen eliniän ajan, sillä usein sovelluksen kehityksen aikana testiympäristöä päivitetään uusin sovelluksin ja laittein.

### 3.2 Testattavan sovelluksen suorituskyvylle asetetut kriteerit

Kun sovelluksen testiympäristö on testaajalle tuttu, testaajan kannattaa tutustua sovellukseen ja sille asetettuihin suorituskyvyn kriteereihin ja raja-arvoihin. Nämä kriteerit ja raja-arvot kannattaa määritellä, tai ainakin karkeasti ideoida jo sovelluksen suunnitteluvaiheessa. Jos kriteerejä aletaan määritellä liian myöhään, voi koko sovelluksen määrittelyn joutua päivittämään.

Web-sovelluksen suorituskyvylle määritellään yleensä raja-arvoja vasteaikoihin, kapasiteettiin ja laitteistojen suorituskyvyn hyväksikäyttöön. Näin sovelluksen kehittäjät pystyvät tekemään sovelluksesta riittävän vakaan, nopean ja suorituskykyisen heti alusta alkaen. Jos suorituskyvyn nostoon tulee äkillinen tarve sovelluksen kehityksen loppuvaiheilla, voi koko sovelluksen arkkitehtuuria joutua muuttamaan. Tämä tuo turhia lisäkustannuksia, jotka voidaan välttää jo suunnitteluvaiheessa.

Suorituskykytestaajan tulee kiinnittää suorituskykykriteerejä hahmottaessa erityistä huomiota

- sovelluksen tuotannossa kohtaamiin vaatimuksiin
- käyttäjän odotuksiin
- sopimuksellisiin velvoitteisiin
- ohjeelliseen joustamiseen ja talouden vaatimuksiin
- asiakkaan haluamiin vaste-aikoihin
- tärkeisiin suorituskyvyn-ilmaisimiin
- mahdollisien kilpailijoiden vastaaviin sovelluksiin
- sovelluksen tietoturvaan, kasvunvaraan ja vakauteen
- aikatauluun, budjettiin, voimavaroihin ja muihin prioriteetteihin.

Niin kuin sovelluksen testiympäristön tunteminen, myös sovelluksen suorituskyvylle asetettujen kriteerien tunteminen tekee testien suunnittelusta ja toteuttamisesta helpompaa ja varmempaa.

### 3.3 Testien hahmottelu ja suunnittelu

Testejä hahmotellessa ja suunniteltaessa testaajan täytyy selvittää testattavasta sovelluksesta monia tärkeitä asioita. Hänen täytyy selvittää sovelluksen tärkeimmät ja käytetyimmät toiminnot. Myös sovelluksen käyttäjien mahdollisen eroavaisuudet tulee selvittää. Testaajan täytyy määritellä ja luoda testeissä käytettävä testidata ja eritellä ne kohdat sovelluksesta, joista halutaan mitata esimerkiksi vasteaikoja.

Testaajan tulee selvittää loppukäyttäjien käyttäytyminen mahdollisimman hyvin, sillä testaajan tavoitteena on saada aikaan testi, joka on mahdollisimman lähellä sovelluksen tuotannossa kohtaamaa käyttöä ja rasitusta. Mitä lähempänä testi on sovelluksen aitoa käyttöä, sitä enemmän testauksesta saaduilla tuloksilla on käyttöarvoa ja merkitystä.

Realististen testien tulee sisältää käyttäjän pitämät ajattelutauot (engl. think time), kun halutaan aitoa ympäristöä ja reaalimaailman käyttöä muistuttavat testit. Testien tulee sisältää myös käyttäjien sovelluksessa mahdollisesti tekemiä virheitä ja mahdollisia käyttäjien tekemiä keskeytyksiä.

Testaajan tulee testejä suunniteltaessa keskittyä sovelluksen käyttöön tuotannossa ja sovelluksen toimintaan vaikuttaviin ulkoisiin tekijöihin. Tällaisia ulkoisia tekijöitä ovat muun muassa muut laitteistot ja järjestelmät, jotka toimivat testatun järjestelmän kanssa samassa ympäristössä.

Testaajan täytyy jo suunnitteluvaiheessa miettiä tai päättää, millä sovelluksilla testit toteutetaan ja miten testiskriptit tullaan nauhoittamaan.

Testejä suunniteltaessa on tunnettava testattavan sovelluksen toiminta kuin myös sovelluksen käyttäjien toiminta. Käyttäjien toiminta selvitetään yleensä yhdessä sovelluksen tilanneen asiakkaan kanssa. Hyvin suunnitellut testit ovat tärkeässä osassa suorituskykytestejä tehdessä.

### 3.4 Testattavan sovelluksen testiympäristön konfigurointi

Testiympäristö tulee konfiguroida ennen suorituskykytestauksen aloittamista ja samalla tulee varmistaa, että kaikki tarvittavat laitteistot, testattavan sovelluksen osat ja verkko ovat toiminnassa ja yhteyksissä toisiinsa. Konfiguroinnin tekee yleensä sovelluksen toteuttajat.

Suorituskykytestaus-sovellus tai monitorointi on yleensä vaivalloisia saada käyntiin ja toimimaan moitteettomasti, koska testiin vaikuttavia tekijöitä on paljon. Tällaisia ovat muun muassa verkon toiminta, laitteiston toiminta, IP-osoitteiden kontrollointi, testeissä käytettävien sovelluksien toiminta keskenään. Ongelmia voi myös tulla yllättävistä ja odottamattomistakin suunnista. Tällaisia ongelmia ovat esimerkiksi verkon ongelmat tai palvelimen väärin konfiguroidut asetukset.

Ennen suorituskykytestien aloittamista kannattaa pyytää sovelluksen testaajia todentamaan sovelluksen toiminta. Näin säästetään aikaa ja vaivaa, jos sovellus ei jostain syystä toimisikaan halutulla tavalla.

### 3.5 Testisuunnitelman muuttaminen testiskripteiksi

Testisuunnitelman muuttaminen testiskripteiksi riippuu käytettävästä suorituskykytestaus-työkalusta. Yleensä se tarkoittaa sovelluksen yhden toiminnon nauhoittamista ja sen jälkeen niiden yhdistelemistä kokonaisuudeksi, joka sitten mallintaa sovelluksen realistista käyttöä ja kuormitusta.

Suorituskykytestaus-työkalut laahaavat väistämättä teknologian ja kehityksen perässä. Työkalun toimittajat pystyvät ainoastaan kehittelemään testaustyökalunsa niin, että se tukee vain yleisimpiä ja tunnetuimpia teknologioita ja silloinkin testaustyökalut ovat jo vanhentuneet, kun ne saavat työkalunsa tukemaan jotain uutta teknologiaa.

Suorituskykytestauksen suurin haaste on saada keinotekoiset testit vastaamaan sovelluksen reaalimaailmassa kohtaamaa kuormitusta. Tätä kannattaa suunnitella tarkasti eikä kannata yllättyä, vaikka se ei ihan heti onnistuisikaan.

Myös testidatan löytyminen testattavan sovelluksen tietokannoista on tärkeää, sillä ilman oikeaa testidataa koko suorituskykytestaus epäonnistuu ja voi antaa vääriä tuloksia ja päätelmiä sovelluksen suorituskyvystä.

### 3.6 Testien toteuttaminen

Testien toteuttaminen on juuri se, mitä monet pitävät suorituskykytestauksena. Monet suorituskykytestaus-koulutukset käsittelevät testien käynnistämisestä ja ajosta ainoastaan sen, että testiajot lähtevät liikkeelle ja pyörivät odotetusti. Oikeassa maailmassa testien käynnistäminen ja ajo ovat paljon muutakin kuin testin käynnistäminen ja ajon seurailu.

Alla muutamia ohjeita testaajalle testiajon käynnistämisestä ja itse ajosta.

- Seuraa ajoa ja monitoroi tärkeitä mitattavia parametrejä
- Validoi testit ja asetukset, sekä varmista testidatan olemassa olo testattavassa sovelluksessa
- Muista testinaikainen datan, virtuaalikäyttäjien (engl. virtual user), laitteiston ja itse ajon seuraaminen
- Testin päättyessä, tarkista nopeasti onko testi ja saadut tulokset analysointikelpoisia
- Tallenna ajettujen testit, testidata, tulokset ja muut tarpeelliset tiedot paikkaan, josta ne on helppo löytää tulevaan käyttöön
- Ota testin alku-ja loppuaika muistiin, nimeä tulokset selkeästi ja niin edelleen. Tämä helpottaa tulevia uudelleentestauksia ja edellisiin tuloksiin vertailua.

Kun testaaja on aloittamassa suorituskykytestit, niin hänen kannattaa tarkistaa vielä, että [1, s. 39.]

- testiympäristön asetukset ovat samat, mihin testit suunniteltin ja nauhoitettiin.
- testattava sovellus ja testiympäristön asetukset ovat yhteensopivia testidatan kanssa.
- savutesti on ajettu ja testattava sovellus toimii normaalisti.
- testattava sovellus on käynnistetty uudelleen (jos nauhoitetut skriptit eivät estä sen tekemistä).
- nauhoitetut skriptit mallintavat sovelluksen realistista toimintaa mahdollisimman hyvin.

- skripteistä saadaan esiin juuri oikeat kohdat, joista halutaan tarkempia tietoja ja analyyseja.

Kun kaikki edellä olevat asiat on tarkistettu ja tehty, niin testaaja voi aloittaa suorituskykytestauksen, eli käynnistää suorituskykytestaus-työkalun tai muun sovelluksen.

### 3.7 Tulosten analysointi, raportointi ja uudelleentestaus

Sovelluksen tilanneen yrityksen esimiehet ja sidosryhmät tarvitsevat suoritetuista suoritukykytesteistä muutakin kuin pelkät testitulokset. He tarvitsevat johtopäätöksiä, joista selviää jatkotoimenpiteet . He tarvitsevat myös selvityksiä, jotka tukevat noita johtopäätöksiä.

Teknisen ryhmän henkilöt tarvitsevat analyyseja, vertauksia edellisiin testeihin ja tietoa siitä, miten saadut tulokset on saavutettu.

Ennenkuin testitulokset voidaan raportoida, ne pitää analysoida huolellisesti. Testituloksia analysoitaessa testaajan kannattaa ottaa huomioon, että hän analysoi saadut tulokset niin erikseen kuin yhdessä teknisen tiimin kanssa. Hänen täytyy myös verrata niitä sovellukselle määriteltyihin suorituskykykriteereihin ja raja-arvoihin. Testin jälkeen testaajan tulee myös selvittää, onko testattu sovellus toiminut hyväksyttävällä tavalla ja jos ei ole, niin hänen täytyy tutkia epäonnistumisen syyt ja mahdollisuuksien mukaan korjata ne. Mahdollisien pullonkaulojen löytyessä ne korjataan ja sen jälkeen testit uusitaan, jotta saadaan varmuus korjauksen onnistumisesta.

Toteuttajat saavat suorituskykytestauksesta saaduista tuloksista usein tärkeää tietoa sovelluksen eri osista. Testitulokset täytyy toimittaa koko ryhmän käyttöön heti kun mahdollista, jotta mahdolliset korjaukset saadaan aloitettua heti. Jos testituloksista ei saada selville sitä, mitä testeillä haettiin, niin testaajan tulee modifioida testejä ja ajaa testit uudestaan.

Jos tehdyt testit ovat tehneet kantaan liikaa dataa, voidaan kannasta poistaa testeissä syntynyt data ennen uudelleentestausta. Kannattaa olla kuitenkin varovainen, että poistetaan vain sellainen data, jota ei tarvita.

Kun testitulokset on analysoitu, voidaan niistä tehdä testitulosraportit. Raportteja on yleensä kahdenlaisia, on omalle tiimille annettava tekninen-raportti ja asiakkaalle ja esimiehille annettava sidosryhmäraportti.

Tekninenraportti sisältää yleensä [1, s. 41.]

- selvityksen ajetusta testistä, mukaanlukien kuormitusmallin ja testiympäristön
- helposti luettavaa ja minimaalisesti käsiteltyä tietoa
- pääsyn tai linkin perusteellisiin testituloksiin ja testiolosuhteisiin sekä
- lyhyen selostuksen havainnoista, huolista, kysymyksistä ja anomuksista mahdollisesta yhteistyöstä.

Sidosryhmäraportti sisältää

- sovellukselle asetetut suorituskykykriteerit, joita vastaan tulokset ovat analysoitu
- kaikkein tärkeimpien tulosten visuaalinen esityksen (esimerkiksi grafiikoin ja taulukoin)
- lyhyen sanallisen referaatin siitä, kuinka sovellus toimi kriteereihin nähden
- kuormitusmallin ja testiympäristön visuaalisen havainnollistamisen.
- pääsyn tai linkin tekniseen raporttiin, perusteellisiin testituloksiin ja testiolosuhteisiin
- lyhyen selostuksen havainnoista, huolista ja suosituksista.

Suorituskykytestauksen tuloksia analysoitaessa ja raportoidessa testaajan kannattaa siis muistaa, että hän [1, s. 41-42.]

- raportoi aikaisin, raportoi usein
- raportoi visuaalisesti
- raportoi oivaltavasti
- käyttää oikeita testituloksia
- kustomoi raportteja läsnäolijoiden mukaan
- laittaa testitulokset kaikkien saataville, myös asiakkaan sekä
- poistaa testituloksista turhat tulokset ja tiedot.

Kun luvun 3 asiat omaksuu, ymmärtää niiden tarkoituksen ja tekee ne vielä oikeassa järjestyksessä, niin voidaan sanoa, että tietää jo paljon suorituskykytestauksesta.

# 4 SKRIPTIEN NAUHOITUS WEB-SOVELLUKSELLE

Suorituskykytesteissä muodostetaan testattavalle sovellukselle niin aitoa tilannetta muistuttava kuormitus kuin mahdollista. Tätä varten on nauhoitettava skriptejä, joilla reaalimaailman tilannetta voidaan mukailla.

Seuraavissa luvuissa kerrotaan tarkemmin käytetystä LoadRunner-testityökalusta ja kuinka LoadRunner-työkalulla nauhoitetaan skriptejä tulevia testejä varten.

## 4.1 Testityökalu

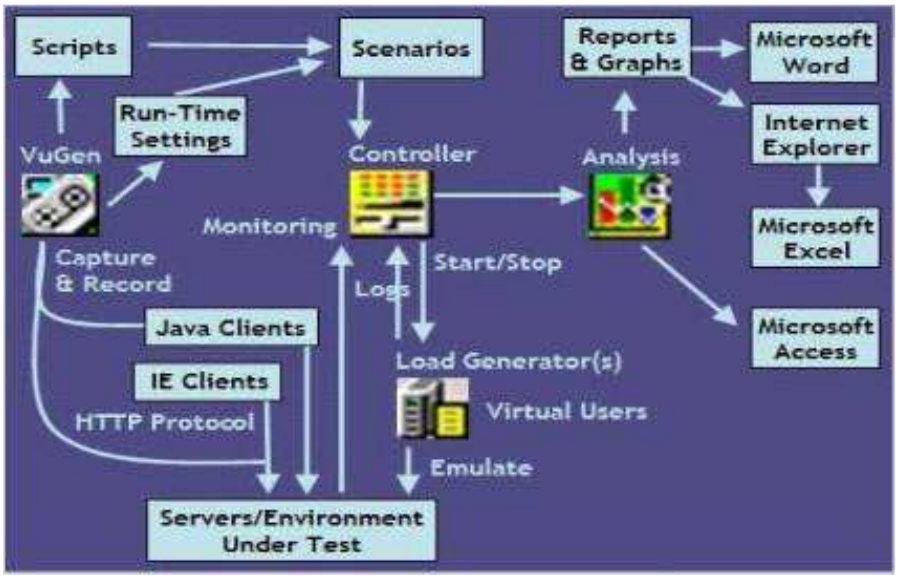
Testityökalua valittaessa tutkittiin kahta eri suorituskykytestaussovellusta, Mercuryn (nykyisin HP:n) LoadRunneria ja Compuwaren QALoadia. Molemmista sovelluksista oli kokemuksia jo edellisistä testeistä. Valinnassa päädyttiin LoadRunneriin, koska siitä oli hyviä kokemuksia jo aikaisemmista testeistä.

LoadRunner 8.0 on suorituskykytestaukseen tarkoitettu työkalu, jolla voidaan selvittää testattavan sovelluksen käyttäytyminen ja suorituskyky. Sovelluksella voi ajaa haluttu määrä kuormaa testattavaa sovellusta vastaan ja näin simuloida aitoa tuotantoympäristöä muistuttavaa tilannetta. LoadRunnerilla voi testata usealla eri protokollalla tehtyjä sovelluksia, mutta LoadRunneria käytettiin tässä työssä ainoastaan web-sovelluksen testaukseen.

LoadRunner pystyy jäljittelemään tuhansien käyttäjien (riippuu lisenssistä, Logican lisenssi kattoi 100 yht’aikaista käyttäjää) aiheuttamaa kuormaa testattavalle sovellukselle. LoadRunner pystyy tarkkailemaan ja nauhoittamaan testattavan sovelluksen vasteaikoja ja näin sillä saa tärkeää tietoa testattavasta sovelluksesta.

LoadRunnerilla pystytään simuloimaan tilanteita, joita sovellus tulee tuotannossa kohtaamaan. LoadRunnerilla pystyy simuloimaan ruuhka-aikoja, joita syntyy tiettyinä kellonaikoina, esimerkiksi tuntikirjaus-sovelluksen käyttäjämääriin tulevia piikkejä aamulla klo 8:00-9:00 ja iltapäivällä klo 16:00-17:00.

LoadRunnerissa on kolme työkalua. Virtual User Generator- (VuGen), Controller- ja Analysis- työkalut. Seuraavassa esitellään pääpiirteittäin näiden kolmen työkalun käyttöä ja toimintaa.




*Kuva 2. LoadRunner 8.0 toimintakaavio [2]*

Kuvassa 2 on esitetty LoadRunnerin toimintakaavio. Skriptit nauhoitetaan kuvan vasemmassa reunassa näkyvässä VuGen-työkalulla. Kun skriptit on nauhoitettu, niistä muodostetaan Controllerissa skenaario. Skenaariossa määritellään haluttu määrä virtuaalikäyttäjiä, testien pituus ja kuinka virtuaalikäyttäjien tulisi testien aikana käyttäytyä. Kun testit on saatu päätökseen, niin tulokset voi analysoida Analysis-työkalulla. Siitä raportit voi tallentaa HTML,- tai tekstimuotoon, esimerkiksi Microsoft Wordille.

*Virtual User Generator*

Virtual User Generator -työkalulla voi nauhoittaa testissä ajettavat skriptit. Skriptistä käytetään sanontaa virtuaalikäyttäjä. Tällaista sanontaa käytetään, koska skriptillä mallinnetaan sovelluksen käyttäjän toimintaa virtuaalisesti. Virtual User Generator kääntää nauhoitetun skriptin suoraan C-kielelle, joten käytännössä koodauskieliä osaamatonkin testaaja osaa ja pystyy käyttämään LoadRunner-sovellusta. Skriptiä voi myöhemmin muokata esimerkiksi parametrisoimalla hakuehdot tai tallennettavat tiedot. Näin saadaan useita reaalimailmaa vastaavia tilanteita yhdellä skriptillä.

*Controller*

Itse testit ajetaan Controller-työkalulla. Controllerissa voi valita halutun määrän virtuaalikäyttäjiä. Jokaiselle virtuaalikäyttäjälle voi määrittää kuinka monta Iteraatiota jokainen kerrallaan tekee. Jos laittaa virtuaalikäyttäjän tekemään esimerkiksi 4 iteraatiota, niin se toistaa nauhoitetun skriptin neljä kertaa. Näin voi pienemmälläkin lisenssillä saada aikaan tarpeeksi suuren ja halutun kuorman.

Controllerissa voi määrittää halutun kuorman lisäämällä tai poistamalla käyttäjämääriä, eli kontrolloimalla virtuaalikäyttäjien määrää. Controllerissa voi myös ajoittaa testinsä haluamallaan tavalla esimerkiksi syöttämällä kaksi käyttäjää aina 30 sekunnin välein niin kauan, kun käyttäjämäärä saavuttaa halutun tason. Testin loppua voi myös generoida halutulla tavalla, esimerkiksi lopettaa kaikki virtuaalikäyttäjät tiettynä kellonaikana tai sitten tiputtaa niitä 30 sekunnin välein niin kauan, kun virtuaalikäyttäjiä ei enää ole testattavassa sovelluksessa. Myös testin pituuden voi määrittää ja testiä voidaan myös generoida ajon aikana, esimerkiksi lisäämällä tai poistamalla käyttäjiä manuaalisesti.

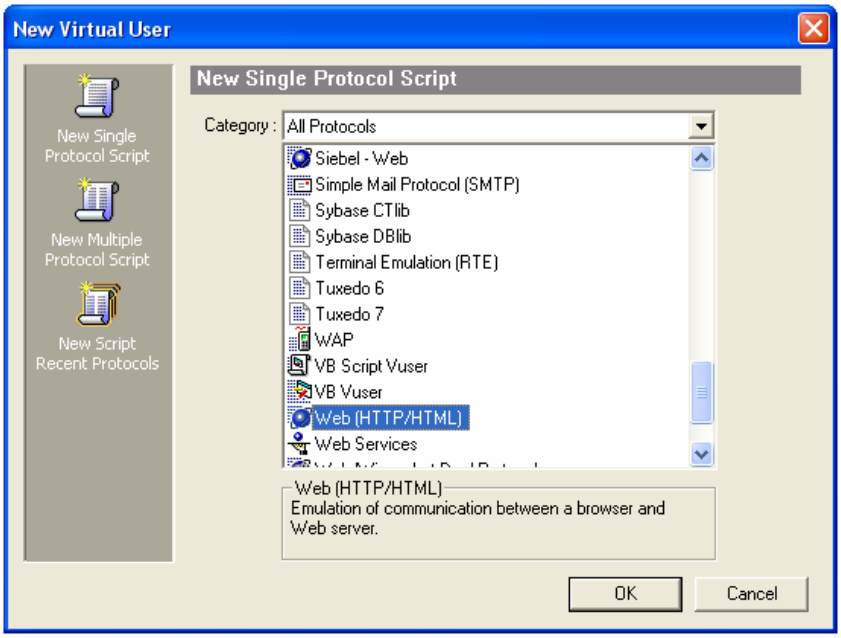
*Analysis*

Ajetut testit analysoidaan Analysis-työkalulla. Analysis-työkalulla saadaan ajetusta testistä kaikki tiedot, taulukot, käyrät ja grafiikat. Saatuja grafiikoita ja käyriä voi yhdistää ja muokata haluamallaan tavalla. Analysis-työkalulla saa samaan grafiikkaan esimerkiksi se kuinka monta käyttäjää on testissä ollut ja mikä on testattavan sovelluksen maksimikapasiteetti.

Yleensä grafiikkaesitykseen valitaan vain yksi näytettävä käyrä. Jos kuvaan lisää toisen käyrän, kuva skaalautuu usein väärin ja antaa käyttäjälle väärää informaatiota sisältävän grafiikan. Jos halutaan tutkia jotain tiettyä kohtaa testistä, esimerkiksi ruuhka-aikaa, niin graafiset esitykset voi muokata näyttämään vain tiettyä ajan hetkeä testistä. Kun halutut analyysit ja graafiset esitykset on saatu valmiiksi, niin Analysis-työkalulla saa tehtyä raportit niin HTML- kuin tekstimuodossa.

## 4.2 Yleiset ohjeet skriptien nauhoitukseen LoadRunnerilla

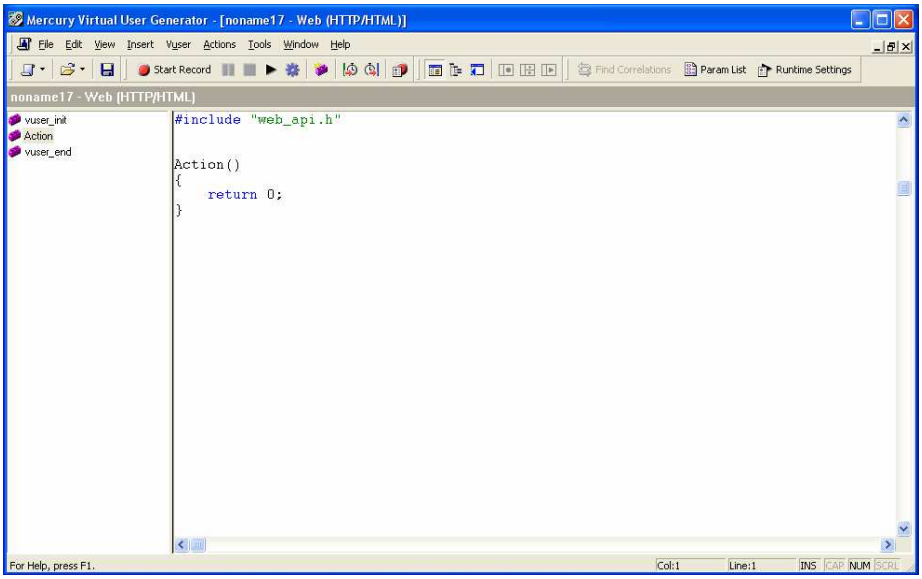
Skriptit nauhoitetaan LoadRunnerin Virtual User Generatorilla. Sovelluksen avautuessa sovellus kysyy käyttäjältä, mitä protokollaa nauhoituksissa on tarkoitus nauhoittaa.




*Kuva 3. Nauhoitettavan protokollan valinta*

Kun sovelluksen avaa, avautuu samalla kuvassa 3 näkyvä ikkuna. Tästä ikkunasta valitaan ’Web(HTTP/HTML)’ ja siirrytään itse ohjelmaan valitsemalla ’OK’.

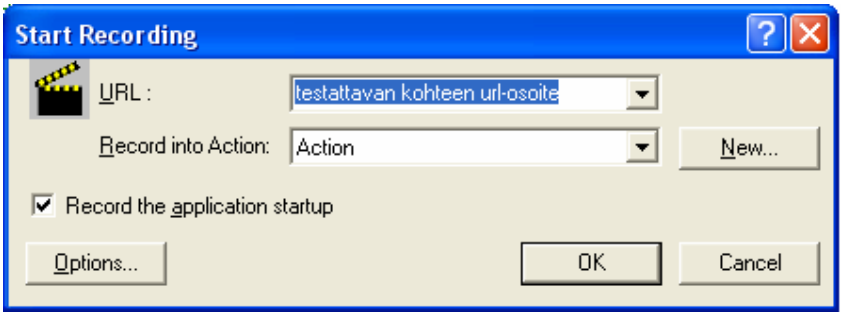
Load Runnerilla voi myös nauhoittaa skriptejä moneen muuhunkin protokollaan, mutta tässä työssä sitä käytetään ainoastaan Web(HTTP/HTML)-protokollaan.




*Kuva 4. Virtual User Generator –etusivu*

Protokollan valinnan jälkeen avautuu kuvassa 4 näkyvä Virtual User Generatorin etusivu. Tämä on sivu, johon nauhoitettu toiminto käännetään C-kieliseksi skriptiksi. Nauhoitettua skriptiä voi muokkailla halutulla tavalla. Esimerkiksi parametrisoimalla muuttujat saadaan LoadRunnerin kuormitusta levitettyä useammille tapahtumille tai hakuehdoille.

Skriptin nauhoitus aloitetaan etusivun ylätyökalupalkista löytyvästä ’Start record’ -painikkeesta.




*Kuva 5. Start Recording –näyttö*

Tämän jälkeen LoadRunner avaa kuvassa 5 näkyvän ikkunan, johon syötetään testattavan sovelluksen URL-osoite. ’Record into action:’-kohtaan jätetään oletuksena oleva ’Action’. Myös näytön alaosan ’Record the application startup’ -valinta jätetään ruutuun. Skriptiä siirrytään nauhoittamaan valitsemalla ’OK’.

LoadRunner avaa testattavan web-sovelluksen ja sen päällä on kuvassa 6 näkyvä nauhoitus-työkalupalkki.


*Kuva 6. Recording...-näyttö*

Kun skriptejä nauhoitetaan, kannattaa käyttää kuvassa 6 näkyviä alku- ja loppumerkkejä (kuvassa olevat kellot), jotta skriptiin saa erilaisia osia. Nämä kelloilla erotellut osat näkyvät myöhemmin testiä ajaessa Controller-työkalussa omina käyrinään ja niitä pystytään analysoimaan helpommin ja tarkemmin. Alku- ja loppumerkit voi lisätä skriptiin myöhemminkin, mutta kun ne lisää jo nauhoitusvaiheessa, niin sitä ei tarvitse enää erikseen tehdä.

Kun on päästy sisään testattavaan sovellukseen, niin halutut toiminnot nauhoitetaan sovelluksessa etukäteen suunnitellussa järjestyksessä. Kaikki toiminnot, joita halutaan tarkkailla ja joista halutaan tarkempia tuloksia, erotellaan alku- ja loppumerkeillä. Tällaisia ovat tässä testissä esimerkiksi erilaisten tietojen tallennukset, haut ja päätöksien muodostus. Näin kaikista tärkeistä toiminnoista saadaan tarkat ajat ja tiedot analysoitaessa testiä.

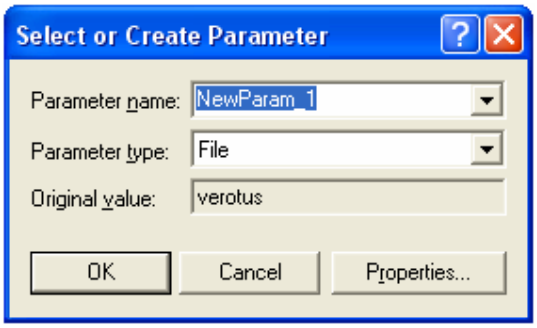
Kun kaikki tarvittava on nauhoitettu, painetaan kuvassa 6 näkyvää ’Stop’-painiketta ja palataan takaisin kuvassa 4 näkyvään Virtual User Generatoriin.

Nauhoitettu skripti näkyy nyt Virtual User Generatorin ’Action’-välilehdellä C-kielisenä skriptinä. Skriptistä kannattaa leikata mahdollinen sisäänkirjautuminen-osio ja siirtää se vuser_init osioon. Samoin menetellään uloskirjautumis-osion kanssa, mutta se siirretään vuser_end osioon. Näin tehdessä virtuaalikäyttäjä kirjautuu ainoastaan kerran sisään ja ulos koko testin aikana. Näin ei kannata toimia silloin, jos halutaan juuri tarkkailla ulos- ja sisäänkirjautumisen toimintaa.

Skriptien nauhoituksessa syötetyt muuttujat kannattaa parametrisoida. Tällaisia ovat esimerkiksi hakuehdot, mahdolliset syötettävät perus- ja lisätiedot ja kaikki muu, mitä skriptien nauhoituksen aikana joutuu kirjoittamaan tai valitsemaan hiirellä. Parametrisoinnilla estetään se, että LoadRunner ei joka kerta syöttäisi samoja hakuehtoja ja samoja tietoja samoihin paikkoihin,

vaan muuttaa niitä joka kerralla erilaisiksi. Näin testeistä saadaan enemmän todenmukaisia ja kuormitus ei kohdistu aina samoille tapahtumille.

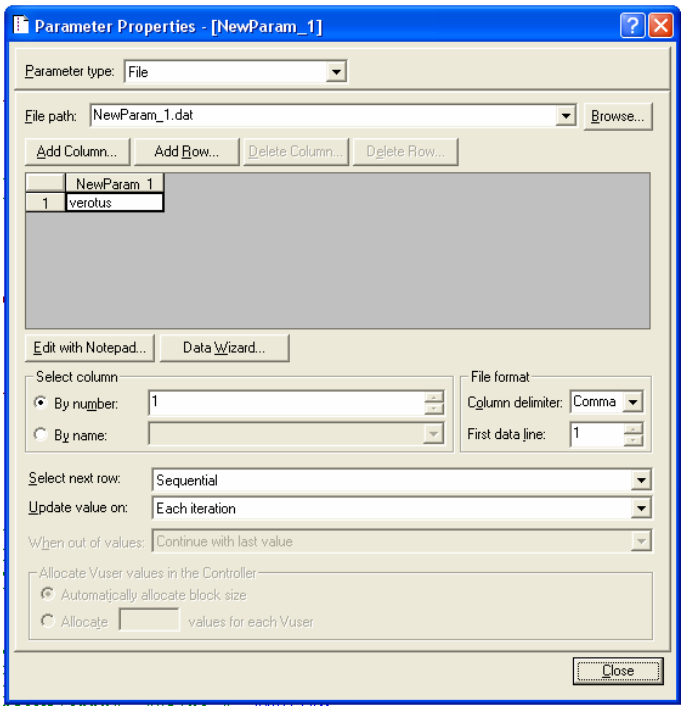
Parametrisointi tapahtuu maalaamalla haluttu muuttuja. Seuraavaksi klikataan hiiren oikealla näppäimellä halutun muuttujan päältä ja valitaan 'Replace with a parameter'.




*Kuva 7. Select or create parameter*

Uudelle parametrille annetaan nimi kuvassa 7 näkyvällä ikkunalla. Parametrit kannattaa nimetä selkeästi, niin niitä on helpompi käsitellä myöhemmässä vaiheessa. Aina käsittelyyn ei ole tarvetta, mutta esimerkiksi uudelleen testauksessa niitä voi joutua muokkaamaan, jos tulos tai rasitus ei ole ollut halutun kaltaista.

Kun parametrisointi on valmis, klikataan kuvassa 7 näkyvällä ikkunalla 'OK' -painiketta ja siirrytään kuvassa 8 näkyvälle 'Parameter Properties' –ikkunalle. 'Parameter Properties' –ikkunalla voi määrittää parametrien yksityiskohdat ja käsittelysäännöt.




*Kuva 8. Parameter Properties –näyttö*

Kuvassa 8 näkyvässä ’Edit with a notepad’ –kohdassa voidaan notepad-ohjelman avulla helposti lisätä ja muokata parametreja. Parametrien luonti onnistuu vaivattomasti ja nopeasti ’Data Wizard’ –ohjelmalla. ’Select next row’- ja ’Update value on’ –kohdissa voi määritellä, milloin ja miten parametria vaihdetaan ja luetaan.

Kun parametrisointi ja skriptin muokkaus on valmista, niin skripti tallennetaan halutulla ja helposti tunnistettavalla nimellä. Tämä tallennus tapahtuu kuvassa 4 löytyvältä Virtual User Generator-työkalun –etusivulta komentosarjalla File -> Save As..

Skriptejä nauhoitetaan haluttu määrä, ja ne kannattaa kaikki tallentaa samaan paikkaan, jotta ne on myöhemmin helppo lisätä Controllerissa itse testiskenaarioon.

# 5 ERÄÄN WEB-POHJAISEN JÄRJESTELMÄN SUORITUSKYKYTESTAUS

Tässä luvussa kerrotaan erään web-pohjaisen järjestelmän suorituskykytestauksesta LoadRunnerilla. Luvussa kerrotaan tarkemmin testien suunnittelusta, valmistelusta ja testien toteuttamisesta.

## 5.1 Testattava sovellus

Testattava sovellus on web-pohjainen järjestelmä, jonka asiakas on tilannut Logica Suomi Oy:ltä. Sovellus on ollut tuotannossa jo muutaman vuoden ja on toiminut hienosti. Suorituskykytesteille tuli nyt tarvetta, koska sovelluksesta oli tulossa uusi versio. Piti siis varmistaa, että uusi versio toimii yhtä hyvin kuin edeltäjänsä, ellei jopa paremmin.

Testattava sovellus on asiakkaan sisäiseen käyttöön suunniteltu web-pohjainen järjestelmä. Sovelluksella asiakas voi käydä selailemassa, päivittämässä ja hakemassa tietoja. Käyttäjä voi myös käydä etsimässä henkilön tietoja, tekemässä manuaalisia päätöksiä ja laskemassa erilaisia laskelmia. Järjestelmään voi syöttää uusia tapahtumia xml-sanomien muodossa ja tämänkin puolen toiminnallisuutta ja nopeutta testattiin. Xml-sanomia voi lähettää järjestelmään myös asiakkaan ulkopuolelta, mutta siihen tarvitsee olla rekisteröitynyt asiakas.

## 5.2 Asiakkaan ja toimittajan vaatimukset sovelluksen suorituskyvylle

Logican ja asiakkaan yhdessä määrittelemät suorituskykykriteerit ja –rajaarvot testattavalle sovellukselle on esitelty taulukossa 2.

*Taulukko 2 Suorituskyvyn tavoitearvot [10, s. 2.]*

| Mitattava tapahtuma | Tavoitearvo 90 % tapauksista | Hyväksyttävät arvot | Tapahtumia max / tunti |
|---|---|---|---|
| Tapahtuman haku | 2 sek. | 5 sek. | 100 kpl |
| Tietojen tallennus | 2 sek. | 5 sek. | 30 kpl |
| XML-sanomat | 4 sek. | 7 sek. | 420 kpl |

Tapahtuman haku saa kestää enintään 5 sekuntia. Tietojen tallennus saa kestää korkeitaan 5 sekuntia ja XML-sanomien tulisi mennä läpi alle 7 sekunnissa.

Koska sovellus on toiminut moitteettomasti tuotannossa, niin samat kriteerit säilyttäen, sovelluksen uusin versio on valmis tuotantoon. Tulevia testituloksia vertaillaan myös edellisen version testituloksiin.

### 5.3 Testien suunnittelu

Testien suunnitteluvaiheessa järjestettiin useita suunnittelupalavereita projektipäällikön kanssa, käytiin keskusteluja sovelluksen toteuttajien kanssa, vaihdettiin asiakkaan kanssa sähköposteja ja keskusteltiin toisen testaajan kanssa työnjaosta. Näin syntyi kokonaiskäsitys siitä, mitä piti testata ja mihin oli kiinnitettävä erityistä huomiota.

Toteuttajien ja projektipäällikön mielestä erityistä huomiota tuli kiinnittää sovelluksen uusiin muutoksiin, vanhoja toiminnallisuuksia tietenkään unohtamatta. Puhetta oli myös todella pitkäkestoisista testeistä, jotta saataisiin havainnollistettua mahdollista muistin täyttymisongelmaa. Nämä pitkäkestoiset testit toteutetaan itsenäisinä testeinä vasta lähitulevaisuudessa, joten ne eivät ehtineet tähän opinnäytetyöhön mukaan. [4]

Suorituskykytestimme taustalla ajettiin 1500 xml-sanomaa oman sanomatesterin kautta, siten testattava järjestelmä joutui tekemään LoadRunner–ajon taustalla koko ajan uusia päätöksiä. Sanomatiedostosta tehtiin sen verran suuri, että siitä riitti sanomia koko LoadRunner-ajon ajaksi. Sanomien lisäksi laitettiin käyntiin LoadRunner- ajo. Sen skriptit nauhoitettiin siten, että ne testasivat sovelluksen selaus-, haku-, tallennus- ja manuaalisen päätöksen muodostustoimintoja. Skripteistä ja niiden tarkemmista yksityiskohdista lisää skriptien nauhoitusta ja itse sovelluksen testausta käsittelevissä luvuissa. LoadRunner–ajo määriteltiin 4 tunnin mittaiseksi, jotta voitiin vertailla tuloksia myös edellisten versioiden testien kanssa.

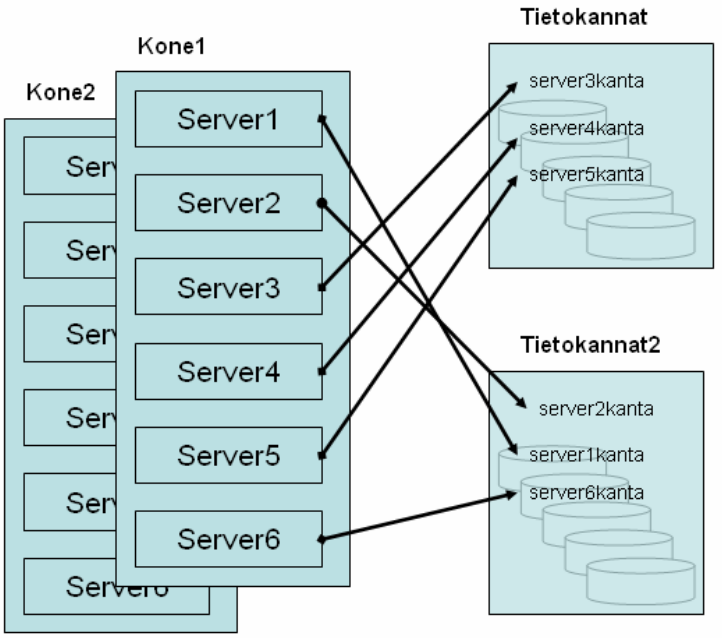
Näin toimien saatiin aikaan asiakkaan määrittelemien vaatimusten ja rajaarvojen pohjalta tilanne, joka on hyvin lähellä aidossa tilanteessa tapahtuvaa toimintaa ja kuormitusta järjestelmää kohtaan.

Testit ajettiin perjantaina 03.10.2008. Tarkka ajankohta valittiin sen tähden, että testien aikana sovelluksen muu testaus oli lopetettava kokonaan, jotta saatiin halutun vakaa ympäristö suorituskykytesteillemme.

### 5.4 Testattavan sovelluksen konfigurointi ja testiympäristö

Sovelluksen konfigurointi jätettiin kokonaan sovelluksen toteuttajien hoidettavaksi. Sovellusta ei tarvinnut konfiguroida erikseen, koska se oli tehty jo aiemmin muita testejä silmällä pitäen. Uusi xml-sanomatesteri tosin asennettiin, jotta xml-sanomien luku onnistuisi testausympäristössä nopeammin kuin ennen.

Testiympäristönä toimi Logican oma testiympäristö. Kuvassa 9 näkyvä testiympäristö on rakennettu kahdesta tietokoneesta, eli se on niin sanottu klusteri-mallinen testiympäristö. Samassa klusteri-koneessa toimi useita muitakin Logican kehityksessä olevien sovelluksien palvelimia. Testattava sovellus sijaitsi Server3:ssa, ja se käytti kantaa server3kanta.




*Kuva 9. Testiympäristö*

Logican testiympäristö muistutti asiakkaan tuotannossa käyttämää ympäristöä. Tietokannat olivat Oracle-pohjaisia. Klusterikoneissa oli 4 Gigatavun muistit, kaksi prosessorikantaa, joissa molemmissa kaksiytimiset Xeon 5140 prosessorit. Levytilaa oli molemmissa koneissa 300 Gigatavua ja verkkokortit olivat yhden Gigatavun kokoiset.

### 5.5 Testiä varten nauhoitetut skriptit

Skriptit nauhoitettiin luvussa 4.2 määritellyllä tavalla. Testiin nauhoitettiin 3 skriptiä, joilla pyrittiin simuloimaan sovelluksen reaalimaailman käyttöä mahdollisimman hyvin, mutta keskittyen samalla uusien toiminnallisuuksien rasitukseen. Seuraavassa on selvennetty nauhoitettujen skriptien toiminta.

Kaikissa kolmessa skriptissä käytetyt muuttujat (hakuehdot ja tallennettavat tiedot) parametrisoitiin. Testeissä käytettyjen skriptien parametrisoinnista lisää seuraavassa luvussa. Uusi muuttuja vaihdetaan aina kun iteraatio vaihtuu, eli virtuaalikäyttäjät tallentaa, hakee ja muodostaa päätöksiä aina eri hakuehdoilla, tunnuksilla ja valmistenumeroilla.

Tässä testissä, eli kaikissa kolmessa skriptissä, niin sisään- kuin uloskirjautumis-osio siirrettiin omille välilehdilleen, eli vuser_init ja vuser_end välilehdille. Näin tehdessä jokainen virtuaalikäyttäjä kirjautuu testin aikana sisään vain kerran ja testin päättyessä ne kirjautuvat vain kerran ulos.

Näin ei kannata tietysti menetellä jos halutaan tarkkailla sisään- ja uloskirjautumisen aikoja tai sen aiheuttamaa kuormitusta. Tässä testissä sen ei kuitenkaan nähty olevan tarpeellista. Tärkeämpänä pidettiin uusien toiminnallisuuksien testaamista.

*Skripti 1*

Ensimmäisessä skriptissä virtuaalikäyttäjä kirjautuu sisään järjestelmään, tekee haun, valitsee tämän jälkeen halutun tapahtuman hakutuloksista, selailee tapahtuman tietoja, tekee muutaman tallennuksen ja tämän jälkeen kirjautuu ulos.

*Skripti 2*

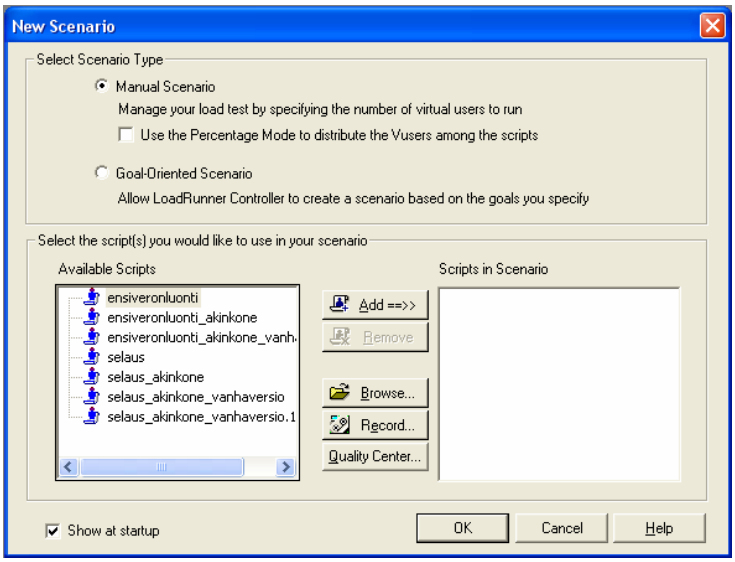
Toinen skripti on manuaalisen päätöksen muodostaja-skripti. Tässä skriptissä virtuaalikäyttäjä kirjautuu sisään järjestelmään, tekee haun (millä ei löydy hakutuloksia), täyttää uutta manuaalista päätöstä varten tarvittavat esitiedot ja tallentaa ne. Näin virtuaalikäyttäjä on lisännyt uuden manuaalisen päätöksen järjestelmään, mutta ei vie sitä päätökseen asti, joten tapahtuma jää tilaan ’käsittelyssä’. Tämän jälkeen virtuaalikäyttäjä kirjautuu ulos järjestelmästä.

*Skripti 3*

Kolmannessa skriptissä virtuaalikäyttäjä kirjautuu sisään sovellukseen, tekee haun (haulla ei löydy hakutuloksia), täyttää uutta 3 kk:n lupahakemusta varten tarvittavat tiedot, syöttää asiakkaan tiedot, muodostaa päätöksen ja tämän jälkeen kirjautuu ulos järjestelmästä.

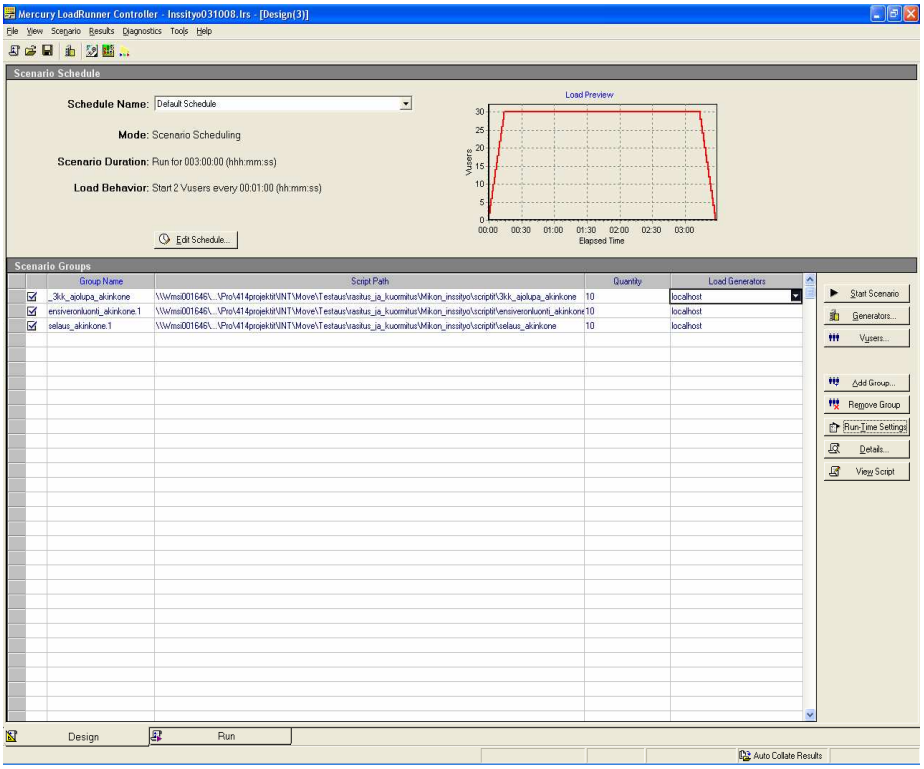
## 5.6 Sovelluksen testaus

Sovelluksen testaus toteutettiin LoadRunnerin Controller-työkalulla. Controllerissa täytyy ensin luoda skenario. Skenario kannattaa nimetä ja tallentaa sillä tavalla, että se on helposti löydettävissä tulevia käyttökertoja varten.




*Kuva 10. New Scenario –ikkuna*

Ensimmäiseksi pitää määritellä kuvassa 10 näkyvällä ’New Scenario’ -ikkunalla mitkä kaikki skriptit halutaan ottaa mukaan tulevaan skenarioon. Tässä testissä valittiin mukaan kaikki 3 nauhoitettua skriptiä ja painettiin ’OK’-nappulaa, jolloin avautui kuvassa 11 näkyvä LoadRunnerin Controller-työkalun etusivu.




*Kuva 11. LoadRunnerin Controllerin Design –välilehti*

Controller-työkalussa on Design- ja Run-välilehdet. Kuvassa 11 näkyvällä Design–välilehdellä suunniteltiin testit valmiiksi ja Run–välilehdelle siirryttiin vasta sitten, kun itse testi laitetaan käyntiin.

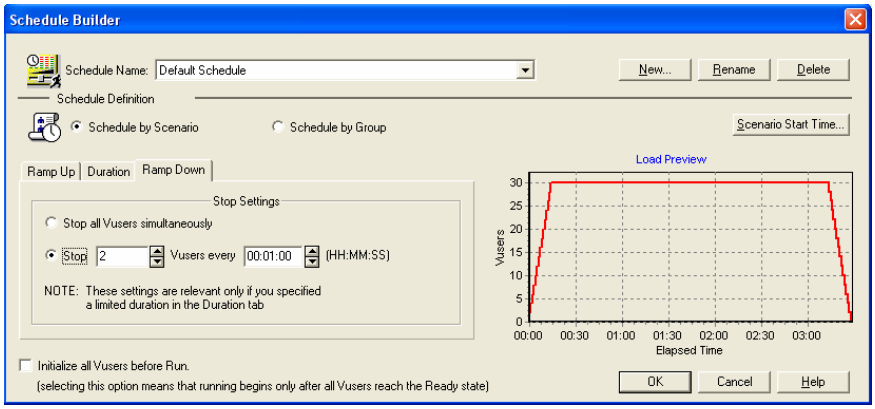
Scenario Groups-taulukossa on näkyvillä kaikki testiin valitut skriptit. Quantity-kohdassa skripteille pitää määrittää haluttu lukumäärä.

Tässä testissä ajettiin ajo seuraavilla käyttäjämäärillä

- 1.SKRIPTI Quantity 10
- 2.SKRIPTI Quantity 10
- 3.SKRIPTI Quantity 10.

Tässä testissä oli yhteensä 30 virtuaalikäyttäjää.

Edit schedule -kohdassa voi skriptille antaa halutut aikamääreet ja testin alku- ja loppupään toimintaohjeet. Klikkaamalla ’Edit Shedule’ –nappia järjestelmä avasi kuvan 13 ’Schedule Builder’ –ikkunan.




*Kuva 12. Schedule Builder –ikkuna*

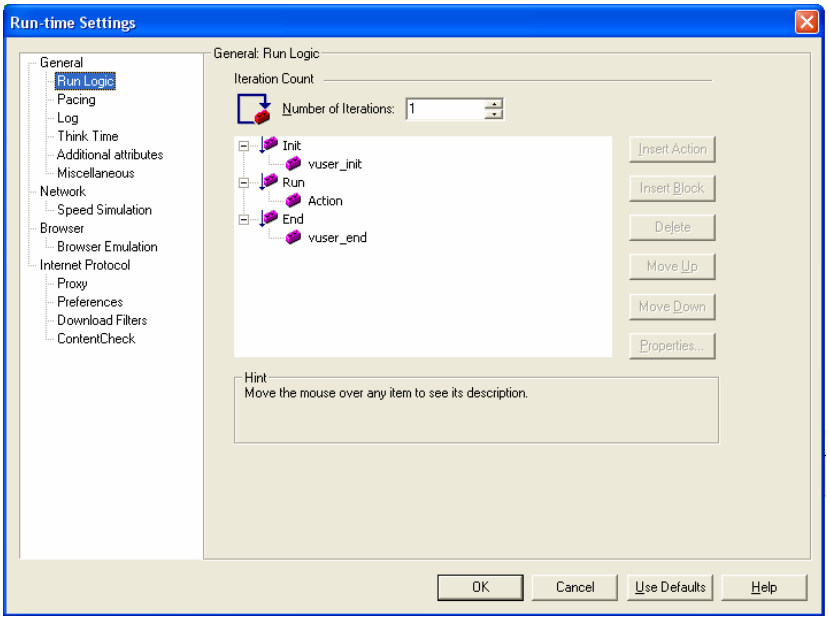
Ikkunassa annettiin testille aikamääreet: Testin alussa syötetään 2 virtuaalikäyttäjää joka minuutti. Testi ajaa itseään 03:00 tuntia ja lopussa virtuaalikäyttäjiä tiputetaan pois 2 kpl joka minuutti.

Kuvan 12 oikeassa reunassa näkyvästä Load preview –kohdasta nähdään koko testin aikainen virtuaalikäyttäjien määrä. Kun sopivat asetukset oli asetettu, niin palattiin Controlleriin painamalla ’OK’-nappulaa.

Nyt kuvan 11 Controllerin ’Load Preview’ -kohdassa on piirrettynä virtuaalikäyttäjien määrä testin aikana (sama kuva näkyy myös kuvan 12 Schedule Builder -ikkunalla).

Ennen testien aloittamista skriptit tarkistettiin ’View Skript’ -kohdasta. Näin LoadRunner avasi jälleen Virtual User Generatorin (kuva 4). Nauhoitettu skripti pyöräytettiin kerran ympäri, että nähtiin sen toimintakyky vielä ennen testin alkua. Tämä tapahtui painamalla ’Play’ –nappulaa. Tällainen varmistus oli tässä testissä tarpeen sillä edellisellä viikolla nauhoitettuihin skripteihin oli jäänyt vanhat päivämäärät, jotka olisi ajanut jokaisen tallennuksen virhetilanteeseen. Skripteistä päivitettiin oikeat päivämäärät ja näin ne olivat valmiita itse testiä varten.

Myös ’run-time settings’- asetukset muokattiin haluttuun muotoon. Näihin asetuksiin päästiin klikkaamalla kuvassa 11 näkyvää ’Run-time Settings’ –nappulaa.




*Kuva 13. Run-time Settings –ikkuna*

Kuvassa 13 näkyvästä Run-time Settings –asetuksista vaihdettiin seuraavat parametrit.

Run Logic -valikossa valittiin virtuaalikäyttäjän iteraatioiden määrä, eli kuinka monta kertaa kukin virtuaalikäyttäjä pyörittää itsensä uudestaan ja uudestaan.

Tässä testissä iteraatioasetukset määriteltiin seuraavasti:

- 1.SKRIPTI 4 -iteraatiota
- 2.SKRIPTI 2 -iteraatiota ja
- 3.SKRIPTI 2 -iteraatiota.

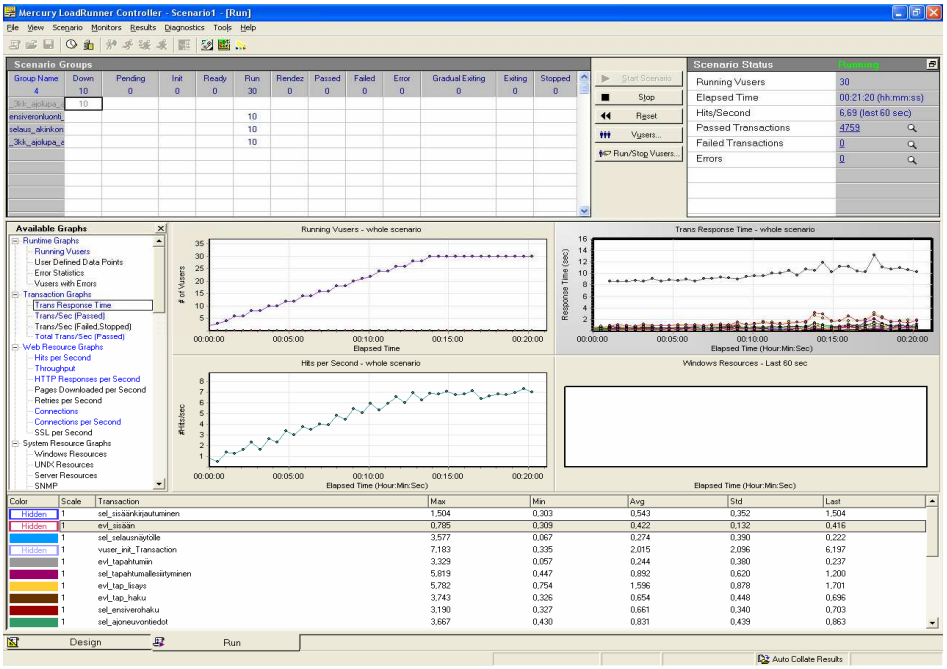
’Browse emulation’ –kohdassa otettiin valinta pois ruudusta ”Clear cache on each Iteration”, koska silloin testi mukailee reaalimaailman järjestelmän käyttöä.

Controllerin Design –välilehdelle palattiin takaisin klikkaamalla ’OK’-painiketta.

Ennen testin aloittamista käynnistettiin xml-sanomatesteri, joka syötti sähköisiä sanomia sovellukselle sitä mukaa, mitä sovellus pystyi vastaanottamaan, eli tekemään edellisen sanoman päätökseen asti valmiiksi.

Tässä suorituskykytestissä ajettiin edellisten määritysten mukaan ajo, joka alkoi portaittain nostamaan virtuaalikäyttäjien määrää. Kun LoadRunner oli saanut kaikki 30 virtuaalikäyttäjää testiin mukaan, niin ajo ajoi itseään seuraavan kolmen tunnin ajan, jonka jälkeen se alkoi portaittain tiputtamaan virtuaalikäyttäjiä pois testistä. Taustalla testattava sovellus teki sähköisiä päätöksia vastaanotetuista xml-sanomista. Kun LoadRunner–ajo oli valmiina, niin voitiin myös xml-liikenne pysäyttää.

Ennen suorituskykytestien aloittamista varmistettiin sovelluksen kehittäjiltä, että kaikki oli valmista ajon toteutukseen. Kun lupa oli saatu, niin testi pyöräytettiin käyntiin. Testi käynnistettiin ’Start scenario’ –nappulasta.




*Kuva 14. Controllerin Run -välilehti (ajo käynnissä)*

Kuvassa 14 näkyvässä Run–välilehdellä on siis testi käynnissä. Sivun vasemmassa yläkulmassa on näkyvillä virtuaalikäyttäjät ja niiden tilat. Oikealla yläkulmassa on nähtävillä testin faktat, esimerkiksi läpi menneet transaktiot, epäonnistuneet transaktiot ja testiin kulunut aika.

Sivun keskivaiheilla oleviin neljään laatikkoon saa valita haluamansa grafiikat. Valitsemalla hiirellä joku grafiikoista aktiiviseksi, tulee sivun alalaitaan näkyviin valitun grafiikan tarkemmat tiedot.

Kun testi ajoi itseään, ei testaaja voi muuta kuin odotella testien loppumista ja seurailla testin edistymistä.

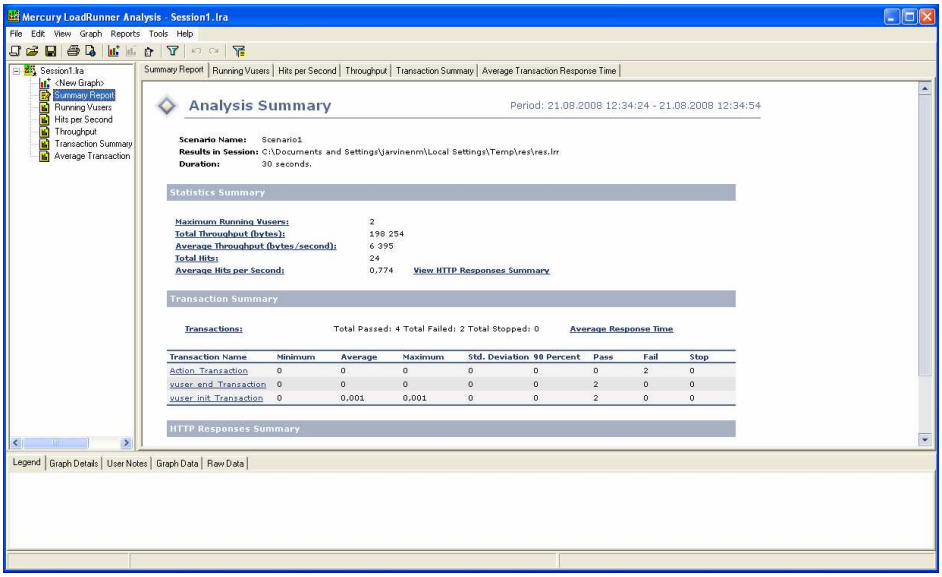
Testin loputtua Controller ilmoitti testin päättyneen ja näin siirryttiin analysoimaan tuloksia.

Huomioitavaa on myös se, että näissä LoadRunner- ajojen grafiikoissa tai näyttökuvissa ei ole mitään viitteitä xml-ajoon, joka pyöri erillisenä ajona koko ajan tämän LoadRunner-ajon taustalla.

# 6 TULOSTEN ANALYSOINTI JA RAPORTOINTI

Tulosten analysointiin käytettiin LoadRunnerin Analysis-työkalua ja tulokset raportoitiin Analysis-työkalusta ulos HTML-muotoon.

Analysis-työkaluun päästään, kun kuvassa 15 näkyvässä Controllerin ylävalikosta valitaan ’Analyze Results’.




*Kuva 15. LoadRunnerin Analysis-työkalun etusivu*

Kuvan 15 näytön vasemmassa reunassa on valikko, mihin Analysis on valinnut oletuksena muutamia grafiikoita. Nämä ovat siis kuvia, joita voi sitten muokata oman maun mukaan. Graafisiin esityksiin voi esimerkiksi lisätä tai poistaa käyriä, joita ei haluta mukaan raporttiin. Tässä testissä eroteltiin jokaisen kolmen skriptin tulokset omiksi kuvikseen ja näin niistä saatiin tarkempaa tietoa.

Analysis-työkalulla ei pysty analysoimaan Xml-viestejä, joten ne analysoitiin erikseen mittaamalla yhden sanoman käsittelyyn kulunut aika.

## 6.1 Tulosten analysointi

Tuloksia lähdettiin analysoimaan testatun sovelluksen rasitus- ja kuormitustestaussuunnitelmassa määriteltyihin raja-arvoihin nojautuen ja myös edellisiin testien tuloksia vertailtiin nyt saatuihin. Testin kokonaiskesto oli 3 tuntia, 28 minuuttia ja 33 sekuntia.

| | | |
|---|---|---|
| Maximum Running Vusers: | 30 | |
| Total Throughput (bytes): | 1 577 470 329 | |
| Average Throughput (bytes/second): | 126 056 | |
| Total Hits: | 79 280 | |
| Average Hits per Second: | 6,335 | View HTTP Responses Summary |

**Transaction Summary**

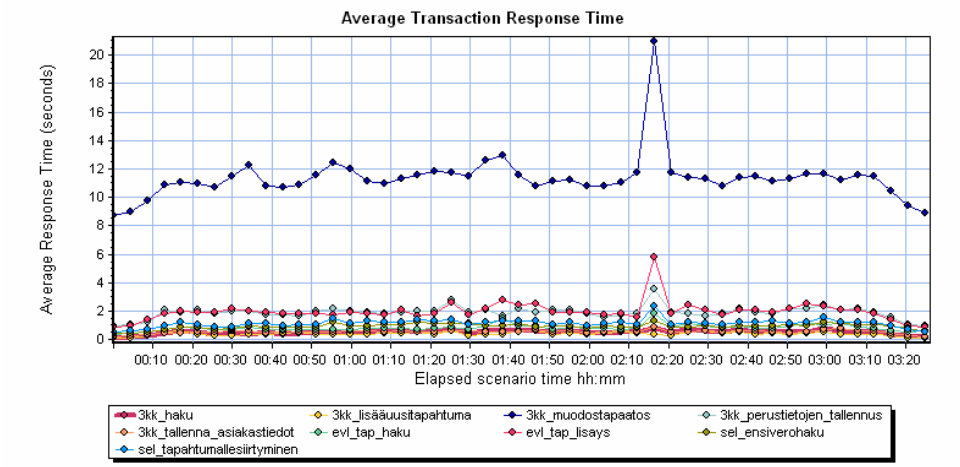
Transactions: Total Passed: 62 741 Total Failed: 2 Total Stopped: 0 Average Response Time

| Transaction Name | Minimum | Average | Maximum | Std. Deviation | 90 Percent | Pass | Fail | Stop |
|---|---|---|---|---|---|---|---|---|
| 3kk_asiakastietoihin | 0,141 | 0,52 | 7,672 | 0,422 | 0,944 | 2 725 | 1 | 0 |
| 3kk_esikatselu | 0,422 | 0,99 | 17,267 | 0,693 | 1,562 | 2 725 | 0 | 0 |
| 3kk_haku | 0,115 | 0,538 | 8,829 | 0,464 | 0,995 | 2 726 | 0 | 0 |
| 3kk_lisääuusitapahtuma | 0,078 | 0,428 | 8,078 | 0,455 | 0,884 | 2 726 | 0 | 0 |
| 3kk_muodostapaatos | 8,375 | 11,436 | 74,461 | 3,139 | 13,212 | 2 725 | 0 | 0 |
| 3kk_perustietojen_tallennus | 0,719 | 1,951 | 25,284 | 1,13 | 3,086 | 2 726 | 0 | 0 |
| 3kk_sisään | 0,306 | 0,513 | 0,961 | 0,223 | 0,884 | 10 | 0 | 0 |
| 3kk_tallenna_asiakastiedot | 0,141 | 0,571 | 6,126 | 0,397 | 1,025 | 2 725 | 0 | 0 |
| 3kk_Tapahtumiin | 0,047 | 0,264 | 6,781 | 0,367 | 0,541 | 2 726 | 0 | 0 |
| 3kk_ulos | 0,078 | 0,128 | 0,281 | 0,058 | 0,17 | 10 | 0 | 0 |
| Action_Transaction | 27,172 | 37,393 | 110,808 | 6,752 | 43,732 | 9 350 | 1 | 0 |
| evl_sisään | 0,309 | 0,422 | 0,785 | 0,132 | 0,491 | 10 | 0 | 0 |
| evl_tap_haku | 0,326 | 0,852 | 13,048 | 0,598 | 1,369 | 2 761 | 0 | 0 |
| evl_tap_lisays | 0,75 | 2,039 | 34,128 | 1,596 | 3,117 | 2 761 | 0 | 0 |
| evl_tapahtumiin | 0,047 | 0,281 | 7,063 | 0,371 | 0,572 | 2 761 | 0 | 0 |
| evl_ulos | 0,078 | 0,122 | 0,219 | 0,039 | 0,14 | 10 | 0 | 0 |
| sel_ajoneuvontiedot | 0,391 | 1,115 | 51,13 | 1,141 | 1,714 | 3 864 | 0 | 0 |
| sel_ensiverohaku | 0,327 | 0,965 | 13,517 | 0,559 | 1,542 | 3 864 | 0 | 0 |
| sel_kuntotiedot | 0,281 | 0,771 | 42,317 | 1,055 | 1,197 | 3 864 | 0 | 0 |
| sel_perustelut | 0,266 | 0,76 | 9,798 | 0,558 | 1,217 | 3 864 | 0 | 0 |
| sel_selausnäytölle | 0,063 | 0,304 | 3,577 | 0,261 | 0,592 | 3 864 | 0 | 0 |
| sel_sisäänkirjautuminen | 0,303 | 0,543 | 1,504 | 0,352 | 0,803 | 10 | 0 | 0 |
| sel_tapahtumallesiirtyminen | 0,422 | 1,179 | 28,565 | 0,822 | 1,867 | 3 864 | 0 | 0 |
| sel_uloskirjaus | 0,078 | 0,22 | 0,75 | 0,193 | 0,26 | 10 | 0 | 0 |
| vuser_end_Transaction | 0,078 | 0,157 | 0,75 | 0,126 | 0,26 | 30 | 0 | 0 |
| vuser_init_Transaction | 0,335 | 2,015 | 7,183 | 2,096 | 4,54 | 30 | 0 | 0 |

*Kuva 16. Testin faktat*

Kuvassa 17 on ajetun testin tulokset. Mitatut vasteajat ovat kuvassa aakkosjärjestyksessä. Skriptin 1 vasteajat on nimetty sel_-alkuisiksi, skriptin 2 vasteajat on nimetty evl_alkuisiksi ja skriptin 3 vasteajat on nimetty 3kk_alkuisiksi.

Jo kuvasta 17 pystyy testistä kertomaan paljon, esimerkiksi haut ovat reilusti alle määriteltyjen maksimivasteaikojen ja tietojen tallennukset ovat nekin alle hyväksyttävien tasojen.




| Color | Scale | Measurement | Graph's Minimum | Graph's Average | Graph's Maximum | Graph's Median | Graph's Std. Deviation |
|---|---|---|---|---|---|---|---|
| | 1 | 3kk_haku | 0,149 | 0,52 | 0,772 | 0,535 | 0,135 |
| | 1 | 3kk_lisääuusitapahtuma | 0,112 | 0,409 | 0,669 | 0,407 | 0,128 |
| | 1 | 3kk_muodostapaatos | 8,761 | 11,344 | 20,96 | 11,319 | 1,622 |
| | 1 | 3kk_perustietojen_tallennus | 0,935 | 1,903 | 3,523 | 1,95 | 0,422 |
| | 1 | 3kk_tallenna_asiakastiedot | 0,212 | 0,552 | 0,902 | 0,553 | 0,128 |
| | 1 | evl_tap_haku | 0,402 | 0,833 | 1,831 | 0,77 | 0,237 |
| | 1 | evl_tap_lisays | 0,856 | 1,976 | 5,756 | 1,911 | 0,678 |
| | 1 | sel_ensiverohaku | 0,422 | 0,943 | 1,36 | 0,949 | 0,187 |
| | 1 | sel_tapahtumallesiirtyminen | 0,56 | 1,148 | 2,34 | 1,172 | 0,277 |

*Kuva 17. Suorituskykytestin vasteajat*

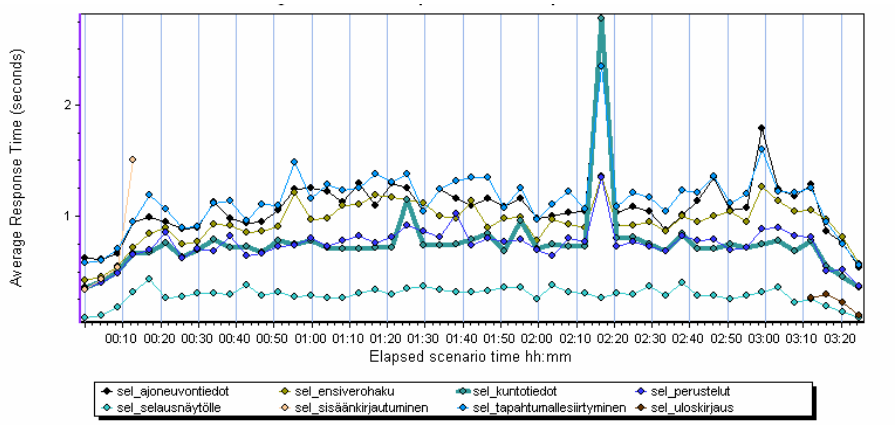
Ajetun testin tärkeimmät vasteajat on esitetty kuvassa 18. Kuvan x-akseli kuvaa testin aikaa ja y-akselilla on vasteaikoihin kulunut aika. Käyrät on eroteltu eri värein ja ne on vielä selitettynä käyrän alareunassa ja omassa taulukossaan kuvan alareunassa. Korkeimmalla näkyvä 3kk_muodostapaatos-käyrä on selvästi muita korkeammalla, eli on siis kestänyt muita kauemmin. Eli eniten aikaa vienyt operaatio oli juuri tuo päätöksen muodostus.

Kuvassa näkyvä vuorenhuippu johtui siitä, että yksi kehittäjistä unohti, muistutuksista huolimatta, suorituskykytestauksen olevan käynnissä ja kokeili yhden ajon testattavaan sovellukseen. Tämä ei kuitenkaan sotkenut testiä, vaan aiheutti ainoastaan kuvassa näkyvän äkillisen nousun. Nousun jälkeen järjestelmä palasi aiemmalle tasolleen.

Seuraavaksi analysoitiin tarkemmin ajettua testiä, eli pilkottiin edellä mainitut 3 skriptiä omiksi grafiikoikseen.

### 6.1.1 Skripti 1

Skriptin 1 vasteajat on esitetty kuvassa 19.




| Color | Graph | Scale | Measurement | Graph's Minimum | Graph's Average | Graph's Maximum | Graph's Median | Graph's Std. Deviation |
|---|---|---|---|---|---|---|---|---|
| | Average Transaction Response Time | 1 | sel_ajoneuvontiedot | 0,541 | 1,091 | 2,774 | 1,055 | 0,321 |
| | Average Transaction Response Time | 1 | sel_ensiverohaku | 0,422 | 0,943 | 1,36 | 0,949 | 0,187 |
| | Average Transaction Response Time | 1 | sel_kuntotiedot | 0,358 | 0,755 | 2,774 | 0,724 | 0,319 |
| | Average Transaction Response Time | 1 | sel_perustelut | 0,35 | 0,741 | 1,35 | 0,757 | 0,16 |
| | Average Transaction Response Time | 1 | sel_selausnäytölle | 0,088 | 0,292 | 0,437 | 0,303 | 0,073 |
| | Average Transaction Response Time | 1 | sel_sisäänkirjautumin | 0,338 | 0,703 | 1,504 | 0,538 | 0,468 |
| | Average Transaction Response Time | 1 | sel_tapahtumallesiirty | 0,56 | 1,148 | 2,34 | 1,172 | 0,277 |
| | Average Transaction Response Time | 1 | sel_uloskirjaus | 0,104 | 0,222 | 0,293 | 0,266 | 0,072 |

*Kuva 18. Skriptin 1 vasteajat*

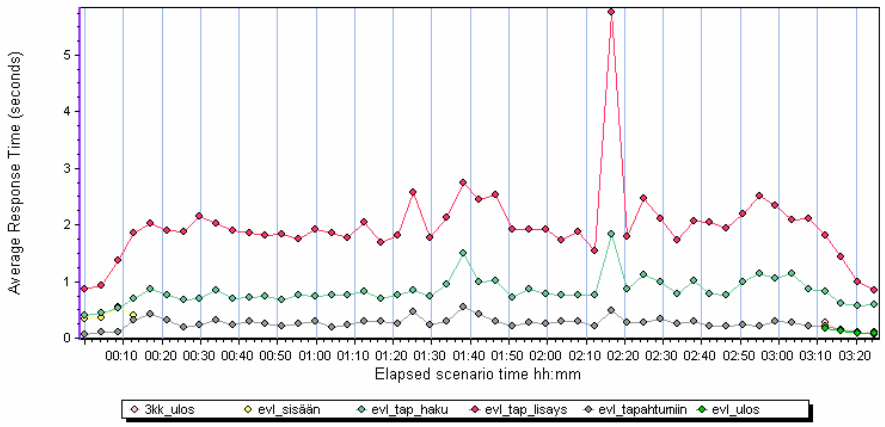
Vasteajoista voi nähdä että ne ovat reilusti alle tavoiteaikojen. Esimerkiksi haku on kestänyt keskimäärin 0,943 sekuntia, kun se olisi määritysten mukaan saanut 90 % tapauksista kestää 2 sekuntia. Jopa 5 sekunnin hakuaika olisi vielä ollut hyväksyttävän rajoissa [10, s2]. Kuntotietojen tallennus on kestänyt keskimäärin 0,755 sekuntia, kun sen tavoiteajat olivat 90 % tapauksista 2 sekuntia ja maksimissaan 5 sekuntia [10, s2].

Selaus toimii testatussa järjestelmässä todella hyvin ja reilusti alle odotusarvojen. Edellisiin testeihin verrattuna tämä osa järjestelmää toimii jopa nopeammin kuin ennen.

Myös tässä grafiikassa oleva huippu johtuu samasta syystä kuin edellä.

### 6.1.2 *Skripti 2*

Skriptin 2 vasteajat ovat esitetty kuvassa 20.




| Color | Graph | Scale | Measurement | Graph's Minimum | Graph's Average | Graph's Maximum | Graph's Median | Graph's Std. Deviation |
|---|---|---|---|---|---|---|---|---|
| | Average Transaction Response Time | 1 | 3kk_ulos | 0,089 | 0,154 | 0,281 | 0,135 | 0,076 |
| | Average Transaction Response Time | 1 | evl_sisään | 0,352 | 0,422 | 0,553 | 0,416 | 0,079 |
| | Average Transaction Response Time | 1 | evl_tap_haku | 0,402 | 0,833 | 1,831 | 0,77 | 0,237 |
| | Average Transaction Response Time | 1 | evl_tap_lisays | 0,856 | 1,976 | 5,756 | 1,911 | 0,678 |
| | Average Transaction Response Time | 1 | evl_tapahtumiin | 0,071 | 0,269 | 0,559 | 0,264 | 0,098 |
| | Average Transaction Response Time | 1 | evl_ulos | 0,094 | 0,125 | 0,18 | 0,125 | 0,034 |

*Kuva 19. Skriptin 2 vasteajat*

Samalla tavalla kuin skriptin 1 vasteajat, niin myös skriptin 2 vasteajat ovat reilusti alle tavoiteaikojen [10, s. 2.] ja näin myös tämä kohta järjestelmästä toimii loistavasti.

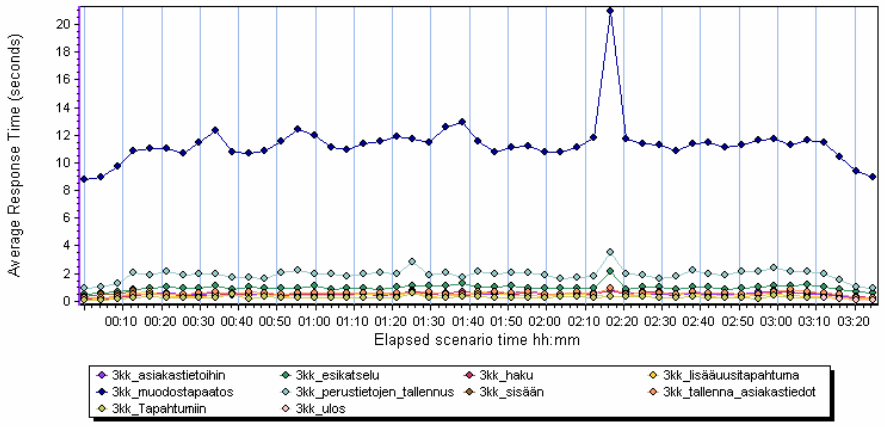
Haku kesti keskimäärin 0,833 sekuntia, tapahtuman lisäys kesti keskimäärin 1,976 sekuntia ja tapahtumiin siirtyminen keskimäärin ainoastaan 0,269 sekuntia. Nämä kaikki ovat reilusti alle tavoitteiden [10, s. 2.].

Tapahtuman lisäykselle ei testatun sovelluksen suorituskyky- ja rasitustestaussuunnitelmasta löytynyt tavoitearvoa. Sille on erillisissä palavereissä asiakkaan kanssa sovittu tavoitearvoksi, että 90 % tapauksista saa kestää 3 sekuntia ja maksimissaan 6 sekuntia. Tuo 1,976 sekuntia alittaa nuo arvot roimasti.

Suurimmat uudistukset on tehty tämän skriptin mittaamaan hakuun ja tapahtumanlisäykseen, joten niiden toimintaan kiinnitettiin tässä testissä erityistä huomiota. Ne molemmat toimivat moitteettomasti ja alle raja-arvojen, joten uudistukset ovat määritellyllä tavalla toteutettuja eikä niihin tarvitse tehdä lisämuutoksia.

### 6.1.3 Skripti 3

Skriptin 3 vasteajat on esitetty kuvassa 20.




| Color | Graph | Scale | Measurement | Graph's Minimum | Graph's Average | Graph's Maximum | Graph's Median | Graph's Std. Deviation |
|---|---|---|---|---|---|---|---|---|
| | Average Transaction Response Time | 1 | 3kk_asiakastietoihin | 0,186 | 0,503 | 0,766 | 0,512 | 0,12 |
| | Average Transaction Response Time | 1 | 3kk_esikatselu | 0,519 | 0,969 | 2,163 | 0,968 | 0,232 |
| | Average Transaction Response Time | 1 | 3kk_haku | 0,149 | 0,52 | 0,772 | 0,535 | 0,135 |
| | Average Transaction Response Time | 1 | 3kk_lisääuusitapahtu | 0,112 | 0,409 | 0,669 | 0,407 | 0,128 |
| | Average Transaction Response Time | 1 | 3kk_muodostapaato | 8,761 | 11,344 | 20,96 | 11,319 | 1,622 |
| | Average Transaction Response Time | 1 | 3kk_perustietojen_ta | 0,935 | 1,903 | 3,523 | 1,95 | 0,422 |
| | Average Transaction Response Time | 1 | 3kk_sisään | 0,333 | 0,532 | 0,784 | 0,546 | 0,164 |
| | Average Transaction Response Time | 1 | 3kk_tallenna_asiaka | 0,212 | 0,552 | 0,902 | 0,553 | 0,128 |
| | Average Transaction Response Time | 1 | 3kk_Tapahtumiin | 0,071 | 0,254 | 0,621 | 0,249 | 0,092 |
| | Average Transaction Response Time | 1 | 3kk_ulos | 0,089 | 0,154 | 0,281 | 0,135 | 0,076 |

*Kuva 20. Skriptin 3 vasteajat*

Vasteajat pysyvät määritysten mukaisina, esimerkiksi haku kesti keskimäärin vain 0,52 sekuntia, eli toimi nopeammin kuin skripteissä 1 ja 2. Myös asiakkaan tietojen tallennus kesti keskimäärin vain 0,552 sekuntia, joka on sekin reilusti alle määritysten [10, s.2].

Skriptin 3 vasteajoista pistää silmään yksi muita korkeammalla oleva käyrä. Tämä käyrä on 3kk_muodostapaatos ja sen keskimääräinen vasteaika oli 11,344 sekuntia. Tätäkään vasteaikaa ei suorituskyky- ja rasitustestaussuunnitelmasta löydy, mutta sille on määritelty asiakkaan kanssa, että 90 % tapauksista saa kestää 9 sekuntia ja maksimissaan 12 sekuntia.

Päätöksen muodostuksen vasteaika 11,344 sekuntia oli sen verran korkea, että järjestelmän toteuttajat ja projektipäällikkö olivat huolissaan sen kestosta. Siihen paneudutaan vielä tarkemmin ja sitä yritetään parannella vielä ennen sovelluksen toimitusta. Myös uudelleentestauksesta päätettiin.

Nämä uudelleentestaukset päätettiin suorittaa vähän kovemmalla kuormalla ja niistä lisää luvussa 6.3.

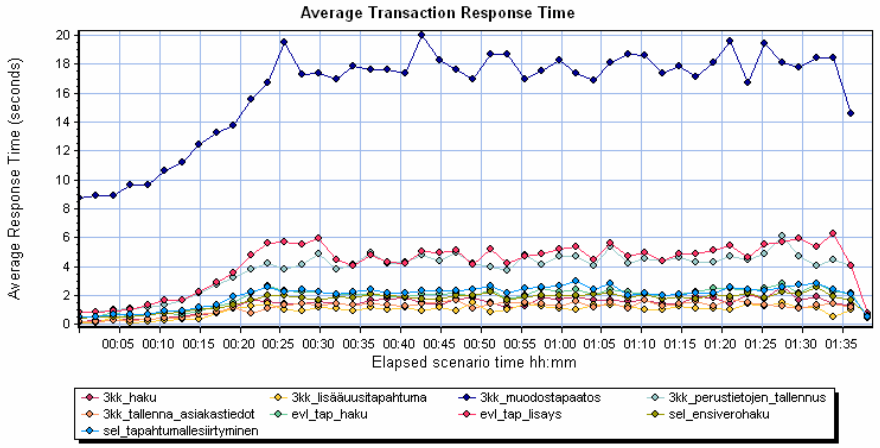
### *6.1.4 XML-viestit*

Xml-viestejä ajettiin omana tiedostonaan LoadRunner-ajon taustalla. Tiedoston koko oli 1500 viestiä ja siitä testin aikana valmiiksi ehti tulla 1397 viestiä. Yksi xml-sanoma vei siten aikaa 8,94 sekuntia. Määritysten mukaan hyväksyttävä arvo on maksimissaan 7 sekuntia [10, s2], joten xml-viestiliikenne ei toimi odotetulla tavalla ja se on toinen kohde mihin toteuttajat paneutuvat ennen sovelluksen toimitusta. Myös uuden xml-testerin toimintaa tutkittiin, mutta sen katsottiin olevan samalla tasolla kuin vanhakin, joten testerin uudistuksesta ei tässä hitaudessa ollut kyse.

## 6.2 Uudelleentestaus

Uudelleentestaus ajettiin nyt suuremmalla kuormalla, jotta nähtiin kuinka 3kk_muodostapaatos-vasteaika suhtautuu suurempaan kuormaan. Tässä testissä nostettiin skriptien 1 ja 2 käyttäjämääriä 20:een ja skriptin 3 käyttäjämäärä pidettiin kymmenessä. Skriptien iteraatio-asetukset pidettiin samoina, eli ne asetettiin arvoihin: 1.SKRIPTI 4 iteraatiota, 2.SKRIPTI 2 iteraatiota ja 3.SKRIPTI 2 iteraatiota. Testin pituutta lyhennettiin hieman ja nyt sen kokonaiskesto oli 1 tunti, 38 minuuttia ja 51 sekuntia.

Uudelleentestauksen vasteajat on esitettynä kuvassa 21.




| Color | Scale | Measurement | Graph's Minimum | Graph's Average | Graph's Maximum | Graph's Median | Graph's Std. Deviation |
|---|---|---|---|---|---|---|---|
| | 1 | 3kk_haku | 0,205 | 1,39 | 2,669 | 1,537 | 0,543 |
| | 1 | 3kk_lisääuusitapahtuma | 0,103 | 0,971 | 1,536 | 1,057 | 0,384 |
| | 1 | 3kk_muodostapaatos | 8,707 | 16,233 | 19,987 | 17,385 | 3,171 |
| | 1 | 3kk_perustietojen_tallennus | 0,828 | 3,815 | 6,112 | 4,209 | 1,296 |
| | 1 | 3kk_tallenna_asiakastiedot | 0,185 | 1,166 | 1,779 | 1,295 | 0,424 |
| | 1 | evl_tap_haku | 0,419 | 1,893 | 2,831 | 2,103 | 0,668 |
| | 1 | evl_tap_lisays | 0,766 | 4,205 | 6,246 | 4,755 | 1,575 |
| | 1 | sel_ensiverohaku | 0,413 | 1,674 | 2,555 | 1,904 | 0,551 |
| | 1 | sel_tapahtumallesiirtyminen | 0,522 | 2,026 | 2,981 | 2,223 | 0,687 |

*Kuva 21. Uudelleen testauksen vasteajat*

Vaikka samanaikainen käyttäjämäärä oli yli 100, silti kaikki muut paitsi päätöksenmuodostus oli sallituissa rajoissa [10, s.2]. Haut kestivät keskimäärin edelleen alle 2 sekuntia ja tapahtuman lisäyskin alle 5 sekuntia. Päätöksenmuodostus kesti keskimäärin 16,233 sekuntia. Uudelleentestaus vahvisti sen, että siihen kohtaan sovelluksesta täytyy paneutua uudestaan ja siihen on myös löydyttävä parannuksia ennen uuden version toimitusta.

Xml-sanomat hidastuivat nekin, johtuen lisääntyneestä kuormasta sovellusta vastaan ja tällä kertaa yhden xml-viestin kesto oli 9,96 sekuntia. Sekin on siis edelleen yli tavoitteiden ja siihenkin toteuttajat yrittävät löytää parannusta ennen version toimitusta.

### 6.3 Tulosten raportointi

Tulokset raportoitiin asiakkaalle ja sovelluksen projektipäällikölle. Asiakkaan kanssa sovittiin, että tarkemmat testiraportit toimitetaan sitten, kun pidempiaikaiset suorituskykytestit on ajettu. Nyt asiakkaalle riitti, kun heille toimitettiin LoadRunnerin Analysis-ohjelmalla toteutettu Word-testausraportti.

Projektipäällikölle koostettiin testeistä sisäinen raportti, jossa testejä oli selitetty ja kuvattu tarkemmin. Tämä ja LoadRunnerin Word-testausraportti löytyvät liitteenä työn takaa, mutta ne on luokiteltu ei-julkiseksi.

## 7 YHTEENVETO

Tässä opinnäytetyössä tavoitteena oli suunnitella ja toteuttaa suorituskykytestit eräälle web-pohjaiselle järjestelmälle Logica Suomi Oy:n valtionpalveluiden osastolla.

Suorituskykytestaukselle oli tarvetta, koska järjestelmän uuden version toimitus oli lähellä. Näin sen suorituskyky tuli testata ennen uuden version julkaisemista.

Testit toteutettiin suorituskykytestaussovellus LoadRunnerilla ja erillisellä XML-viestitesterillä. LoadRunnerin avustuksella saatiin luotua testattavalle järjestelmälle lähes reaalimaailmaa muistuttava kuormitus. Testien aikana järjestelmän tilaa monitoroitiin ja XML-viestiliikenteen nopeutta mitattiin, jotta saatiin tarkat testitulokset analysoitavaksi.

Työssä saadut testitulokset analysoitiin LoadRunnerin Analysis-ohjelmalla. Saaduista testituloksista muodostettiin raportit niin projektipäällikölle kuin sovelluksen tilanneelle asiakkaalle. Lisäksi työssä tutustuttiin suorituskykytestaukseen yleisellä tasolla.

Suorituskykytestauksessa onnistuttiin hyvin ja sen tuloksista oli paljon hyötyä sovelluksen kehittäjä-tiimille. Testissä löytyneiden hitauksien pohjalta tehdyt muutokset nopeuttivat sovellusta huomattavasti. Myös testatun järjestelmän vakaus on nyt entistä paremmalla mallilla.

Tämän opinnäytetyön tuloksena Logica Suomi Oy ja sovelluksen tilannut asiakas saivat testausraportit ja entistä toimivamman järjestelmän.

**VIITELUETTELO**


[1] Meier, J.D - Farre, Carlos - Bansode, Prahant – Barber, Scott, - Rea, Dennis. Performance Testing Guidance for Web Applications. Microsoft 2007.

[2] *LoadRunner Architecture* [verkkodokumentti, viitattu 26.08.2008] *Saatavissa:* http://www.wilsonmar.com/1loadrun.htm.

[3] *QALoad Training Guide (Release 5.6).* Kurssimoniste. Compuware. 2008. (Kurssi järjestettiin 16.–18.6.2008 Compuwaren tiloissa).

[4] Projektipäälikkö Sonja Pietilän kanssa käyty keskustelu (keskustelu käyty 21.8.2008) Logica Suomi Oy.

[5] Bjedov, Goranka. *Talk About Performance Testing* [verkkodokumentti, viitattu28.08.2008] Saatavissa: http://joychester.wordpress.com/2007/11/21/goranka-bjedov-talk-about-performance-testing/.

[6] University of Minnesota. *Testing vocabulary and Definitions* [Verkkodokumentti, viitattu 28.08.2008] *Saatavissa:* www1.umn.edu/oitqa/TestingVocabularyandDefinitions.doc

[7] *Performance testing* [Verkkodokumentti, viitattu 26.8.2008] *Saatavissa: h*ttp://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci1259932,00.html.

[8] Dustin, Elriede. *Effective Software Testing.* Addison-Wesley 2002.

[9] Tamres, Louise. *Introducing Software Testing*. Addison-Wesley 2002.

[10] Testatun sovelluksen suorituskyky- ja rasitustestaussuunnitelma Logica Suomi Oy 2006. Viitattu 03.09.2008