TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

ITI40LT

Kaarel Purde 112705

KOORMUSTESTIMISE PROTSESSI VÄLJATÖÖTAMINE JA LÄBIVIIMINE MAAKLERI RAKENDUSE NÄITEL

Bakalaureusetöö

Juhendaja: Maili Markvardt

Tehnikateaduste
magister
lektor

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kaarel Purde

15.05.2016

Annotatsioon

Käesolevas töös tutvutakse tarkvara koormustestimise protsessiga ja selle läbiviimiseks vajalike vahenditega. Koormustestimine on läbi tehtud Maakleri rakenduse näitel. Protsessi tulemuste põhjal on antud hinnang rakenduse koormustaluvusele ning soovitused edasiseks tegutsemiseks Maakleri rakenduse tarkvaraarenduse protsessis.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 45 leheküljel, 43 peatükki, 24 joonist.

Abstract

Compilation and execution of software performance testing with Maakler application

In this work we are familiarized with the process of software performance testing and the tools necessary for conducting performance tests. Performance testing is carried out with Maakler web application demonstratively. Assessment for the web applications load tolerance and suggestions for further actions are made based upon the testing results.

The thesis is in estonian and contains 45 pages of text, 43 chapters, 24 figures.

Lühendite ja mõistete sõnastik

HTTP	HyperText Transfer Protocol
GET	HTTP poolt kirjeldatud päringu alamliik. Päringu keha puudub. Päringu andmeid edastatakse osana päringu URL-st. Sõnumi eesmärk on olemasoleva loogilise andmeobjekti pärimine.
POST	HTTP poolt kirjeldatud päringu alamliik. Päringule võib lisada keha. Päringu andmeid edastatakse päringu kehas või osana päringu URL-st. Sõnumi eesmärk on uue loogilise andmeobjekti loomine.
DELETE	HTTP poolt kirjeldatud päringu alamliik. Päringule võib lisada keha. Päringu andmeid edastatakse päringu kehas või osana päringu URL-st. . Sõnumi eesmärk on olemasoleva loogilise andmeobjekti kustutamine.
FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
POP3	Post Office Protocol v3
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
TCP	Transmission Control Protocol
Connector	Tomcat rakendusserveri osa, mis võtab vastu päringuid ühel TCP pordil ning suunab need päringud edasi vastavalt seadistusele
Connection Pool	Tarkvarapakett, mis haldab ühendusi teistesse tarkvarakomponentidesse. Hoiab sisemiselt endas avatud ühendusi ning taaskastab ühendusi
Keep-alive	HTTP versioonis 1.0 kirjeldatud päringu päis, millega täpsustatakse, kui kaua ühendust hoitakse avatuna selle taaskasutamise eesmärgil
7z	Andmete tihendamise formaat nimetusega 7z või 7zip

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tarkvara testimise eesmärk on võimalikult varajane vigade tuvastamine, sest mida hilisemas staadiumis viga ilmneb, seda keerulisemaks ja kulukamaks selle parandamine muutub. Veebitarkvara puhul ei pruugi piisata ainult funktsionaalsest testimisest, et avastada kõik potentsiaalsed tõrked. Süsteemi funktsioon võib toimida ootuspäraselt, kui üks testija seda katsetab, aga kas antud funktsioon töötaks ikka veel ootuspäraselt, kui seda kasutaksid 10 samaaegset kasutajat? Või koguni 100, 1000 või 10 000 paralleelset kasutajat? Kui süsteem töötabki korrektselt, ega siis süsteemi reageerimise aeg ei veni liiga pikaks? Et saada vastus neile küsimustele, viiakse läbi koormustestimine, kus simuleeritakse koormus ning kontrollitakse, kas süsteem käitub nõuetekohaselt. Koormustaluvusega seotud probleemid avalduvad alles väga hilises staadiumis, kui süsteemi kasutab juba suur hulk kliente, ning nende parandamine võib väga kulukaks osutuda.

Käesolevas töös tutvutakse koormustestimise põhimõtete ja selleks vajaminevate vahenditega. Koormustestimise vahenditele püstitatakse kriteeriumid mille alusel valitakse antud töö jaoks sobilikud vahendid välja ning neid vahendeid õpitakse kasutama. Testitavast süsteemist on loodud ülevaade ning on kirjeldatud süsteemi seadistust ja parameetreid, mis võivad süsteemi koormustaluvuse seisukohast olulised olla. Antud töö käigus luuakse süsteemile vastavad koormustestid ning need käivitatakse. Töös on välja toodud antud testide kirjeldused ja nende läbiviimisel saadud tulemused. Koostatakse üldine hinnang süsteemi koormustaluvuse kohta ja soovitused süsteemi muutmiseks testimise tulemuste analüüsi põhjal.

Käesoleva töö autor on omanud juhtivat rolli testitava süsteemi arendamises. Antud töö on osa Maakleri rakenduse tarkvaraarenduse protsessist, mis on arendusstaadiumis töö läbiviimise ajal. Töö käigus kogutud andmed ja tulemused võetakse arvesse üleminekul toodangkeskkonnale, kus kliendid hakkaksid süsteemi kasutama.

2 Ülevaade koormustestimisest

Tarkvara testimine on osa terviklikust tarkvara arendusprotsessist. Tarkvara testimise üheks etapiks on koormustestimine, mille eesmärk on tagada arendatava tarkvara koormustaluvuse kvaliteet [1]. Tarkvara koormustestimisest võib mõelda kui mehaanilisest testimisest, mille käigus ehitusdetailile või kogu struktuurile rakendatakse mehaanilist jõudu, eesmärgiga teha kindlaks, kas see ikka talub nõuetes kirjeldatud jõudusid, seejuures ise deformeerumata. Tarkvara puhul koormatakse süsteemi paralleelsete päringute, suurte andmemahtudega või teiste süsteemi ressursse (mälu ja arvutusvõimsus) tarbivate tegevustega. Samaaegselt jälgitakse, kas süsteem käitub nõuetekohaselt koormuse all. Nagu mehaanilise testimise puhulgi, tuleks tarkvara koormustestimine viia läbi enne tema reaalsetele kasutajatele üle andmist.

Koormustestimise käigus võetakse vaatluse alla kogu süsteem, kaasa arvatud riistavaraline infrastruktuur, mitte ainult arendatav tarkvara. Süsteemi riistvaralised parameetrid mõjutavad tugevalt süsteemi koormustaluvust. Sama tarkvara, mis on installeeritud erinevatesse riistvaralistesse keskkondadesse, käitub erinevalt koormustaluvuse seisukohast. Sama väide kehtib ka operatsioonisüsteemi, draiverite jt arendatava tarkvara poolt kasutatavate tarkvarapakettide suhtes. Koormustestimise ülesanne on eelkõige ennetada koormustaluvusega seotud probleeme, mis toodangkeskkonnas reaalsete kasutajatega võiksid ilmneda. Seega testkeskkond peaks võimalikult täpselt jäljendama sellist keskkonda, mis on kasutusel ka toodangkeskkonnas, nii riistvaraliselt kui ka tarkvaraliselt. Vastasel juhul võivad koormustestimise tulemused olla eksitavad [2].

Koormustestimisel tuleks võimalikult täpselt jäljendada toodangkeskkonda. Toodangkeskkonna jäljendamine on hädavajalik tõeste tulemuste saamiseks. Lisaks sellele tuleks koormustestide läbiviimisel simuleerida reaalsete kasutajate poolt tekitatud koormust. Antud teguviis aitab koormustestide tulemustel paremini kajastada süsteemi käitumist toodangkeskkonnas päris kasutajatega. See eeldab uuringu läbiviimist, kuidas keskmine kasutaja süsteemis tegutseb. Inimkasutaja ei vii läbi erinevaid tegevusi süsteemis vahetpidamata. Inimkasutaja teeb mõttepause erinevate tegevuste vahel. Reaalsed kasutajad ei pruugi kõiki tegevusi teha alati samas järjekorras. Süsteemi koormamisel tuleks sellega arvestada, erinevate tegevuste vahele tuleks panna ootepause ja nende järjekord teatud määra juhuslikuks muuta [3].

Koormustestimine on vajalik eelkõige sellistele tarkvarasüsteemidele, kus samaaegselt saab tegutseda mitu kasutajat, nagu näiteks veebilehed, infosüsteemid (näiteks ÕIS),

mitmik arvutimängud ja teised veebirakendused. Sellised rakendused on sageli klient-server arhitektuuriga, kus üks või mõned tsentraalsed serverid teenindavad suurt hulka kliente. Suur hulk kliente, kes pöörduvad oma päringutega ühe ja sama serverimasina poole, võib märkimisväärselt koormata serveri ressursse. Seda enam, kui server peab kliendi päringute teenindamiseks ette võtma kulukaid operatsioone, nagu näiteks andmebaasi operatsioonid, tekstitöötlus, pilditöötlus, failide alla- ja üleslaadimine. Seetõttu on paralleelseid kasutajaid toetavad tarkvarasüsteemid tundlikud päringutest põhjustatud koormusele. Ühte paraleelset kasutajat toetavad töölaua rakendused on vähem tundlikud koormusele, kuna üks kasutaja ei suuda normaalse tegutsemisega tekitada märkimisväärset päringukoormust. Selliseid rakendusi saab koormata suurte andmemahtudega, kui üldse.

Koormustestimise eesmärk on ennetada süsteemi ebapiisavast koormustaluvusest tulenevate probleemide ilmnemist toodangkeskkonnas. Koormustestimise käigus peaksid potentsiaalsed koormustaluvusega seotud probleemid välja tulema enne toodangkeskkonda üleminekut.

2.1 Koormustaluvuse näitajad

Järgnevalt on toodud mõned näitajad, mille alusel saab hinnata süsteemi koormustaluvust. Antud näitajate mõõtmine koormustestimise vältel eeldab. Spetsiaalsete vahendite olemasolu. Allikaks on artikkel koormustestimisest [4].

1. Süsteemi latentsusaeg. See on aeg, mis kulub süsteemil kasutaja sisendile vastamiseks. Veebiserverite puhul on latentsusaeg ajavahemik kliendipoolse päringu väljasaatmisest kuni ajahetkeni, millal serverilt on vastus tagasi jõudnud. Süsteemi koormuse kasvades, tõuseb ka latentsusaeg. Sageli süsteemi koormustaluvusnõuded seotakse latentsusajaga.

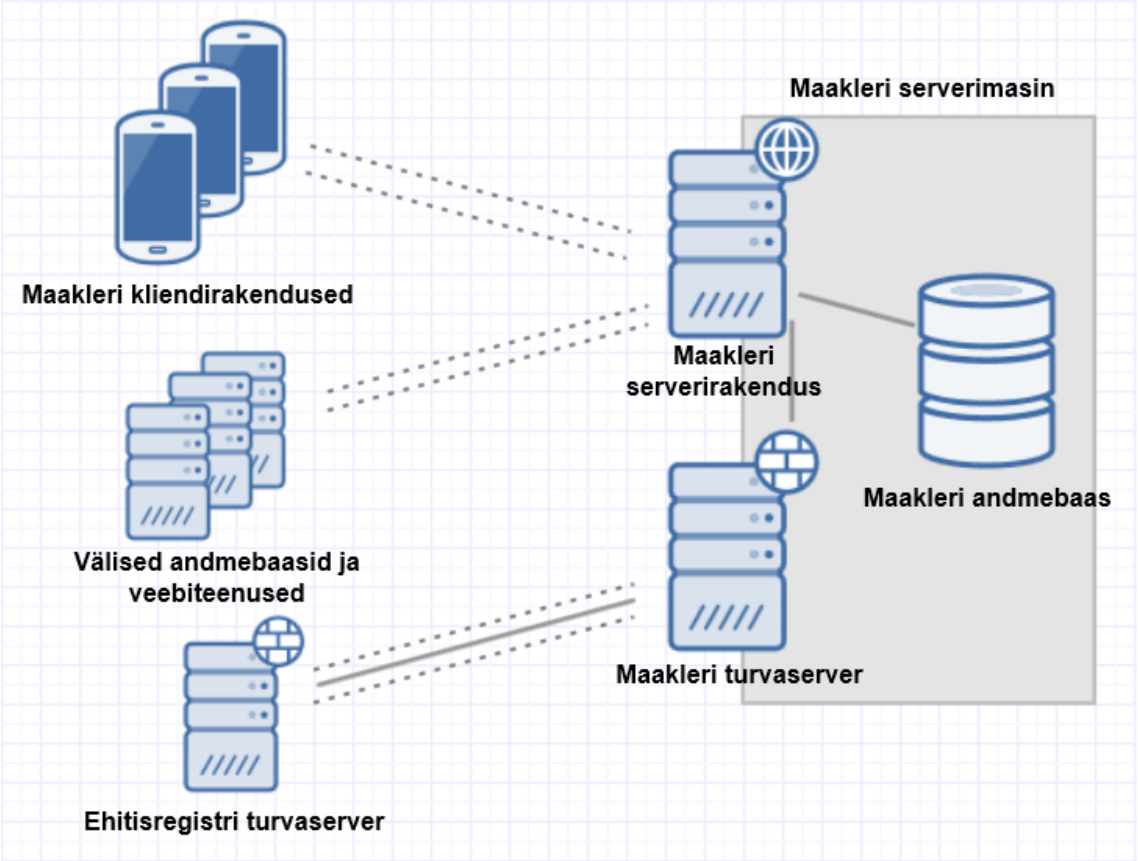
2. Vahemälu kasutus. See näitab, kui suure hulga süsteemi kasutatavast vahemälust tarbivad koormatavad tarkvarapaketid. Enne koormustestide läbiviimist märgitakse, kui palju vahemälu tarkvarapaketid kasutavad puhkeolekus ehk koormuseta. Süsteemi koormamise ajal mõõdetakse, kui palju suureneb vahemälu kasutus ning kas see läheneb maksimumini. Süsteemi koormuse kasvades tõuseb vahemälu kasutus.

3. Protsessori kasutus. Analoogselt vahemäluga jälgitakse protsessori(te) kasutust ning kas see läheneb maksimumini. Süsteemi koormuse suurenedes kasvab ka protsessori kasutus.

4. Võrguseadmete kasutus. Võrguseadmed on võrgukaart, ruuter jt seadmed, millest on võrk üles ehitatud. Süsteemi koormamisel jälgitakse, kui suur osa võrgu infrastruktuuri maskimaalsest andmete läbilaskevõimest ära kasutatakse. Süsteemi koormuse suurenedes kasvab ka võrguseadmete kasutus.

3 Ülevaade tarkvarasüsteemist Maakler

Tarkvarasüsteemi Maakler eesmärk on pakkuda mugavat kinnisvaraga seotud teenust. Süsteem on klient-server arhitektuuriga, kus üks tsentraalne server teenindab kõiki kliente. Kasutajad pääsevad süsteemile ligi läbi kliendipoolsete rakenduste. Kliendirakendusi teenindaval serverimasinal on paigaldatud 3 loogiliselt eristatavat tarkvarakomponenti: kliendirakendustega suhtlev veebirakendus (edaspidi serverirakendus või serveripoolne rakendus), andmebaas ja X-Tee turvaserver. Serverirakendus võtab vastu päringuid kliendirakendustelt, töötleb andmeid ja koostab kliendirakenduste päringutele vastuseid. Serverirakendus hoiab tööks vajaminevaid andmeid andmebaasis. Serverirakendus suhtleb Ehitisregistriga turvaserveri vahendusel üle X-Tee. Lisaks suhtleb serverirakendus veel mitmete süsteemiväliste andmebaaside ja veebiteenustega.



Joonis 1 Maakleri tarkvarasüsteemi skeem

Kliendirakendus vajab töötamiseks internetiühendust, sest teeb sagedasti päringuid Maakleri keskserverile oma töötsükli jooksul. Suhtlus kliendi- ja serverirakenduse vahel toimub üle HTTP.

Maakler kasutab Ehitisregistri andmeid oma töötsüklis. Ehitisregister on need andmed kättesaadavaks teinud läbi X-Tee infrastruktuuri. Suhtlus kahe asutuse vahel üle X-Tee toimub läbi turvaserverite. X-Tee turvaserver on spetsiaalne Riigi Infosüsteemi Ameti poolt arendatav tarkvarapakett, mis vahendab krüpteeritud X-Tee päringuid. Iga asutus, kes soovib suhelda üle X-Tee, peab registreerima ning üles seadma X-Tee turvaserveri. Kui Maakleri serverirakendus vajab andmeid Ehitisregistrilt, siis edastab ta vastava sõnumi Maakleri turvaserverile. Maakleri turvaserver teisendab saadud päringu X-Tee formaati, krüpteerib selle ning saadab läbi X-Tee infrastruktuuri Ehitisregistri turvaserverile. Ehitisregister vastab sõnumile analoogselt oma turvaserveriga. Maakleri turvaserver saab Ehitisregistri vastuse kätte, dekrüpteerib selle ning saadab andmed kokkuleppelisel kujul Maakleri serverirakendusele. Antud protsess koosneb paljudest sammudest ja on ajakulukas. Maakleri süsteemi koormustaluvuse seisukohast on see protsess oluline, sest osadele kliendirakenduste päringutele vastamiseks peab serverirakendus selle protseduuri läbi tegema.

Maakleri serverirakendus suhtleb mitmete Maakleri süsteemiväliste andmebaasidega ja veebiserveritega eesmärgil teenuse pakkumiseks vajalikke andmeid koguda. Need protsessid ei ole olulised Maakleri süsteemi koormustaluvuse seisukohast ning neid lähemalt ei kirjeldata. Antud protsessid viiakse läbi aegadel, kui on oodata kõige väiksemat koormust kliendipoolsetelt rakendustelt.

4 Ülevaade kasutusele võetud koormustestimise vahenditest

Antud töö raames koormustestide läbiviimiseks oleks tarvis kahte vahendit.

Esiteks, koormustestimise tööriist, millega saaks suures mahus teha HTTP päringuid ning päringute vastustele kulunud aega mõõta. Koormustestimisel tuleks simuleerida võimalikult tõetruult sellist koormust, nagu seda tekitaksid süsteemi reaalsed kasutajad toodangkeskkonnas [3]. Seetõttu peaks koormustestimisvahend võimaldama kasutaja tegevuse jäljendamist. See eeldab loogiliste kontrollstruktuuride olemasolu, nagu näiteks võimalus erinevate päringute järjekorda määrata, võimalus ootepause kahe päringu vahele panna, võimalus päringute tegemise järjekorda juhuslikuks muuta. Kuna rakenduse koormustaluvust testitakse mitme erineva stsenaariumiga, siis vahend peab võimaldama erinevate testjuhtumite koostamist ja nende loogilist organiseerimist. Tarvis on mõõta HTTP päringute vastuseid, koostada tulemuste alusel kokkuvõtted ja päringute ajaliste kestuste graafikud. Projektile tuleb kasuks vahendi tasuta kättesaadavus ja avatud lähtekood.

Teiseks, süsteemi jälgimise vahend, millega on võimalik rakendust töötamise käigus jälgida lähtekoodi tasandil. On vaja välja selgitada, kui kaua süsteemi erinevates osades kulub aega päringule vastuse koostamiseks. Jälgimise alla kuuluvad andmebaasi operatsioonid, päringute valideerimine, andmete teisendamise operatsioonid jt. Rakenduse tegevuse jälgimine toimub samal ajal, kui süsteemi koormatakse HTTP päringute vahendiga. Vahend peab olema kasutatav käsurealt ilma graafilise liideseta, sest testimiseks üles seatud serveril ei ole graafilist tuge. Server toetab ainult käsurida. Projektile tuleb kasuks vahendi tasuta kättesaadavus ja avatud lähtekood.

4.1 Kriteeriumid koormustestimise vahenditele

Kriteeriumid HTTP päringute tegemise vahendile on järgmised:

1. Võimaldab HTTP GET ja POST päringuid teha.
2. Päringutele kulunud aega on võimalik mõõta.
3. Päringutele saab seada juhusliku või fikseeritud järjekorra; päringute vahele saab panna ootepause.
4. Saab koostada testjuhtumeid ja neid loogiliselt organiseerida.
5. Võimaldab kujutada päringute vastusteid kokkuvõttena ja graafikuna.

6. Tasuta kättesaadav.

Järgnevalt on välja toodud kriteeriumid rakenduse jälgimise vahendile.

1. Võimaldab kindlaks määrata aega, mis kulus rakenduse erinevates osades lähtekoodi tasandil.
2. Kasutatav graafilise liideseta.
3. Tasuta kättesaadav.

4.2 Valitud koormustestimise vahendid

Koormustestimise tööriistaks valiti Apache JMeter versioon 2.13.

Erinevaid koormustestimise tööriistu, mis vastasid kõikidele eelnevalt püstitatud kriteeriumidele, leidus mitmeid, nende hulgas Gatling, Tsung, OpenSTA, Apache JMeter, HttpRider ja Pylot. Lõplik otsus langetati kriteeriumi väliseid asjaolusi arvestades, kuna kõik kaalumise all olevad vahendid vastasid juba kriteeriumidele. Otsus langetati Apache JMeter-i kasuks kuna:

1. vahendi lähtekood on avatud;
2. võrdlemisi suur ja aktiivne kasutajaskond, kelle poole saab pöörduda probleemide korral;
3. saadaval suur hulk tasuta õppematerjale;
4. vahendile on tasuta saadaval mitmed funktsionaalsust laiendavad lisapaketid.

Rakenduse tegevuse jälgimiseks lähtekoodi tasandil otsustati mitte kasutada selleks ettenähtud spetsiaalset tarkvara. Enamus jälgimistarkvara töötab ainult läbi graafilise kasutajaliidese, mille kasutamine antud töö puhul on välistatud. Rakenduse tegevuse jälgimiseks otsustati rakenduse lähtekoodi huvipakkuvatesse kohtadesse sisestada logi väljakirjutamise käsud. Antud käsud kirjutavad tekstifaili sisse ajakulu, mis kulus erinevates rakenduse osades päringutele vastuse koostamiseks. Ajakulu väljakirjutamiseks kasutatakse Tomcat rakendusserveri poolt pakutavat JULI logimise tarkvarapaketti [5]. Põhjendused antud meetodi ja JULI tarkvarapaketi kasutamiseks:

1. kõik 3 püstitatud kriteeriumit on täidetud;

2. tarkvarapaketi lähtekood on avatud;

3. äärmiselt lihtne kasutada;

4. kasutamiseks ei ole vaja tarkvara juurde installeerida, Tomcat juba sisaldab seda paketti.

4.3 Apache JMeter

Peatükk on koostatud Apache JMeter ametliku kasutusjuhendi alusel [6].

Apache JMeter on serverite koormus- ja funktsionaaltestimiseks mõeldud vahend. JMeter on tasuta saadaval ja avatud lähtekoodiga Java rakendus, mis töötamiseks vajab Java virtuaalmasinat. JMeter on rohkete kasutusvõimalustega, millest antud töö raames rakendatakse vaid murdosa. Tööriist toetab mitmeid erinevaid päringuprotokolle, nagu näiteks HTTP, FTP, SMTP, POP3, SOAP jt. Tööriist võimaldab testjuhtumeid loogiliselt organiseerida. Päringuid on võimalik järjestada ja päringute vahele on võimalik ootepause määrata, nagu koormustestimise vahendi kriteeriumides ette nähtud. Päringute vastuste alusel on võimalik graafikuid koostada ja erinevaid kokkuvõtteid teha.

Vahend võimaldab testjuhtumeid ja testjuhtumite tulemusi salvestada ning hiljem taas avada. Salvestamiseks kasutab tööriist XML formaati, mis tähendab, et testjuhtumeid saab koostada JMeter-ist eraldiseisvalt tekstiredaktoriga.

JMeter töötab läbi graafilise kasutajaliidese või läbi käsurea. Graafiline kasutajaliides võimaldab mugavat testjuhtumite koostamist, testjuhtumite käivitamist ning tulemuste vaatamist. Läbi käsurea saab ainult testjuhtumeid käivitada. Väga koormavate testjuhtumite puhul soovitatakse nad käivitada käsurea kaudu, mitte läbi graafilise liidese. Antud teguviis vähendab koormust teste läbiviivale arvutile.

4.3.1 Testide organiseerimine

Testjuhtumite organiseerimine toimub *Test Plan* elemendiga (vt Lisa 1 joonis 1). Üks *Test Plan* element sisaldab endas ühte testjuhtumit. Graafilises kasutajaliideses saab korraga olla avatud täpselt üks *Test Plan* element.

Testjuhtumi käivitamisel käivitatakse kõik *Test Plan* elemendi sees asuvad *Thread Group* elemendid.

Kui säte *Run Thread Groups consecutively* on tõene, siis *Thread Group* elemendid käivitatakse järjestikuliselt ükshaaval. Vastasel juhul *Thread Group* elemendid käivitatakse kõik üheaegselt.

4.3.2 Lõimed

Kõik tegevused, mida JMeter võimaldab, viiakse läbi selleks ette nähtud lõimede poolt. Lõimed peab eelnevalt deklareerima *Thread Group* elemendiga (vt Lisa 1 joonis 2). Elemendi *Thread Group* sisse saab paigutada erinevaid käske, mida iga lõim peab täitma, kui testjuhtum käivitatakse. Element *Thread Group* võimaldab täpsustada, mitu lõime käivitamisel luuakse.

Väli *Number of Threads* määrab, mitu uut lõime testjuhu käivitamisel luuakse.

Väli *Ramp-Up Period* määrab, millise ajaperioodi vältel kõik lõimed käivitatakse.

Väli *Loop Count* määrab, mitu korda elementi *Thread Group* käivitatakse. Väärtus 1 määraks, et lõimed käivitatakse ainult ühe korra. Väärtus 2 määraks, et kui kõik lõimed antud *Thread Group* elemendis on oma käsud lõpetanud, siis nad käivitatakse ühe korra uuesti.

Väljaga *Forever* on võimalik lõimed lõpmatuseni tsükliliselt käima panna.

4.3.3 HTTP päringud

Elemendiga *HTTP Request* antakse lõimele käsk teha HTTP päring. See element tuleb paigutada elemendi *Thread Group*-i sisse, et lõimed saaksid seda täita (vt Lisa 1 joonis 3).

Väli *Server Name or IP* määrab ära, mis aadressile päring tehakse.

Väli *Port* määrab, mis TCP pordile päring tehakse.

Väli *Method* määrab, millist HTTP meetodit päringu tegemiseks kasutatakse. Toetatud on POST, GET, DELETE jt meetodid.

Väli *Path* täpsustab, milline on päringu sihtkoht antud aadressil. Näiteks, kui soovitakse päring teha aadressile https://en.wikipedia.org/wiki/Main_Page, siis välja *Path* väärtuseks tuleks panna wiki/Main_Page.

Väli *KeepAlive* võimaldab TCP ühenduste taaskasutamist. Kui see seadistus on tõeseks määratud, siis TCP ühendused jäetakse taaskasutamise eesmärgil avatuks. Kui lõim hakkab päringut tegema, siis kõigepealt püüab ta olemasolevaid TCP ühendusi taaskasutada. Kui ühtegi avatud TCP ühendust ei ole saadaval, alles siis luuakse uus ühendus. Joonisel on seadistus vääraks seatud, seega iga päringu jaoks luuakse uus TCP ühendus, mis päringu lõppedes suletakse ega jäeta avatuks taaskasutamise eesmärgil.

4.3.4 Juhtelemendid

Järgnevalt on välja toodud mõned elemendid, millega juhtida testide tööd.

Element *Constant Timer* (vt Lisa 1 joonis 4), tekitab ootepausi lõime töös.

Väli *Thread Delay* elemendis *Constant Timer* määrab, kui pikk on ooteperiood lõimedele.

Element *Loop Controller* (vt Lisa 1 joonis 5) võimaldab luua tsükleid testplaanis. Antud elemendi sisse on võimalik paigutada teisi käske, millega saavutatakse samade käskude korduv täitmine.

Väli *Loop Count* elemendis *Loop Controller* määrab, mitu korda tsükli sees olevad käsud täidetakse.

Element *Random Order Controller* (vt Lisa 1 joonis 6) võimaldab juhuslikku järjekorda käskude täitmisel. Element mõjutab käske, mis on paigutatud tema sisse.

4.3.5 Tulemuste mõõtmine

Järgneval kirjeldatakse mõnda elementi, millega on võimalik lõimede tööd mõõta.

Element *View Result Tree* (vt Lisa 1 joonis 7) salvestab põhjaliku informatsiooni kõikidest päringutest, mida lõimed teevad. Antud element paigutatakse *Thread Group* elemendi sisse.

Elemendis *View Result Tree* tulemuste leheküljel väli *Load time* näitab, kui suur oli päringu kogu ajakulu. Teisisõnu, ajavahemik hetkest, kui JMeter päringu välja saatis, hetkeni, kui JMeter vastuse täielikul kujul kätte sai.

Elemendis *View Result Tree* tulemuste leheküljel näitab väli *Connect time* kui palju võttis JMeter-il aega, et testitava serveriga ühendus luua.

Elemendis *View Result Tree* tulemuste leheküljel näitab väli *Latency* kui pikk oli ajavahemik hetkest, kui JMeter päringu välja saatis, hetkeni, kui esimene vastuse pakett tagasi jõudis. TCP ühenduste puhul saadetakse sõnumid diskreetsete pakettidena, mitte tervikuna.

Elemendis *View Result Tree* tulemuste leheküljel näitab väli *Size in bytes* vastuse suurust baitides.

Elemendis *View Result Tree* paneeliga *Write results to file / Read from file* on võimalik antud tulemused eksportida XML formaadis.

Element *Aggregate Report* (vt Lisa 1 joonis 8) koostab kokkuvõtte kõikidest päringutest. Antud element paigutatakse *Thread Group* elemendi sisse.

Elemendis *Aggregate Report* kujutatakse andmeid tabelis. Iga individuaalse päringu sihtkoha jaoks koostatakse tabelis üks rida ja kõikide sihtkohtade jaoks üks kokkuvõttev rida nimega *TOTAL*.

Elemendis *Aggregate Report* tulemuste tabelis tulp *#Samples* näitab, mitu päringut antud sihtkohta tehti.

Elemendis *Aggregate Report* tulemuste tabelis tulp *Average* näitab, milline oli aritmeetiliselt keskmine päringu kestus.

Elemendis *Aggregate Report* tulemuste tabelis tulp *Median* näitab, kui suur oli mediaankeskmine päringu kestus.

Elemendis *Aggregate Report* tulemuste tabelis tulp *90% Line* näitab, milline oli suurim päringukestus 90% kiiremate päringute seas. Teisisõnu, kui kõikidest päringutest jätta välja 10% aeglasemad, siis allesjäänud päringutest kõige aeglasem võetakse tulba *90% Line* väärtuseks.

Elemendis *Aggregate Report* tulemuste tabelis tulp *95% Line* näitab, milline oli suurim päringukestus 95% kiiremate päringute seas.

Elemendis *Aggregate Report* tulemuste tabelis tulp *99% Line* näitab, milline oli suurim päringukestus 99% kiiremate päringute seas.

Elemendis *Aggregate Report* tulemuste tabelis tulp *Min* näitab, milline oli minimaalne päringu kestus.

Elemendis *Aggregate Report* tulemuste tabelis tulp *Max* näitab, milline oli suurim päringu kestus.

Elemendis *Aggregate Report* tulemuste tabelis tulp *Error%* näitab, mitu protsenti kõikidest päringutest ebaõnnestusid.

Elemendis *Aggregate Report* tulemuste tabelis tulp *Throughput* näitab, kui suur oli keskmine päringu läbilaskvus testi vältel (ühikuga päringut sekundis).

Elemendis *Aggregate Report* tulemuste tabelis tulp *KB/sec* näitab, kui suur oli keskmine andmevahetuse kiirus JMeter-i ja testitava serveri vahel (ühikuga kilobaiti sekundis).

Elemendis *Aggregate Report* paneeliga *Write results to file / Read from file* on võimalik antud tulemused eksportida XML formaadis.

Element *Response Time Graph* (vt Lisa 1 joonis 9) koostab graafiku keskmistest päringukestustest testi vältel. Iga päringu sihtkoha jaoks on üks joon graafikul.

Elemendis *Response Time Graph* graafikul horisontaalsel teljel on testi töötamise aeg. Vertikaalsel teljel keskmine päringu kestus antud ajahetkel.

Elemendis *Response Time Graph* graafilise kasutajaliidese paneeliga *Write results to file / Read from file* on võimalik antud tulemused eksportida XML formaadis.

4.4 Logimise tarkvarapakett JULI

JULI on rakenduse logi kirjutamise tarkvarapakett, mis tuleb vaikimisi Tomcat rakendusserveriga kaasa. JULI põhineb *Apache Commons Logging* paketil ning on kohandatud Tomcat-i rakendusserveri vajadustele [2].

JULI paketile pääseb ligi serveripoolsest rakendusest läbi Java standardse logi kirjutamise liidese, mis asub paketis *java.util.logging*. Ühendus Java standardse logi kirjutamise liidese ja JULI logi kirjutamise implementatsiooni vahel on loodud Tomcat-i vaikekonfiguratsiooniga. Teisisõnu, ühendust antud liidese ja implementatsiooni vahel ei ole vaja rakenduses programmaatiliselt luua, sest see on juba vaikimisi olemas.

```
import java.util.logging.Level;
import java.util.logging.Logger;

public class LoggingUtility {

      private static void logInfo(Class caller, String message) {
            Logger.getLogger(caller.getSimpleName()).log(Level.INFO,
                  message);
      }
}
```

Joonis 2 Näide JULI logimise kasutamisest programmeerimiskeeles Java

JULI logimise paketi kasutamine käib läbi standardse Java liidese *Logger.getLogger()*. Programmeerija ei pruugi teadlik ollagi, et liidese taga on JULI logi kirjutamise implementatsioon.

5 Koormustestide läbiviimine

Antud töös piirdutakse ainult serveripoolse rakenduse koormustestimisega. Kliendipoolse rakenduse koormustestimine jääb väljapoole selle töö piire. Põhjendused selleks on järgmised.

Kliendipoolse rakenduse koormustestimine ei ole kriitilise tähtsusega: rakendus ei satu oma töötsükli jooksul kordagi märkimisväärse koormuse alla, seetõttu oleks tööjõu kulutamine kliendirakenduse koormustaluvuse väljaselgitamiseks tulutu tegevus.

Androidi rakenduste koormustestimiseks mõeldud vahendid ei ole niivõrd efektiivsed, kui serverite koormustestimiseks mõeldud vahendid. See asjaolu suurendaks tööjõu kulu koormustestimise läbiviimiseks.

Androidi serveripoolse rakenduse testimine ei annaks projektile piisavalt lisaväärtust, et õigustada tööjõu kulu, mis selle läbiviimine nõuaks.

Serverirakenduse koormustestimine viiakse läbi HTTP päringutega, mis imiteerivad kliendirakenduse poolt tehtavaid päringuid. Koormustestimine läbi kliendirakenduse oleks liiga ressursimahukas ettevõtmine. Koormustestimises tekitatakse koormust suure hulga paralleelsete kasutajatega, mis tähendaks suure hulga samaaegsete kliendirakenduste tööle panemist. Kliendirakendus töötab Android operatsioonisüsteemil. Seega oleks tarvis ligipääsu suurele hulgale füüsilistele Androidi seadmetele või oleks tarvis emuleerida suurel hulgal virtuaalseid Androidi seadmeid. Mõlemad variandid oleksid liiga kulukad antud töö jaoks.

5.1 Koormustaluvuse nõuded

Serveripoolsele rakendusele seatud koormustaluvuse nõuded:

1. süsteemis saab üheaegselt tegutseda 1000 kasutajat;
2. vähemalt 95% päringutest kestab alla 1 sekundi.

Kliendipoolsele Androidi rakendusele ei ole koormustaluvuse nõudeid seatud, kuna ei ole ette nähtud, et rakendus satuks kordagi märkimisväärse koormuse alla oma töötsükli jooksul. Koormustaluvusnõuete püstitamine ja kontrollimine kliendipoolse rakenduse puhul ei ole kriitilise tähtsusega.

5.2 Testkeskkonna kirjeldus

Testide läbiviimiseks on serveripoolne rakendus ja seda toetav tarkvara üles seatud selleks ettenähtud virtuaalserveril. Testkeskkonna ja toodangkeskkonna virtuaalserveritele on eraldatud identsed riistvaralised ressursid ja installeeritud samad tarkvarapaketid.

Kasutatud on zone.ee poolt pakutavat virtuaalset serveriteenust (Pilveserver PRO SSD). Virtuaalserverile on eraldatud üks tuum protsessorist Intel Xeon E5-2630 2.60GHz, 1Gb vahemälu ja 30Gb püsimälu SSD tüüpi mäluseadmel. Virtuaalserveril töötab operatsioonisüsteem Ubuntu 14.04 LTS 64-bit. Paigaldatud on Oracle Java virtuaalmasin 1.8.0_74 64-bit, andmebaasisüsteem PostgreSQL 9.3, X-Tee turvaserver v5.5 ja rakendusserver Apache Tomcat 8.0.32.

Serveripoolne Java veebirakendus on installeeritud Tomcat rakendusserverile ja käivitatud.

5.2.1 Tomcat seadistus

Järgnevalt on kirjeldatud rakendusserveri Tomcat 8 seadistust, mis võib olla oluline süsteemi koormustaluvuse seisukohast. Antud seadistuse muutmisega on võimalik mõjutada süsteemi koormustaluvust. Kui koormustestimise käigus selgub, et Tomcat-i seadistus põhjustab probleeme süsteemi koormustaluvuse osas, siis pakutakse välja lahendusi seadistuse muutmiseks.

```
<Connector port="9080" protocol="HTTP/1.1"
connectionTimeout="20000"
maxConnections="8000"
redirectPort="9443" />
```

Joonis 3 väljavõte Tomcat konfiguratsioonifailist conf / server.xml

Connector instants on pandud vastu võtma sissetulevaid päringuid pordil 9080. Virtuaalserverisse ei ole paigaldatud *APR/native Connector* implementatsiooni, seega atribuudi väärtus protocol="HTTP/1.1" tähendab, et kasutusel on vaikimisi *NIO Connector* implementatsioon. *NIO* implementatsioon võimaldab suurema ühenduste läbilaskvuse kui *BIO Connector* implementatsioon, kuna erinevalt *BIO* implementatsioonist vabastab *NIO* ühendust teenindava lõime peale igat päringut, ka juhul, kui tegu on *HTTP/1.1 keep-alive* ühendusega.

Väärtustamata on jäetud atribuut *maxThreads*, seega on kasutusel vaikimisi väärtus 200. See tähendab, et Tomcat teenindab samaaegselt kuni 200 ühendust pordil 9080. Kui see arv ületatakse, siis ülejäänud ühendused pannakse järjekorda ootele, kuni teenindavad lõimed vabanevad.

Atribuut *maxConnections*="8000" tähendab, et kuni 8 000 ühendust võib järjekorras oodata teenindavate lõimede vabanemist. Kui see arv ületatakse, siis järgnevatest ühendustest Tomcat keeldub ning need suletakse automaatselt veateatega: org.apache.http.conn.HttpHostConnectException: Connection to http://localhost:9080 refused.

Atribuut connectionTimeout="20000" tähendab, et uue ühenduse puhul Tomcat ootab kuni
20 000 ms, et klient saadaks päringu keha või päringu URI osa. Kui klient ei ole selle aja möödudes saatnud täielikku päringut, siis Tomcat sulgeb ühenduse.

Peatükis on allikana kasutatud ametlikku Tomcat HTTP Connector dokumentatsiooni [7].

5.2.2 Andmebaasi seadistus

Järgnevalt on kirjeldatud andmebaasisüsteemi PostgreSQL 9.3 seadistust, mis võib olla oluline süsteemi koormustaluvuse seisukohast. Antud konfiguratsiooni muutmisega on võimalik mõjutada süsteemi koormustaluvust. Kui koormustestimise käigus selgub, et andmebaasi konfiguratsioon põhjustab madalat koormustaluvust, siis pakutakse välja lahendusi konfiguratsiooni muutmiseks.

Andmebaasiga saab luua maksimaalselt 30 samaaegset ühendust. See on seatud atribuudiga *max_connections* [8].

```
max_connections = 30
```

Joonis 4 väljavõte PostgreSQL konfiguratsioonifailist postgresql.conf

Kui maksimaalsete samaaegsete ühenduste arv 30 on täitunud, siis järgnevatest ühendustest PostgreSQL keeldub veateatega, neid ühendusi ei panda järjekorda ootele.

5.2.3 Andmebaasi kirjeldus

Järgnevalt on välja toodud andmebaasi omadused, mis võivad koormustaluvuse seisukohast olulised olla.

Testimiseks kasutatav andmebaas on programmiga *pg_dump* loodud täielik koopia reaalsest andmebaasist [9].

Koormustestide läbiviimise ajal on andmebaasis ligikaudu 15 000 rida, mis on võrreldav andmehulgaga, mis oleks andmebaasis toodangkeskkonnale üleminemise hetkel. Koos kasutajaskonna suurenemisega toodangkeskkonnas prognoositakse andmemahu kasvu

paarisajatuhande reani poole aasta jooksul. Praegune seadistus, 15 000 rida, kajastab andmemahte esimese paari kuu vältel toodangkeskkonnas.

5.2.4 Maakleri serveripoolse rakenduse kirjeldus

Järgnevalt on kirjeldatud serverirakenduse konfiguratsiooni, mis võib olla oluline süsteemi koormustaluvuse seisukohast. Kirjeldatud on rakenduse ühendust andmebaasiga ja rakenduse siseseid lõimesid. Antud konfiguratsiooni muutmisega on võimalik mõjutada süsteemi koormustaluvust. Kui koormustestimise käigus selgub, et serverirakenduse konfiguratsioon põhjustab madalat koormustaluvust, siis pakutakse välja lahendusi konfiguratsiooni muutmiseks.

Kuna andmebaasisüsteemis PostgreSQL ei ole sisse ehitatud *connection pool* funktsionaalsust [10], siis Maakleri serveripoolne rakendus kasutab andmebaasiga ühenduste loomiseks Tomcat-i poolt pakutavat *connection pool* paketti. Andmebaasiühenduse loomine on ressursimahukas ja küllaltki aeganõudev protsess [11]. *Connection pool* oluliselt vähendab andmebaasiühenduste loomisele kuluvat aega ja ressursside tarvitamist, mis saavutatakse olemasolevate avatud andmebaasiühenduste taaskasutamisega [12]. Kasutatav *connection pool* on konfigureeritud järgmiselt:

```
<Resource name="jdbc/MAAKLER_TEST"
auth="Container"
type="javax.sql.DataSource"
driverClassName="org.postgresql.Driver"
url="jdbc:postgresql://localhost:5432/MAAKLER_TEST
username="..."
password="..."
maxTotal="20"
maxIdle="20"
fairQueue="true"
maxWaitMillis="15000"/>
```

Joonis 5 väljavõte serverirakenduse konfiguratsioonifailist META-INF / context.xml

Atribuutidega *maxTotal*="20" ja *maxIdle*="20" on tekitatud olukord, et igal ajahetkel on *connection pool-is* avatud 20 ühendust andmebaasiga. Maakleri serveripoolsel rakendusel on võimalik pöörduda *connection pool*-i poole ja küsida avatud ühendust andmebaasiga. *Connection pool* tagastab kasutusvalmis andmebaasiühenduse või kui kõik 20 ühendust on juba hõivatud, siis pannakse küsiv lõim järjekorda mõne ühenduse vabanemist ootama. Kui lõim on oma tegevused andmebaasiühendusega lõpetanud, siis tagstatakse ühendus *connection pool*-ile taaskasutamiseks, seda sulgemata.

Atribuut *fairQueue*=”true” täpsustab, et ootejärjekorrast võetakse alati see lõim, mis on kõige pikemalt oodanud.

Atribuut *MaxWaitMillis*=”15000” täpsustab, et lõim võib ootejärjekorras viibida maksimaalselt 15 000 ms. Ajapiirangu ületamisel lõim väljutatakse järjekorrast erandiga java.sql.SQLTimeoutException.

Konfiguratsiooni atribuutide puhul on allikana kasutatud ametlikku *Tomcat JDBC Connection Pool* dokumentatsiooni [13].

Maakleri serveripoolne rakendus on üles ehitatud selliselt, et võimalikult kiiresti vabastada Tomcat'i kliendi päringuid teenindavad lõimed (Tomcat konfiguratsioonis atribuudiga *maxThreads* on sissetulevaid päringuid teenindavate lõimede arvuks seatud 200). Kõik tegevused, mis nõuavad rohkem aega ja ei ole vajalikud päringule vastuse koostamiseks, tehakse ära taustal teiste lõimedega. See võimaldab kiiremini vabastada Tomcat'i päringuid teenindavad lõimed ja päringutele kiiremini vastused tagasi saata. Taustategevuste läbiviimiseks mõeldud lõimede arv on piiratud 20-le *thread pool* komponendi abil paketist *java.util.concurrent.ThreadPoolExecutor*. *Thread pool* on tarkvarakomponent, mis võimaldab rakendusesiseselt piirata lõimede arvu ja neid taaskasutades ressursse kokku hoida [14].

```
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

public class TrackingThreadPool extends ThreadPoolExecutor {
      private static final int MAX_WAITING_TASKS = 4000;
      private static final int MAX_ACTIVE_THREADS = 20;
      private static final int MIN_ACTIVE_THREADS = 5;
      private static final int DEACTIVATE_THREADS_AFTER_SECONDS = 60;

      private static final BlockingQueue<Runnable> waitingTasks = new
            LinkedBlockingQueue<>(MAX_WAITING_TASKS);

      private TrackingThreadPool() {

            super(MIN_ACTIVE_THREADS, MAX_ACTIVE_THREADS,
                  DEACTIVATE_THREADS_AFTER_SECONDS, TimeUnit.SECONDS,
                  waitingTasks);
      }

private static final TrackingThreadPool instance = new
TrackingThreadPool();

…
```

Joonis 6 väljavõte Maakleri serverirakenduse klassist TrackingThreadPool

Joonsiel 6 välja toodud klass *TrackingThreadPool* piirab rakenduse siseselt maksimaalset lõimede arvu.

Igal ajahetkel on 5 lõime valmis taustategevusi läbi viima. Kui taustategevuste hulk kasvab kiiremini, kui 5 lõime suudavad neid täita, siis listakse uusi lõimesid kuni 20-ni. Kui lõimesid ei rakendata taustatöödega 60 sekundi jooksul, siis lõimede hulka *thread pool*-is vähendatakse tagasi kuni 5-ni. Kui kõik 20 lõime on hõivatud, siis järgnevad taustatööd pannakse ootele järjekorda, kus saab oodata maksimaalselt 4000 taustatööd.

5.2.5 Suhtlus kliendi- ja serverirakenduse vahel

Järgnevalt on kirjeldatud kliendi- ja serverirakenduse vahelist suhtlust, mis potentsiaalselt mõjutab süsteemi koormustaluvust.

Suhtlus kliendi- ja serveripoolse rakenduse vahel toimub *HTTP/1.0* sõnumite vahendusel. Iga päringu jaoks luuakse uus TCP ühendus. Ühendusi ei taaskasutata.

Sõnumid saadetakse avatud tekstina. Sõnumid ei ole tihendatud ega krüpteeritud. Keskmise sõnumi suurus on paarkümmend kuni mõnisada baiti. Erandiks on serveri punkti /ads päringuvastused, mis on ligikaudu pool megabaiti suured. Eeldatakse, et toodangkeskkonnas antud sõnumi mahud kasvavad 5 kuni 10 megabaidi suuruseks.

5.2.6 X-Tee turvaserveri lahtiühendamine

Testkeskkonnas on Maakleri serverirakendus lahti ühendatud X-Tee turvaserverist. Turvaserveri asemele on pandud lihtne imitatsioon, mis kuulab ja võtab vastu X-Tee päringuid, kuid tagastab alati ühesuguse vastuse. Kasutatav näidisvastus on saadud reaalsest Ehitisregistrile tehtud X-Tee päringu vastusest. Näidisvastus on osaliselt välja toodud Lisa 1.

Järgnevalt on välja toodud põhjendused, miks koormustestimise ajaks asendatakse turvaserver imitatsiooniga.

Esiteks, Maakleri süsteemi koormustestimine koormaks asjata Ehitisregistri servereid ja X-Tee infrastruktuuri, kui turvaserver süsteemiga ühendusse jätta. Ehitisregistri ja X-Tee infrastruktuuri koormustestimine ei ole antud töö eesmärk.

Teiseks, koormustestimise käigus võib selguda, et X-Tee päringud ja turvaserver on pudelikaelaks süsteemis, kui turvaserver süsteemiga ühendusse jätta. Antud teadmine ei annaks lisaväärtust katsete lõpptulemusele. X-Tee ja turvaserver on Riigi Infosüsteemi Ameti poolt arendatavad süsteemid ning nende optimeerimine ei kuulu Maakleri projekti raamesse. X-Tee ja turvaserver on vajalikud Maakleri süsteemi toimimiseks, neid ei saa süsteemist välja tõsta. Maakleri rakenduses on X-Tee päringute potentsiaalne

aeglustav mõju juba minimaalseks viidud turvaserverilt saadud vastuste salvestamisega ning salvestatud vastuste taaskasutamisega.

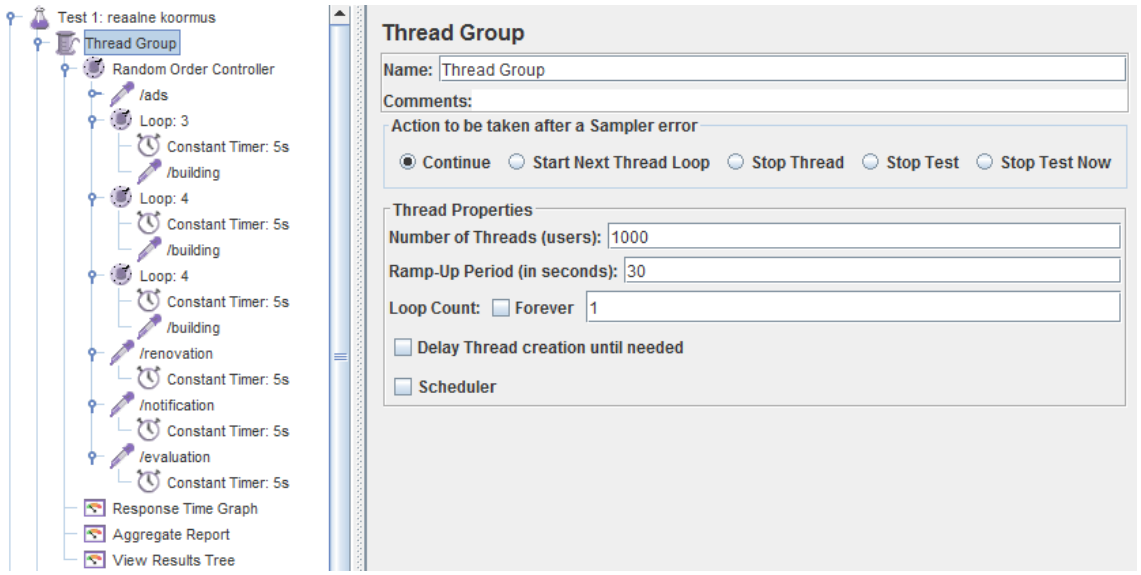
5.3 Andmebaasi algoleku taastamine

Kõik serveri punktid pöörduvad andmebaasi poole oma töötsüklis. Serveri punkt /ads ainult loeb andmebaasist, ülejäänud punktid nii kirjutavad kui ka loevad andmebaasist. Andmebaasioperatsioonid, kus kirjutatakse, muudetakse või kustutatakse andmeid, muudavad aja jooksil andmebaasi aeglasemaks (eriti *UPDATE* ja *DELETE* käsud SQL andmebaasi puhul) [15]. Seega testide läbiviimine võib negatiivselt mõjutada andmebaasi töötamiskiirust. Kui sama testi mitu korda järjest käivitada, siis tulemused võivad teineteisest erineda andmebaasi töötamiskiiruse languse tõttu. Et vältida seda mõju testide tulemustele, siis enne iga testi käivitamist viiakse andmebaas algolekusse. Seda tehakse SQL skriptiga, mis on välja toodud Lisa 2. Skript tühjendab kõik tabelid, kuhu testimise käigus kirjutatakse uusi andmeid ja sisestab algandmed, mida kasutatakse testimise käigus. Skript reindekseerib tabelid ja vabastab tabelitest üleliigse hõivatud mälu programmidega *REINDEX* ja *VACUUM ANALYZE* [16] [17]. Skript jätab puutumata tabelid, kust ainult loetakse andmeid.

5.4 Test 1: reaalse koormuse simuleerimine

Kõiki HTTP päringuid vastuvõtvaid punkte koormatakse simuleerides reaalsete kasutajate tegevusi vastavalt koormustaluvusnõuetele.

Nõuetes on ette nähtud, et rakendus peab toetama 1000 samaaegset kasutajat, seejuures 95% päringute kestusest peab jääma alla 1 sekundi. Rakenduse kasutuskogemuse alusel on testide läbiviimiseks valitud, et keskmine kasutaja teeb 15 päringut serveripoolsele rakendusele ühe kasutusseansi jooksul. Nendest päringutest 11 on suunatud punktile /building ning punktidele /ads, /notification, /evaluation ja /renovation on igaühele suunatud 1 päring. Testjuhtum on implementeeritud *JMeter*-iga (Joonis 7).



Joonis 7 JMeter implementatsioon 1. testjuhtumist

Elementidega *Constant Timer* on iga päringu vahele tekitatud 5-sekundiline paus, millega simuleeritakse reaalsete kasutajate tegevust. Androidi rakenduse kasutaja teeb oma tegevuse käigus keskmiselt iga 5 sekundi tagant ühe päringu serverile. Testjuhtumis käivitatakse 1000 lõime 30-sekundilise käivitusperioodi jooksul (Jooniselt väli *Ramp-Up Period*). Iga lõim töötab pauside tõttu vähemalt 90 sekundit. Täielik koormus, 1000 kasutajat, saavutatakse 30-sekundilise käivitusperioodi möödudes. Käivitusperioodi kasutamisega välditakse ebaloomulikku "sakilist" päringsageduse tekkimist ja imiteeritakse natuke tõetruumalt reaalsete kasutajate poolt tekitatud koormust [18]. Testis on kasutatud juhuslikku järjekorda päringute tegemisel (kasutades elementi *random order controller*), et ühtlustada koormuse jaotumist erinevatele serveri punktidele.

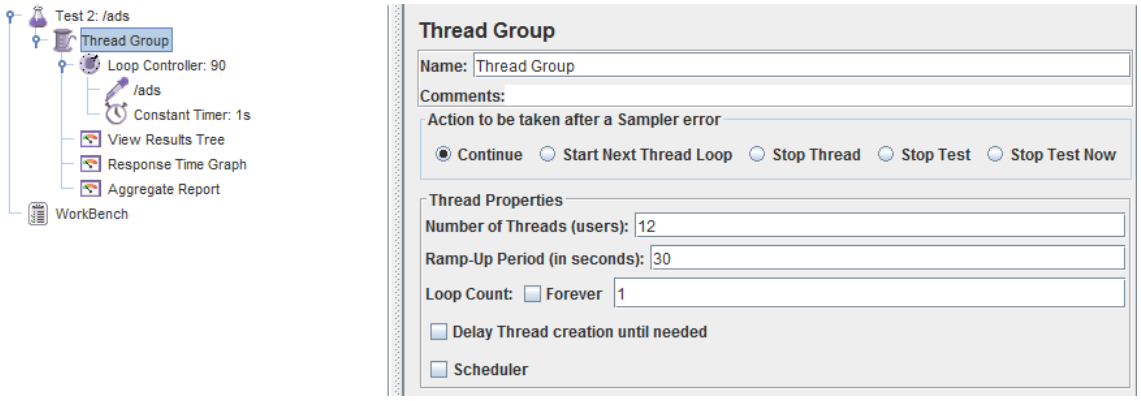
5.5 Test 2: punkti /ads koormamine

Koormatakse ainult serveri punkti /ads. Katse eesmärk on hinnata kogu süsteemist eraldiseisvalt serveri punkti /ads koormustaluvust. Teistele serveri punktidele ei tehta päringuid, et vähendada kõrvalist informatsiooni. Sihiks on kindlaks teha, kui kaua süsteemi erinevates osades kulub aega päringutele vastuste koostamiseks ja üles leida potentsiaalsed pudelikaelad serveri punkti /ads piires.

Lihtsuse huvides eeldame, et päringute kestused on ühesugused kõikides testjuhtumites. Seega võime päringu kestuse arvutusest välja jätta. Sellisel juhul 1. testjuhtumis koormatakse serveri punkti /ads maksimaalselt päringtihedusega 11,1 päringut sekundis: kui 1 lõim tekitab päringutiheduse 1 päring 90 sekundi jooksul ehk 0,011 päringut sekundis, siis tipp-perioodil töötavad samaaegselt 1000 lõime, mis annavad kokku päringtiheduse 11,11 päringut sekundis. Testide käivitamisel tuleb tegelik päringtihedus päringute kestuse tõttu mõnevõrra madalam. See ei tohiks märkimisväärselt varieeruda

erinevate testjuhtumite vahel ega tulemusi oluliselt mõjutada, kuna kõik testjuhtumid on sama koormustaluvuse nõude alusel koostatud.

Testjuhtum on implementeeritud *JMeter*-iga (Joonis 8). Iga lõim läbib 90 tsüklit (defineeritud elemendiga *loop controller*). Igas tsüklis teeb lõim ühe /ads päringu ning seejärel 1-sekundilise pausi (element *constant timer*). Sellise seadistusega tekitab iga lõim konstantse päringkoormuse 1 päring sekundis. Lõim töötab vähemalt 90 sekundit pauside tõttu. Kokku käivitatakse 12 lõime (väli *Number of Threads*) 30 sekundilise käivitusperioodi jooksul (*Ramp-Up Period*). Käivitusperioodi möödudes saavutatakse maksimaalne päringtihedus 12 päringut sekundis, mis oleks samaväärne koormus serveri punktile /ads, kui seda oli testjuhtumis 1.



Joonis 8 JMeter implementatsioon 2. testjuhtumist

Antud testis on üldine päringkoormus kogu süsteemile siiski oluliselt väiksem, kui esimeses testis. Praeguse testi tulemus näitab, kas ja mis ulatuses päringkoormus teistele serveri punktidele mõjutab punkti /ads.

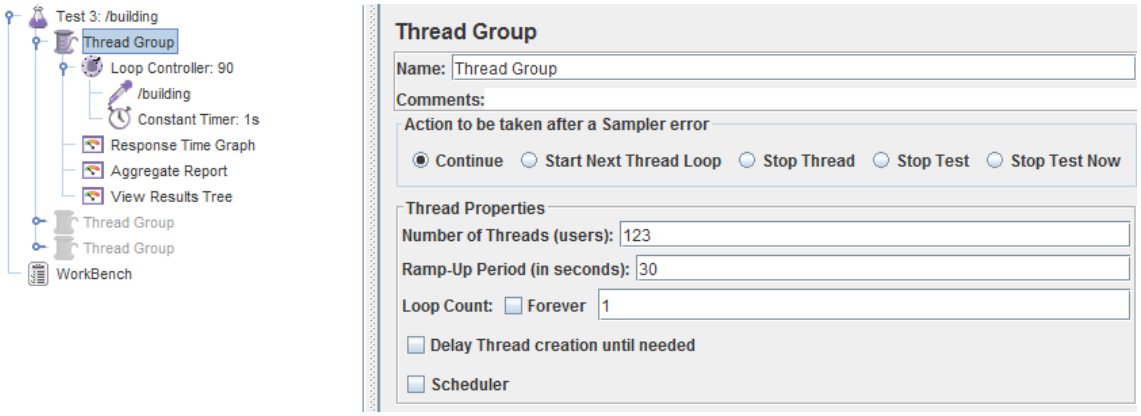
Enne testi käivitamist sisestatakse serverirakendusse kahte kohta aja kulu ülesmärkimise käsud:

1. peale päringu kättesaamist;
2. peale vastuse koostamist ja rakendusest väljasaatmist (peale *flush()* meetodi väljakutset [19]).

Aja kulu ülesmärkimine annab teavet, kui suure osa päringu kestuse ajast möödub Maakleri serverirakendusest väljaspool. Kui kogu päringu ajast kulub märkimisväärne osa väljaspool serverirakendust, siis see võib viidata probleemidele, mis asuvad väljaspool Maakleri rakendust. Näiteks Tomcat ebakorrektne seadistus, vahemälu otsasaamine või mõni muu süsteemi tööd aeglustav probleem, mis ei sõltu Maakleri rakendusest.

5.6 Test 3: punkti /building koormamine

Koormatakse ainult serveri punkti /building. Katse eesmärk on hinnata kogu süsteemist eraldiseisvalt serveri punkti /bulding koormustaluvust. Teistele serveri punktidele ei tehta päringuid, et vähendada kõrvalist informatsiooni. Sihiks on kindlaks teha, kui kaua süsteemi erinevates osades kulub aega päringutele vastuste koostamiseks ja üles leida potentsiaalsed pudelikaelad serveri punkti /building piires.



Joonis 9 JMeter implementatsioon 3. testjuhtumist

Antud test on analoogne testiga 2, kuid serveri punktile /building saab osaks suurem päringtihedus, 123 päringut sekundis (1. testis teevad serveri punktile /building 1000 lõime 11 päringut 90 sekundi jooksul ehk 123,4 päringut sekundis).

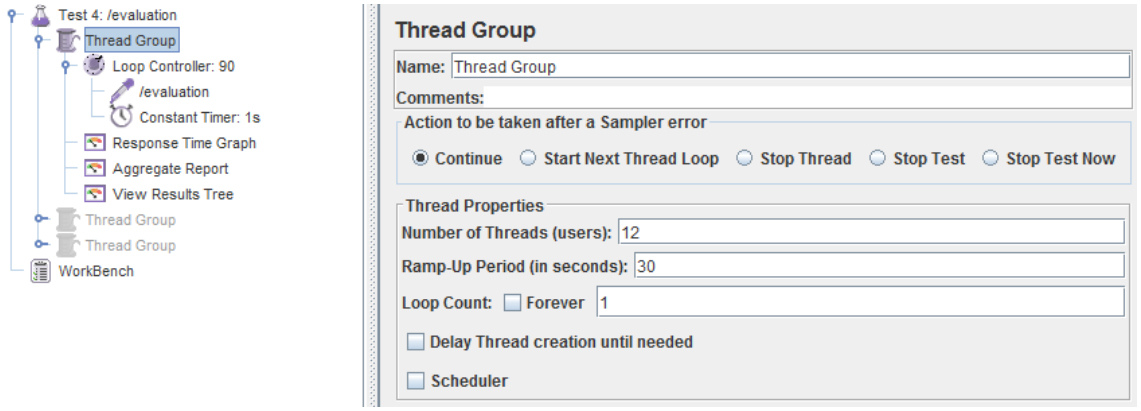
Enne testi käivitamist sisestatakse serverirakendusse viite kohta aja kulu ülesmärkimise käsud:

1. peale päringu kättesaamist;
2. peale sissetuleva päringu valideerimist;
3. peale andmebaasioperatsioone;
4. peale vastuse viimist JSON formaati;
5. peale vastuse koostamist ja rakendusest väljasaatmist (peale *flush()* meetodi väljakutset).

Ajakulu ülesmärkimine annab informatsiooni, kui palju aega kulub päringutele vastuste koostamiseks süsteemi erinevates osades ja kui palju aega kulub väljaspool Maakleri serverirakendust.

5.7 Test 4: punkti /evaluation koormamine

Antud test on analoogne testile 2. Erinevus seisneb ainult päringu sihtpunktis.



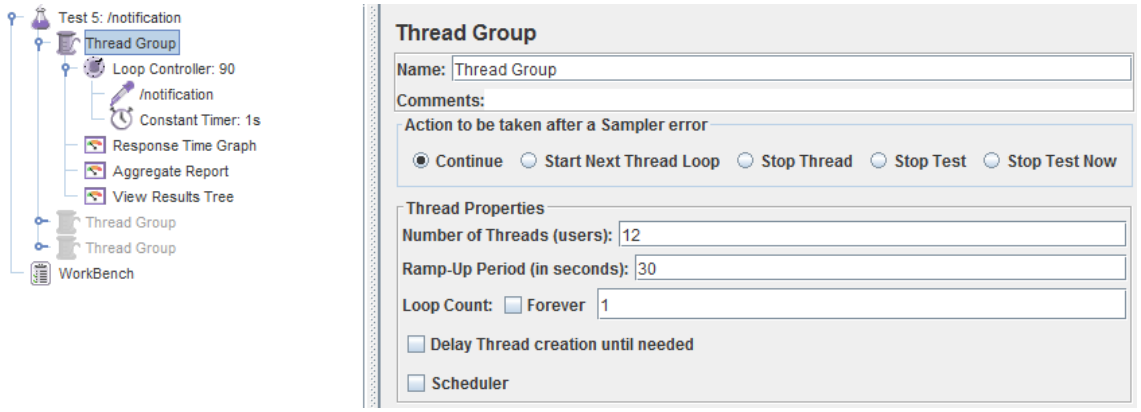
Joonis 10: JMeter implementatsioon 4. Testjuhtumist

Enne testi käivitamist sisestatakse serverirakendusse nelja kohta aja kulu ülesmärkimise käsud:

1. peale päringu kättesaamist;
2. peale sissetuleva päringu valideerimist;
3. peale andmebaasioperatsioone;
4. peale vastuse koostamist ja rakendusest väljasaatmist (peale *flush()* meetodi väljakutset).

5.8 Test 5: punkti /notification koormamine

Antud test on analoogne testile 2. Erinevus seisneb ainult päringu sihtpunktis.



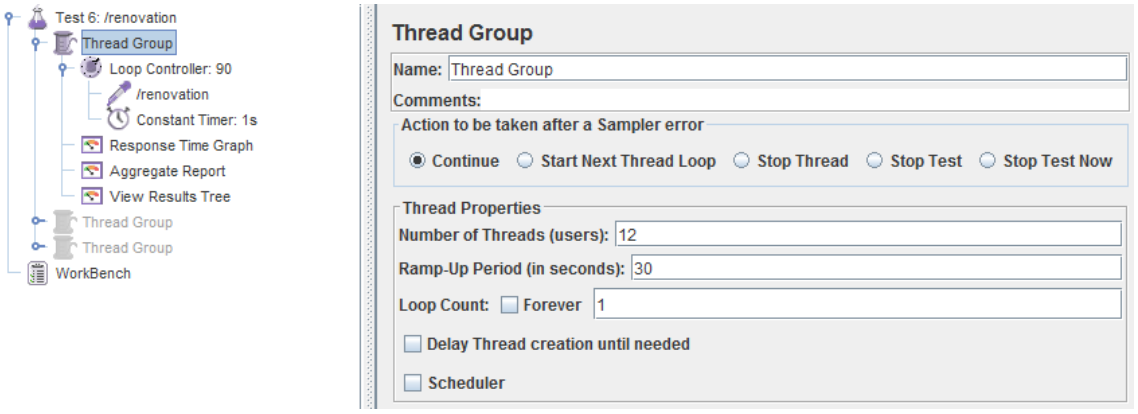
Joonis 11 JMeter implementatsioon 5. testjuhtumist

Enne testi käivitamist sisestatakse serverirakendusse nelja kohta aja kulu ülesmärkimise käsud:

1. peale päringu kättesaamist;
2. peale sissetuleva päringu valideerimist;
3. peale andmebaasioperatsioone;
4. peale vastuse koostamist ja rakendusest väljasaatmist (peale *flush()* meetodi väljakutset).

5.9 Test 6: punkti /renovation koormamine

Antud test on analoogne testile 2. Erinevus seisneb ainult päringu sihtpunktis.



Joonis 12 JMeter implementatsioon 6. testjuhtumist

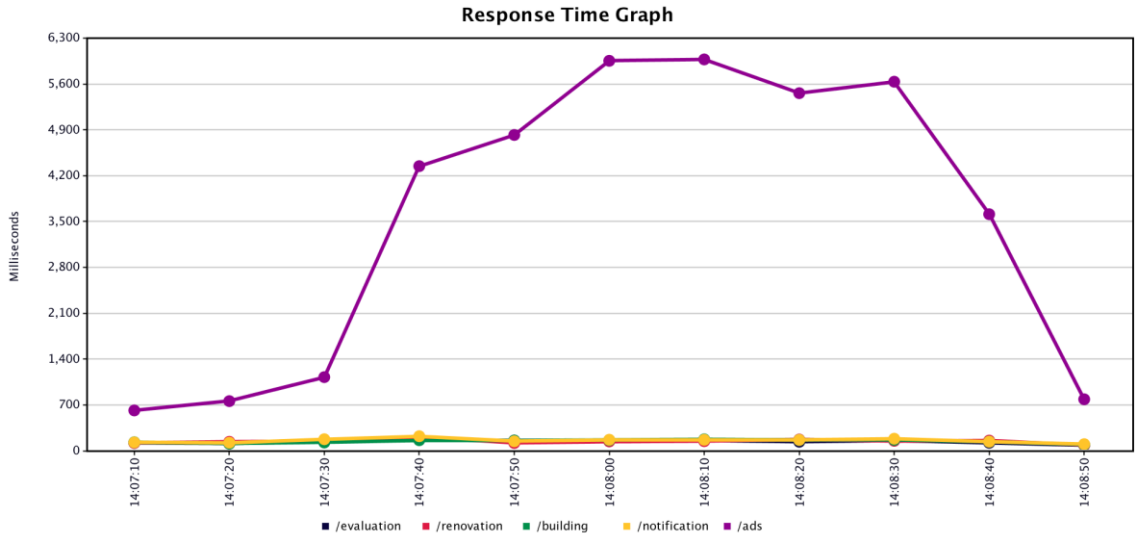
Enne testi käivitamist sisestatakse serverirakendusse nelja kohta aja kulu ülesmärkimise käsud:

1. peale päringu kättesaamist;
2. peale sissetuleva päringu valideerimist;
3. peale andmebaasioperatsioone;
4. peale vastuse koostamist ja rakendusest väljasaatmist (peale *flush()* meetodi väljakutset).

5.10 Testi 1 tulemused

Test 1 tulemustest lähtudes serveripoolne rakendus ei vasta koormustaluvusnõuetele. Rakendusel kulus /ads päringutele vastamiseks märkimisväärselt rohkem aega, kui koormustaluvusnõuetes on ette nähtud. Lõppkasutaja tajuks seda kui väga suurt viivitust kliendipoolse Androidi rakenduse avamisel.

Koormustaluvuse nõue näeb ette, et 95% päringutest peavad kestma alla 1 sekundi. Kui jätame antud katses kõikidest päringutest kõrvale aeglasemad 5%, siis selles hulgas kõige aeglasem päring oli kestusega 8,2 sekundit (Joonis 13 tulp *95% line* ja rida *ads*). Päringud ülejäänud serveri punktidele vastavad koormustaluvusnõuetele maksimaalse päringukestustega umbes 400 millisekundit, kui aeglasemad 5% kõrvale jätta.



Joonis 13 Päringute keskmised kestused koormustesti 1 vältel

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
/evaluation	1000	143	112	208	366	505	75	1830	0.00%	9.0/sec	1.3
/renovation	1000	148	112	215	378	627	74	1621	0.00%	9.0/sec	1.3
/building	11000	153	115	231	408	587	80	3716	0.00%	96.4/sec	17.1
/notification	1000	164	124	222	415	829	83	3812	0.00%	9.2/sec	1.3
/ads	1000	4152	4195	7162	8159	12011	427	13942	0.00%	9.6/sec	4513.8
TOTAL	15000	419	119	440	1917	6644	74	13942	0.00%	131.4/sec	4122.2

Joonis 14 Koormustesti 1 koondtulemused

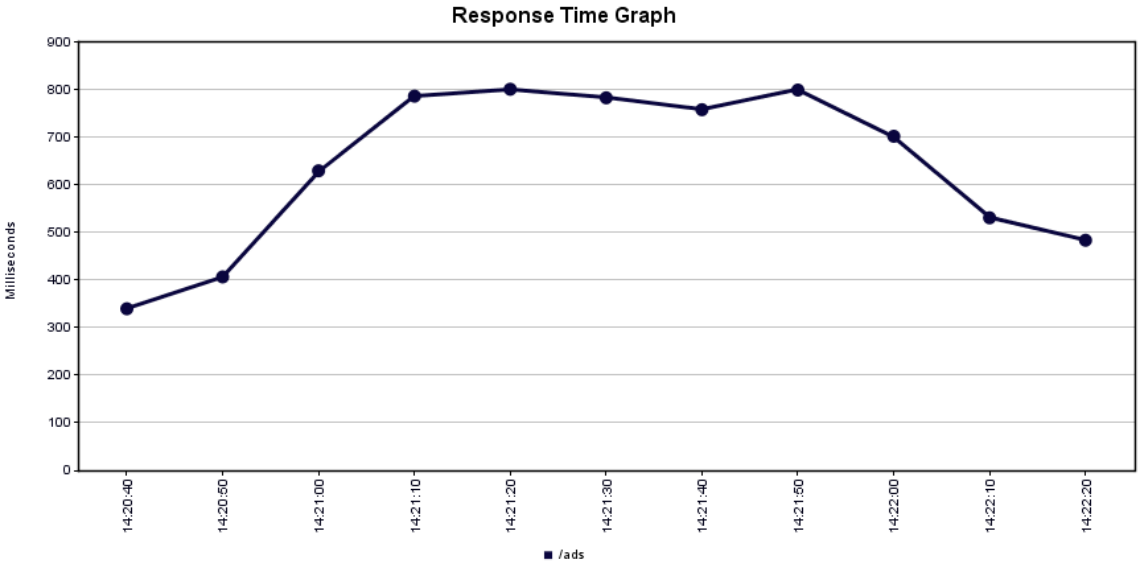
Test jooksis 1 minutit ja 59 sekundit. Päringud serveri punktile /ads võtsid rohkem aega, kui päringud ülejäänud punktidele. Maksimaalne /ads päringu aeg oli 14 sekundit ja minimaalne oli 427 millisekundit. Maksimaalsed päringu kestused ülejäänud serveri punktidele jäid vahemikku 1,5 – 4 sekundit ning minimaalsed päringu kestused oli vahemikus 74 – 83 millisekundit. Kõigi päringute puhul on mediaankeskmine väiksem kui aritmeetiline keskmine, järelikult kõrge kestusega päringuid on arvuliselt vähem kui madala kestusega päringuid. Mitte üksi päring ei ebaõnnestunud.

Katse tulemuse põhjal võime järeldada, et serveri punkt /ads ei vasta koormustaluvuse nõuetele ning tarkvarasüsteemi koormustaluvust oleks vaja suurendada enne toodangkeskkonda minekut.

5.11 Testi 2 tulemused

Testi 2 tulemuste põhjal serveripoolne rakendus ei vasta napilt koormustaluvusnõuetele. Kui jätame aeglasemad 5% päringutest kõrvale, siis kõige suurem päringu kestus on ligikaudu 1,2 sekundit (Joonis 16 tulp *95% Line*), mis ületab natuke ettenähtud 1-sekundilist nõuet. Lõppkasutaja tajuks seda kui kerget viivitust kliendipoolse Android rakenduse avamisel.

Jooniselt 10 näeme, et minimaalne päringu kestus oli 260 millisekundit ja maksimaalne oli ligikaudu 2,4 sekundit. Aritmeetiline keskmine pärnigu kestus 673 millisekundit ületab veidi mediaankeskmist, mis oli 613 millisekundit. Järelikult kõrge kestusega päringuid on arvuliselt vähem kui madala kestusega päringuid. Keskmine ühenduse loomise aeg oli 88 millisekundit (*Connect Time*, saadud JMeter-i elemendist *View Result Tree*, kompaktsuse huvides siin mitte välja toodud).



Joonis 15 päringu keskmised kestused koormustesti vältel

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
/ads	1032	673	613	1014	1191	1717	260	2353	0,00%	9,4/sec	4406,4
TOTAL	1032	673	613	1014	1191	1717	260	2353	0,00%	9,4/sec	4406,4

Joonis 16 koormustesti koonddulemused

Test jooksis 1 minut ja 49 sekundit. Jooniselt 9 näeme, et päringute kestused kasvavad lineaarselt esimesed 30 sekundit. Testi käivitusperiood, mille jooksul käivitatakse kõik testis ettenähtud 12 lõime, kestab samuti 30 sekundit. Seega lineaarne päringtiheduse

kasv kutsub esile lineaarse /ads päringute kestuse suurenemise. Siit saab järeldada, et serveri punkt /ads on tundlik päringukoormusele ning ei ole piisava koormustaluvusega.

Võrreldes Joonis 13 ja Joonis 15 näeme, et test 2 läbiviimisel olid /ads päringud keskmiselt oluliselt väiksema kestusega kui test 1 läbiviimisel. Seega päringkoormus teistele serveri punktidele suurendab samuti /ads päringute kestust. Serveri punkt /ads ei kasuta andmebaasi ega teisi ühiselt kasutatavaid ressursse. Serveri punkt /ads on ülejäänud serveri punktidega seotud ainult seeläbi, et nad töötavad sama Tomcat rakendusserveri peal ning jagavad sama virtuaalse serveri füüsilisi ressursse, nagu näiteks vahemälu ja võrgukaardi läbilaskevõimsus. Siit järeldub, et test 1 läbiviimisel /ads päringute läbilaskvust takistas ka serveri füüsiliste ressursside koormatus ja/või Tomcat rakendusserver.

Tomcat logifailidest on välja võetud aja kulu ülesmärkimisest saadud informatsioon. Iga päringu puhul on välja kirjutatud aeg, millal see saabus Maakleri serverirakendusse ja aeg, kui see sealt väljus. Aritmeetiline keskmine ajakulu nende kahe ülesmärke vahel oli 95 millisekundit ja mediaankeskmine oli 75 millisekundit. Serveri punkti /ads puhul kogu rakendusesisene aeg kulub vastuse väljasaatmise peale. Seega enamus päringu kestuse ajast, umbes 85%, kulub väljaspool serverirakendust, näiteks Tomcat-i sisend-väljund protsessides, operatsioonisüsteemi protsessides, aga ka TCP pakettide liikumisel üle interneti infrastruktuuri. Antud töö skoop ei näe ette ajakulu tuvastamist, mis kulub pakettide liikumiseks kliendi ja serveri vahel üle interneti infrastruktuuri, kuna igal kliendil võib olla erinev interneti infrastruktuur ning leitud tulemused ei pruugi peegelda reaalsete klientide kogemust. Siiski võib oletada, et üle võrgu liikumine avaldab konstantset mõju igale päringule. Katse tulemuse põhjal järeldub, et päringute läbilaskvust Maakleri süsteemis takistavad suuremas osas virtuaalserveri füüsilised ressursid ja infrastruktuur kui Maakleri serveri rakenduse implementatsioon.

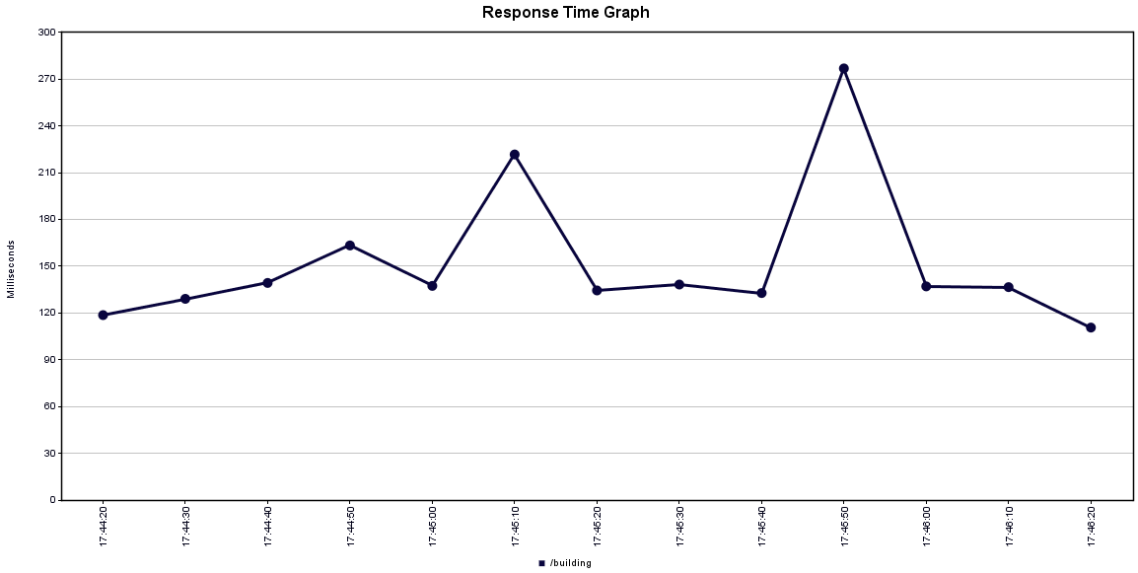
Kuna enamus päringu ajast kulub väljaspool rakendust, siis süsteemi koormustaluvuse suurendamiseks peaks eelkõige parandama infrastruktuuri. Teisisõnu serverirakenduse punktiga /ads seotud lähtekoodi muutmisest ei ole oodata suurt koormustaluvuse suurenemist süsteemis.

5.12 Testi 3 tulemused

Testi 3 läbiviimisel saadud tulemuste põhjal serveripoolne rakendus vastab koormustaluvusnõuetele. Kui jätta aeglasemad 5% päringutest kõrvale, siis suurim päringkestus on 324 millisekundit (Joonis 18 *95% Line*). Päringud serveri punktile /building enamus juhtudel ei põhjusta tajutavat viidet lõppkasutajale.

Jooniselt 12 näeme, et minimaalne päringu kestus oli 60 millisekundit, maksimaalne päringu kestus oli märkimisväärselt suur, umbes 10 sekundit. Aritmeetiline keskmine pärnigu kestus oli 156 millisekundit ja mediaankeskmine päringukestus oli 119

millisekundit. Mediaankeskmine oli väiksem aritmeetilisest keskmisest, mis tähendab, et madala kestusega päringuid oli arvuliselt rohkem. Keskmine ühenduse loomise aeg oli 71 millisekundit (*Connect Time*, saadud *JMeter*-i elemendist *View Result Tree*, kompaktsuse huvides siin mitte välja toodud).



Joonis 17 päringu keskmised kestused koormustesti vältel

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
/building	11070	156	119	219	324	704	60	9981	0,00%	81,6/sec	14,5
TOTAL	11070	156	119	219	324	704	60	9981	0,00%	81,6/sec	14,5

Joonis 18 koormustesti koondtulemused

Test jooksis 2 minut ja 15 sekundit. On märgata vähest päringute kestuste kasvu graafiku esimeses osas ja kahte teravat tõusu graafiku keskosas. Üle graafiku keskmised päringuajad on küllaltki madalad, millest järeldub, et serveri punkt /building on piisava koormustaluvusega. Teravad kasvud graafiku keskosas viitavad, et serveri punkt on natuke tundlik koormuse suhtes.

Serveri punkt /building kasutab iga päringu korral andmebaasi ja osade päringute korral X-Tee turvaserverit. Kuna turvaserveri asemel on imitatsioon, siis katse tulemused ei kajasta täpselt reaalse kasutaja kogemust. Selliste päringute osakaal, kus pöördutakse X-Tee turvaserveri poole, on viidud minimaalseks X-Tee päringute vastuste salvestamisega andmebaasi.

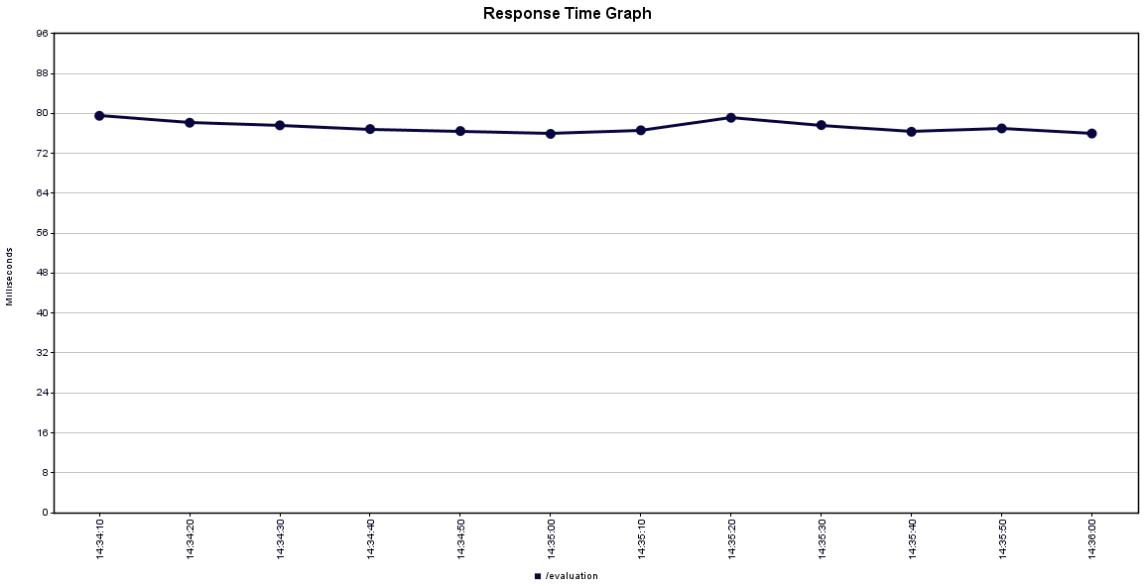
Vaadates Tomcat logifailidest ülesmärgitud ajakulu erinevates süsteemi osades, näeme, et iga etapp serverirakenduses võtab kõigest mõned millisekundid aega (vahemikus 0 – 5 millisekundit). Rakenduse siseselt kulub tühiselt väike osa kogu päringute ajast. Süsteemi koormustaluvuse suurendamiseks ei ole mõtet serverirakenduse punktiga /building seotud koodi muuta. Samuti ei ole mõtet muuta andmebaasi konfiguratsiooni, kuna andmebaasioperatsioonid on väga kiired.

Võrreldes testi 1 tulemustega (5.10), on antud testis keskmised päringukestused väga sarnased.

5.13 Testi 4 tulemused

Testi 4 tulemuste põhjal serveripoolne rakendus vastab koormustaluvusnõuetele. Jättes kõrvale aeglasemad 5% päringutest, siis pikim päringkestus on 83 millisekundit (Joonis 20 tulp *95% Line*). Päringud serveri punktile /evaluation ei põhjusta tajutavat viivitust lõppkasutajale.

Minimaalne päringu kestus oli 71 millisekundit ja maksimaalne oli kõigest 317 millisekundit. Aritmeetiline keskmine pärnigu kestus oli 77 millisekundit ja mediaankeskmine päringukestus oli samuti 77 millisekundit (Joonis 20). Keskmine ühenduse loomise aeg oli 44 millisekundit (*Connect Time*, saadud JMeter-i elemendist *View Result Tree*, kompaktsuse huvides siin mitte välja toodud).



Joonis 19 päringu keskmised kestused koormustesti 4 vältel

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
/evaluation	1080	77	77	81	83	89	71	317	0,00%	8,6/sec	1,4
TOTAL	1080	77	77	81	83	89	71	317	0,00%	8,6/sec	1,4

Joonis 20 koormustesti 4 koondtulemused

Test jooksis 2 minut ja 5 sekundit. Päringute kestused ei kasva koormustesti vältel. Võib järeldada, et serveri punkt /evaluation on samuti piisava koormustaluvusega.

Iga etapp serverirakenduses kestab vaid mõned millisekundid (vahemikus 0-5 millisekundit), kui vaadata ajakulu ülesmärkeid Tomcat-i logifailidest.

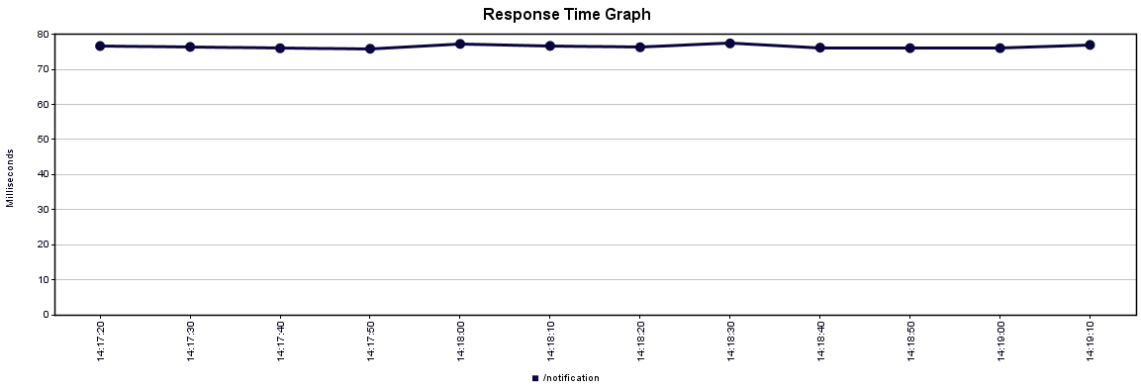
Serveri punkt /evaluation pöördub iga päringu korral andmebaasi poole. X-Tee turvaserveri poole ei pöörduta. Serveri punkt jagab mõnda Java objekti rakenduses teiste lõimedega. Võrreldes testi 1 tulemustega (Joonis 14) on antud katses madalamad

päringkestused /evaluation päringutel. Üldine päringkoormus läbi jagatud ressursside põhjustas testis 1 suuremad päringu kestused kui antud testis.

5.14 Testi 5 tulemused

Testi 5 tulemuste põhjal, serveripoolne rakendus vastab koormustaluvusnõuetele. Jättes kõrvale aeglasemad 5% päringutest, siis pikim päringkestus on 83 millisekundit. Päringud serveri punktile /notification samuti ei põhjusta tajutavat viivitust lõppkasutajale.

Minimaalne päringu kestus oli 70 millisekundit ja maksimaalne oli 120 millisekundit. Aritmmetiline keskmine pärnigu kestus oli 76 millisekundit ja mediaankeskmine päringukestus oli samuti 76 millisekundit (Joonis 22). Keskmine ühenduse loomise aeg oli 58 millisekundit (*Connect Time*, saadud JMeter-i elemendist *View Result Tree*, kompaktsuse huvides siin mitte välja toodud).



Joonis 21 päringu keskmised kestused koormustesti vältel

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
/notification	1080	76	76	81	83	90	70	120	0,00%	8,7/sec	1,4
TOTAL	1080	76	76	81	83	90	70	120	0,00%	8,7/sec	1,4

Joonis 22 koormustesti koondtulemused

Test jooksis 2 minut ja 7 sekundit. Päringute kestused ei kasva koormustesti vältel. Võime järeldada, et serveri punkt /notification on samuti piisava koormustaluvusega.

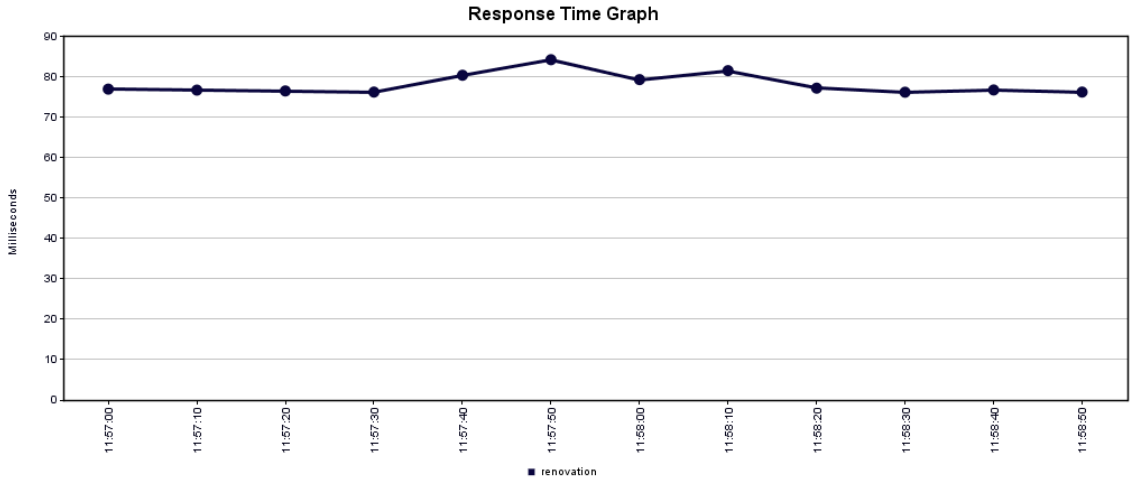
Iga etapp serverirakenduses kestab vaid paar millisekundit (vahemikus 0-5 millisekundit), kui vaadata ajakulu märkmeid Tomcat-i logifailidest.

Serveri punkt /notification pöördub iga päringu korral andmebaasi poole, X-Tee turvaserveri poole ei pöörduta. Serveri punkt jagab mõnda Java objekti rakenduses teiste lõimedega. Võrreldes testi 1 tulemustega (Joonis 14) on antud katses tunduvalt madalamad päringkestused /notification päringutel. Üldine päringkoormus läbi jagatud ressursside põhjustas testis 1 suuremad päringu kestused kui antud testis.

5.15 Testi 6 tulemused

Testi 6 tulemuste põhjal, serveripoolne rakendus vastab koormustaluvusnõuetele. Jättes kõrvale aeglasemad 5% päringutest, siis suurim päringkestus on 88 millisekundit. Päringud serveri punktile /renovation ei põhjusta samuti tajutavat viivitust lõppkasutajale Androidi rakenduses.

Minimaalne päringu kestus oli 71 millisekundit ja maksimaalne oli 211 millisekundit. Aritmeetiline keskmine pärnigu kestus oli 78 millisekundit ja mediaankeskmine päringukestus oli 77 millisekundit (Joonis 24). Keskmine ühenduse loomise aeg oli 39 millisekundit (*Connect Time*, saadud *JMeter*-i elemendist *View Result Tree*, kompaktsuse huvides siin mitte välja toodud).



Joonis 23 päringu keskmised kestused koormustesti 6 vältel

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
renovation	1080	78	77	84	88	124	71	211	0,00%	8,6/sec	1,4
TOTAL	1080	78	77	84	88	124	71	211	0,00%	8,6/sec	1,4

Joonis 24 koormustesti 6 koondtulemused

Test jooksis 2 minutit ja 4 sekundit. Koormustesti vältel on märgata minimaalset päringute kestuse kasvu. Võib järeldada, et serveri punkt /renovation on samuti piisava koormustaluvusega.

Iga etapp serverirakenduses kestab vaid paar millisekundit (vahemikus 0-5 millisekundit), kui vaadata ajakulu märkmeid Tomcat-i logifailidest.

Serveri punkt /renovation pöördub iga päringu korral andmebaasi poole, X-Tee turvaserveri poole ei pöörduta. Serveri punkt jagab mõnda Java objekti rakenduses teiste lõimedega. Võrreldes testi 1 tulemustega (5.10Joonis 14) on antud katses /renovation päringkestused madalamad. Üldine päringkoormus läbi jagatud ressursside põhjustas testis 1 suuremad päringu kestused kui antud testis.

6 Soovitused süsteemi koormustaluvuse suurendamiseks

Koormustestide tulemuste põhjal ei vasta süsteem koormustaluvusnõuetele (testi 1 ja testi 2 tulemuste põhjal, peatükid 5.10 ja 5.11). Probleem piirdub ainult päringutega, mis on suunatud serveri punktile /ads. Testi 1 ja testi 2 läbiviimisel /ads päringud ei vastanud koormustaluvusnõuetele. Testis 1 suurim päringkestus oli 8,1 sekundit 95% kiirema päringu seast, testis 2 oli sama näitaja kordades väiksem 1,2 sekundit. Testis 2 oli üldine päringkoormus süsteemile ligikaudu 10 korda väiksem kui testis 1. Seega /ads päringud muutuvad väga aeglaseks ainult siis, kui süsteemile langeb üldine suur päringukoormus. Serveri punkt /ads on koormuse suurenemise suhtes kõige tundlikum osa süsteemis.

Maakleri kasutaja tajub aeglaseid /ads päringuid viivitusena, kui ta käivitab rakenduse oma Android seadmel. On soovitav süsteemi koormustaluvuse võimet suurendada enne toodangkeskkonda üleminekut.

Koormustestide tulemustest selgus, et enamus päringu kestusest möödub väljaspool serveripoolset rakendust. Sellest järeldub, et serveripoolse rakenduse lähtekoodi optimiseerimine annab suhteliselt väikese koormustaluvuse kasvu kogu süsteemis. Katsete tulemustest tuli välja, et andmebaas on üks kiiremaid süsteemi osasid ning selle optimiseerimine oleks viljatu tegevus.

Tuginedes koormustestide tulemustele on järgnevalt välja pakutud lahendusi Maakleri süsteemi koormustaluvuse suurendamiseks. Peale soovituste implementeerimist süsteemis oleks vaja koormusteste korrata, et veenduda muudatuste positiivses mõjus. Antud soovituste implementeerimine ei kuulu käesoleva töö raamesse.

6.1 TCP ühenduste taaskasutamine

Tarkvarasüsteem Maakler ei taaskasuta TCP ühendusi. Iga HTTP päringu jaoks luuake uus ühendus. TCP ühenduste loomine on kulukas ja aeganõudev protsess [20]. Kui klient teeb serverile mitu järjestikust päringut, siis oleks võimalik iga päringu jaoks kasutada ühte ja sama TCP ühendust.

Maakleri rakenduse kasutuskogemuse põhjal, keskmine kasutaja teeb ühe kasutusseansi jooksul 15 päringut serverirakendusele ning kasutusseanss kestab 1 – 2 minutit. Kõigi nende 15 päringu jaoks võiks taaskasutada ainult ühte TCP ühendust. Selleks peaks ühendusi avatuna hoidma pikemalt, kui kestab keskmine kasutusseanss, näiteks 5 minutit. Testide tulemuste põhjal keskmine aeg, mis kulus JMeter-il ühenduse

loomiseks oli umbes 60 millisekundit. Mida suurem päringkoormus serveril oli, seda pikemaks venis ka aeg, mis kulus JMeter-il ühenduse loomiseks. 15 päringu puhul võiks ajavõit kokku olla ligikaudu 1 sekund. Kui serverile langeb suur päringukoormus (nagu testis 1), siis on oodata suuremat ajavõitu kui 1 sekund.

HTTP ühenduste taaskasutamist oleks võimalik implementeerida nii *HTTP/1.1* kui ka *HTTP/1.0 keep-alive* protokollide alusel, kuna mõlemad protokollid on toetatud süsteemi infrastruktuuri poolt [7].

6.2 Päringute ajaline hajutamine

Päringu /ads vastus sisaldab endas teavet hetkel aktiivsetest kinnisvara kuulutustest. Kliendipoolne rakendus ei saa käivituda seda informatsiooni omamata. Praegu on süsteem üles ehitatud selliselt, et Androidi rakendus teeb päringu serveri punktile /ads iga kord, kui rakendus käivitatakse. See toob endaga kaasa olukorra, kus /ads päringuid tehakse läbisegi päringutega teistele serveri punktidele. Tipp-perioodidel, kui Maaklerit kasutab maksimaalne hulk kliente korraga, muutuvad /ads päringud seetõttu väga aeglaseks.

Päringud serveri punktile /ads on aeglased nende suure mahu tõttu. Keskmine /ads päringu vastus on mahuga 500kb, seejuures vastused ülejäänud päringutele on vahemikus paarümmend kuni paarsada baiti.

Andmed, mis saadetakse /ads päringule vastuseks, ei ole väga kiiresti ajas muutuvad. Kui /ads päringu vastuse lähteandmed muutuvad, siis see ei ole kriitilise tähtsusega, et andmed jõuaksid vahetult peale uuendamist kasutajani. Seetõttu võiks kliendipoolse rakenduse käivitusmomendi ja /ads päringu tegemise omavahel lahutada. See võimaldaks /ads päringud ajaliselt lahutada ülejäänud päringutest ja seeläbi tõsta süsteemi koormustaluvust.

Androidi rakendusse võiks juurde ehitada komponendi, mis perioodiliselt, näiteks kord 2 tunni jooksul, teeks /ads päringuid ja salvestaks saadud vastuse seadme mällu. Käivitamisel rakendus üldjuhul ei tohiks /ads päringut teha, vastavad andmed peaksid seadmes juba eelnevalt olemas olema. Konkreetne kellaaeg, millal päring tehakse ühe perioodi sees, tuleks valida juhuslikult. Sellega saaks vältida korduvaid teravaid hüppeid serveri koormatuses täistundidel, kui kõik kliendid teeksid /ads päringuid. Edasiseks koormustaluvuse suurendamiseks tuleks /ads päringud paigutada ainult madala koormusega aegadele, mis oleks öö ja varahommik.

6.3 Sõnumite tihendamine

Katsete läbiviimisel kõik HTTP sõnumid süsteemis saadeti tavalise tekstina. Nii kliendipoolsele rakendusele kui ka serveripoolsele rakendusele võiks juurde ehitada komponendi, mis enne sõnumi väljasaatmist tihendaks selle ja tihendatud sõnumi kättesaamisel selle tekstilisele kujule tagasi lahti pakiks.

Katsetamise eesmärgil tihendati tavaline /ads päringu vastus *7z* formaati. Tulemus oli märkimisväärne. Esialgne, ligi 500kb suurune fail pakiti kokku 12kb suuruseks failiks, mis on ligikaudu 42 korda väiksema mahuga.

Antud soovitust oleks väljapakutavatest kõige lihtsam implementeerida ja annaks tõenäoliselt kõige suurema koormustaluvuse kasvu süsteemis.

6.4 Füüsiliste ressursside lisamine serverile

Testide 1 ja 2 tulemused viitasid asjaolule, et serveri füüsiliste ressursside liigne koormatus võis mängida rolli päringute aeglustumises. Antud tulemused ei ole piisavalt informatiivsed, et kindlaks teha, millised riistvaralised komponendid takistasid kõige rohkem süsteemi läbilaskvusvõimet.

X-Tee turvaserveri soovituslikud riistvaraparameetrid süsteemile näevad ette 2GB vahemälu olemasolu [21]. Kasutatavale serverile on eraldatud ainult 1GB vahemälu. Testimise ajal turvaserver oli käivitatud olekus, kuid Maakleri süsteemist lahtiühendatud. Peale turvaserveri nõuab süsteemi vahemälu ka Java virtuaalmasin, *PostgreSQL* andmebaas ja *Tomcat* rakendusserver. Võib oletada, et süsteem töötab vahemälu puuduses ja on sellest aeglustatud teatud määral.

Soovituslik oleks lisada serverile vahemälu vähemalt 4GB-ni.

7 Kokkuvõte

Käesolevas töös käsitleti koormustestimise protsessi ja selleks vajaminevaid vahendeid. Viidi läbi põgus uurimistöö koormustestimise protsessist. Püstitati kriteeriumid koormustestimise vahenditele antud töö vajadustest lähtudes ning valituks osutus Apache JMeter ja logimise tarkvarapakett JULI. Väljavalitud vahendeid õpiti kasutama. Uuriti Maakleri tarkvarasüsteemi seadistust, mis võib mõjutada süsteemi koormustaluvust. Koostati koormustestid vastavalt Maakleri rakendusele ja need käivitati.

Koormustestimise tulemustest selgus, et Maakleri tarkvarasüsteem ei vasta koormustaluvuse nõuetele, kuigi nõuete rikkumine ei toimunud väga suurel määral. Süsteemi probleemse osana tuvastati üks serveri punkt viiest. Tuvastati, et probleemi põhjused on nii tarkvaralised, kui ka riistvaralised. Konkreetseid pudelikaelu, mis põhjustasid süsteemi ebapiisava koormustaluvuse, ei tuvastatud, kuid see ei olnud ka antud töö eesmärk. Sellegipoolest pakuti välja riist- ja tarkvaralisi lahendusi, mis suure tõenäosusega lahendaksid süsteemi ebapiisava koormustaluvuse probleemi.

Käesoleva töö tulemused võetakse arvesse tarkvaraprojekti Maakler edasisel juhtimisel.

Kasutatud kirjandus

[1] IBM Corporation. (2006) Best practices for software development projects. [Online]. http://www.ibm.com/developerworks/websphere/library/techarticles/0306_perks/perks2.html

[2] Microsoft Corporation. (2007) Web Application Performance Testing Core Activities. [Online]. https://msdn.microsoft.com/en-us/library/bb924359.aspx

[3] Pegasystems Incorporation. (2008) Ten best practices for successful performance load testing. [Online]. https://pdn.pega.com/ten-best-practices-successful-performance-load-testing

[4] Microsoft Corporation. (2004) Measuring.NET Application Performance. [Online]. https://msdn.microsoft.com/en-us/library/ff647791.aspx

[5] Apache Software Foundation. (2016) Logging in Tomcat. [Online]. https://tomcat.apache.org/tomcat-8.0-doc/logging.html

[6] Apache Software Foundation. (2015) Component Reference. [Online]. https://jmeter.apache.org/usermanual/component_reference.html

[7] Apache Software Foundation. (2016) The HTTP Connector. [Online]. https://tomcat.apache.org/tomcat-8.0-doc/config/http

[8] PostgreSQL Global Development Group. (2016) Connections and Authentication. [Online]. http://www.postgresql.org/docs/9.3/static/runtime-config-connection.html

[9] PostgreSQL Global Development Group. (2016) pg_dump. [Online]. http://www.postgresql.org/docs/9.3/static/app-pgdump.html

[10] PostgreSQL Global Development Group. (2014) Number Of Database Connections. [Online]. https://wiki.postgresql.org/wiki/Number_Of_Database_Connections

[11] Marty Hall, "Connection Pooling," in *Core Servlets And JavaServer Pages*., 2000, ch. 18.7.

[12] David Murphy. (2006) JDBC Connection Pooling Best Practises. [Online]. www.javaranch.com/journal/200601/JDBCConnectionPooling.html

[13] Apache Software Foundation. (2016) The Tomcat JDBC Connection Pool. [Online]. https://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html#Attributes

[14] Oracle Corporation. (2015) Thread Pools. [Online]. http://docs.oracle.com/javase/tutorial/essential/concurrency/pools.html

[15] PostgreSQL Global Development Group. (2016) Routine Vacuuming. [Online]. http://www.postgresql.org/docs/9.3/static/routine-vacuuming.html

[16] PostgreSQL Global Development Group. (2016) reindex. [Online]. http://www.postgresql.org/docs/current/static/sql-reindex.html

[17] PostgreSQL Global Development Group. (2016) vacuum. [Online]. http://www.postgresql.org/docs/current/static/sql-vacuum.html

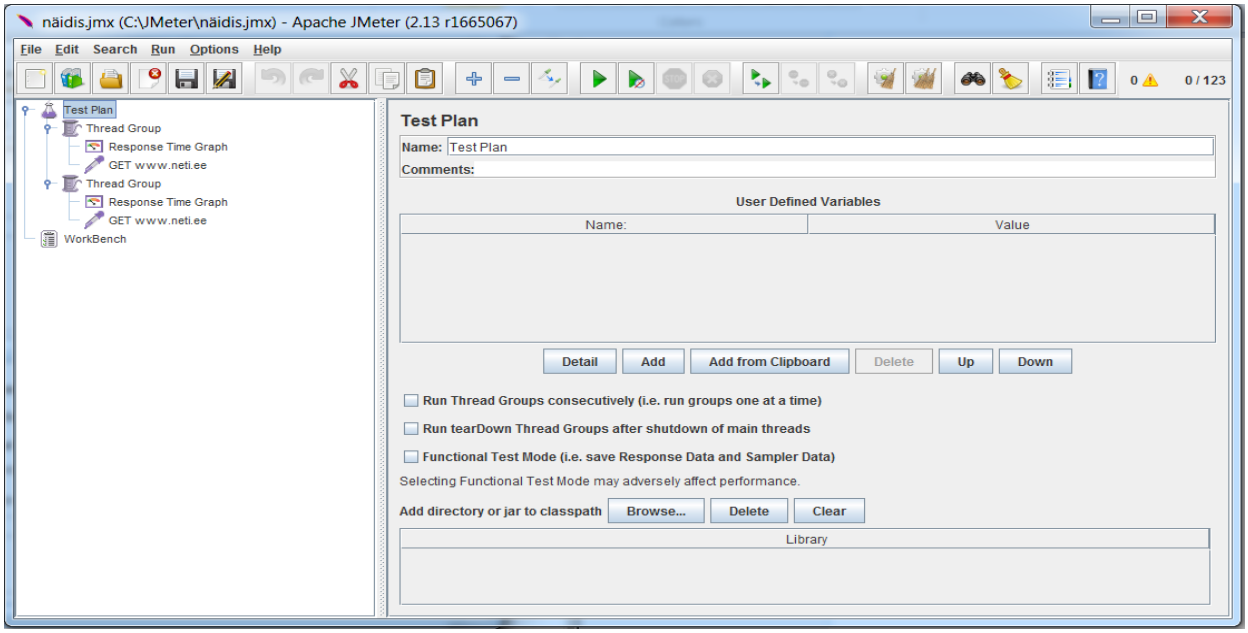
[18] Stefan Karytko. (2011) Web Load Test Ramping Best Practices: Part 1. [Online]. http://apmblog.dynatrace.com/2011/10/21/web-load-test-ramping-best-practices-1/

[19] Oracle Corporation. (2016) Class OutputStream. [Online]. https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html#flush--

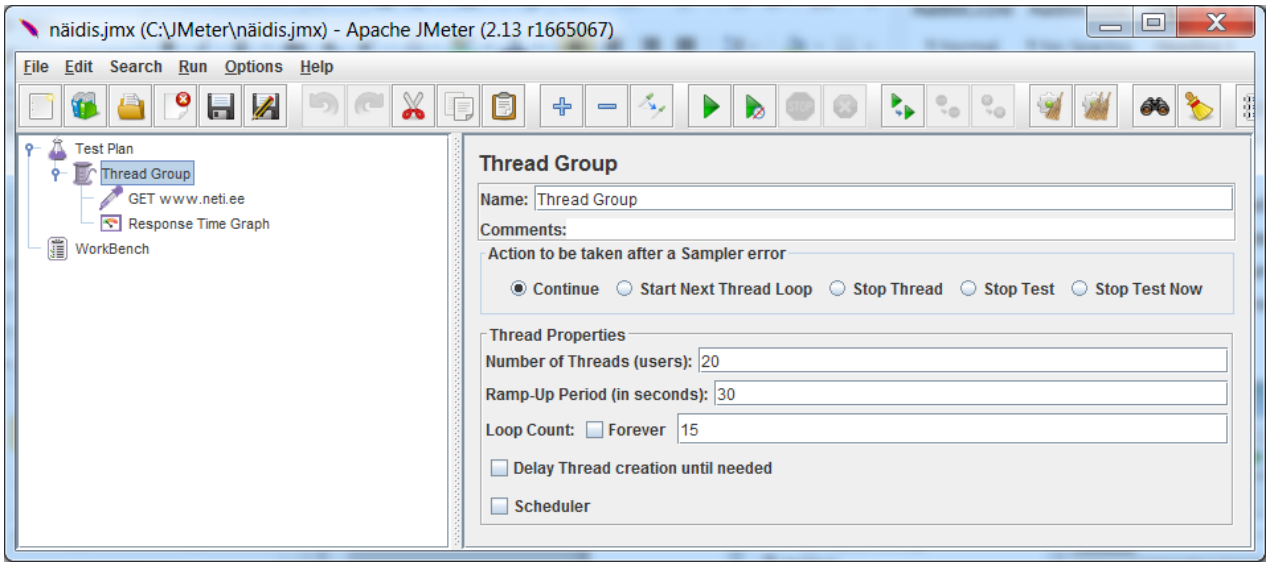
[20] Marjorie Sayer, Sailu Reddy, Anshu Aggarwal, David Gourley Brian Totty, "Persistent Connections," in *HTTP: The Definitive Guide*., 2002, Persistent Connections. [Online]. https://www.safaribooksonline.com/library/view/http-the-definitive/1565925092/ch04s05.html

[21] Riigi Infosüsteemi Amet. (2015) Turvaserveri kasutusjuhend v5 Ubuntu 14.04 LTS. [Online]. http://x-road.ee/docs/est/turvaserveri_kasutusjuhend_14.04_lts.pdf

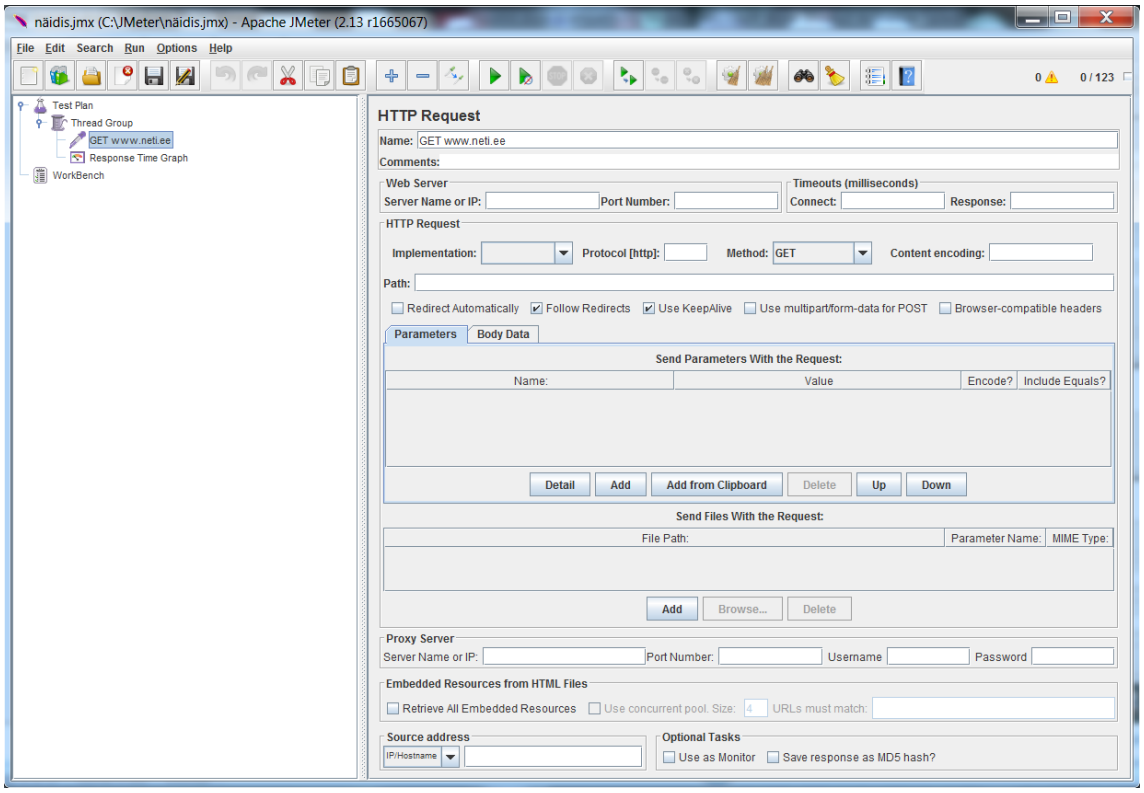
Lisa 1 – Apache JMeteri näidismaterjal



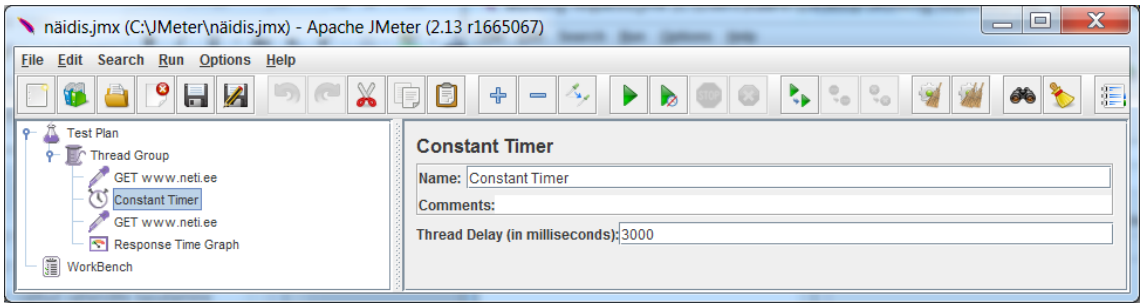
Lisa1 joonis 1 element Test Plan avatud JMeter-i graafilises kasutajaliideses



Lisa 1 joonis 2 element Thread Group avatud JMeter-i graafilises kasutajaliideses. Joonisel on määratud, et käivitamisel luuaks 20 lõime ja on seatud, et antud Thread Group elemendis kõik lõimed käivitatakse tsükliliselt 15 korda.

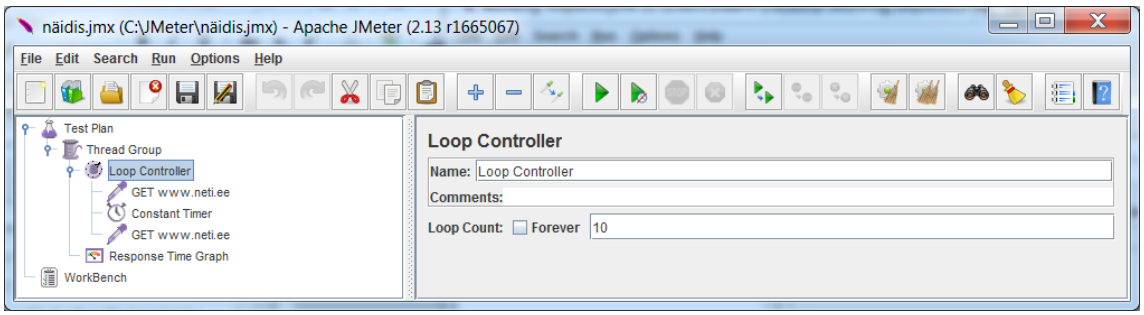


Lisa 1 joonis 3 element HTTP Request avatud JMeter-i graafilises kasutajaliideses. Joonisel on välja *path* väärtus tühjaks jäetud, sest päring tehakse otse aadressile www.neti.ee (mitte www.neti.ee/otsing vmt).

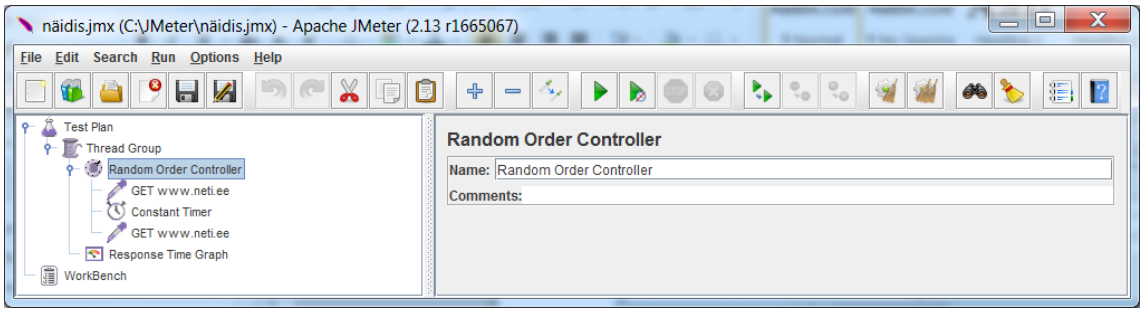


Lisa 1 joonis 4 element Constant Timer avatud JMeter-i graafilises kasutajaliideses

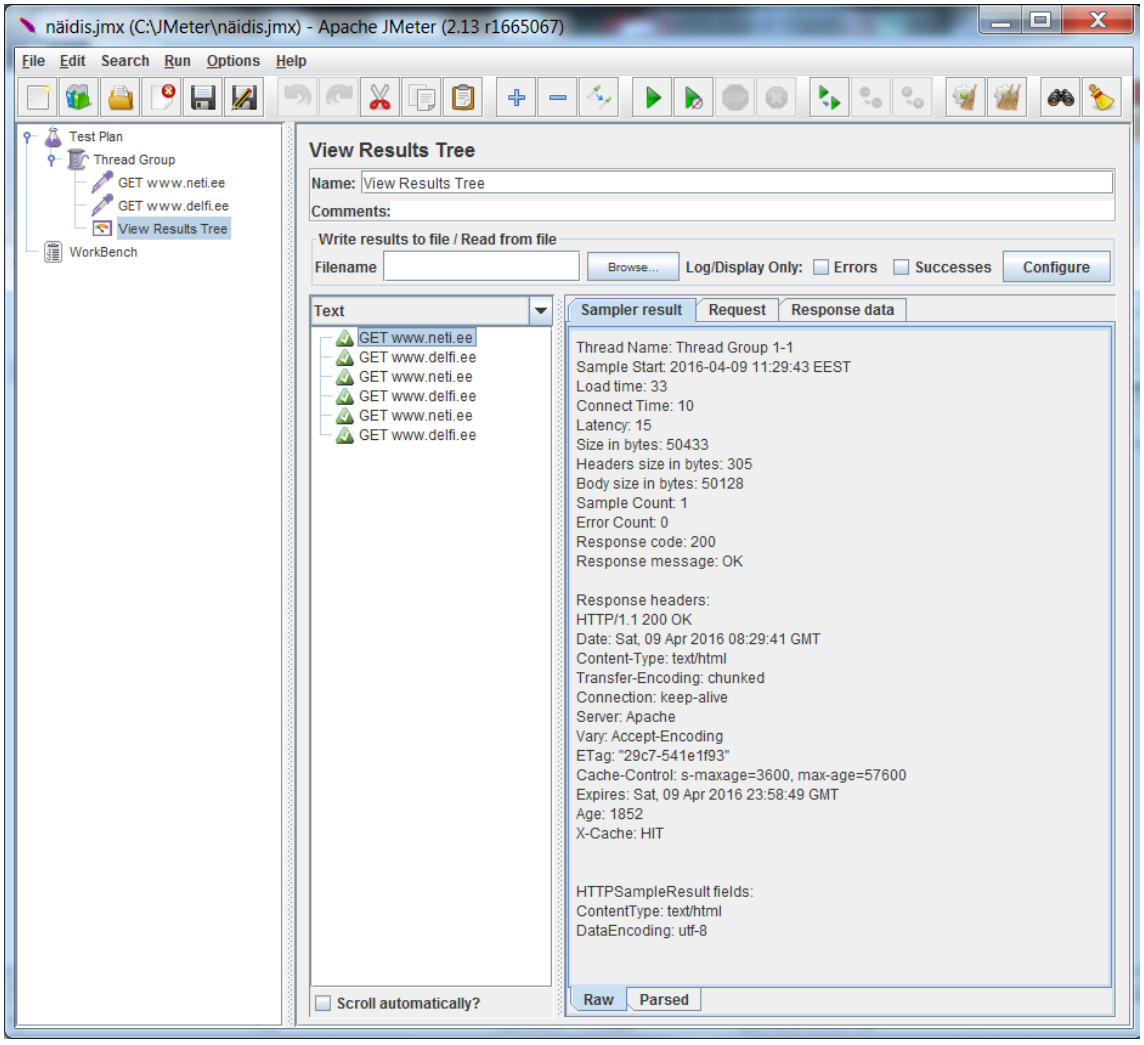
Joonisel 4 on koostatud stsenaarium, kus lõim teeb ühe HTTP päringu, ootab elemendiga *Constant Timer* määratud ajaperioodi ning seejärel teeb veel ühe HTTP päringu.



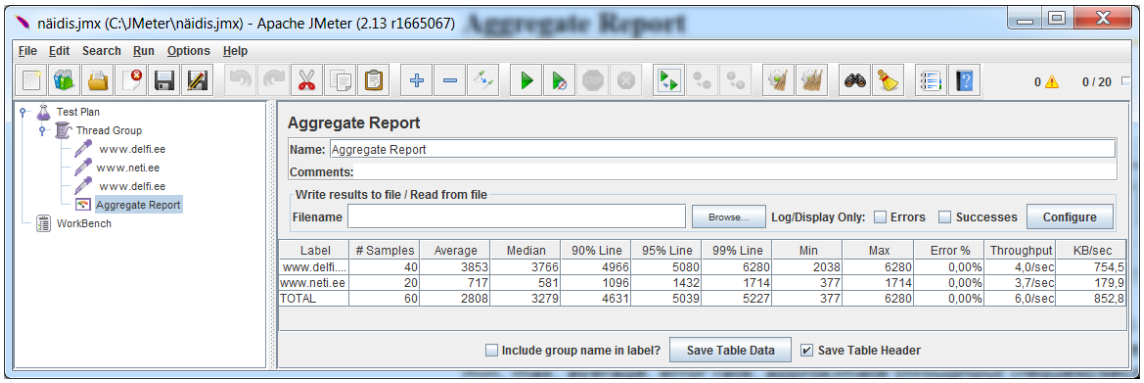
Lisa 1 joonis 5 element Loop Controller avatud JMeter-i graafilises kasutajaliideses



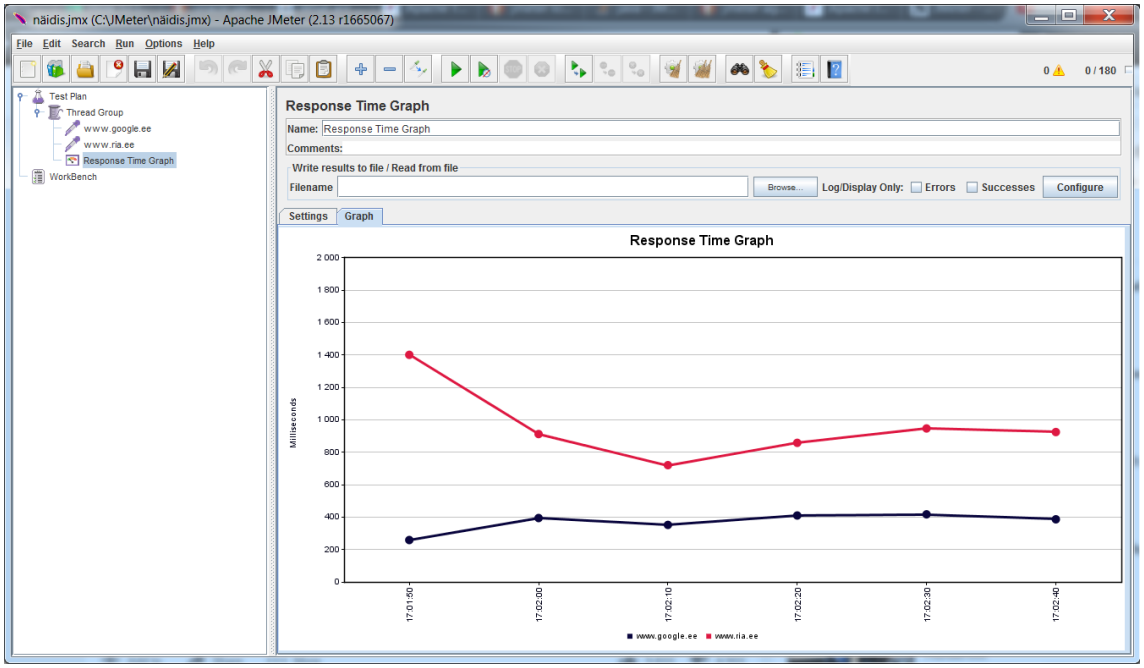
Lisa 1 joonis 6 element Random Order Controller avatud JMeter-i graafilises kasutajaliideses. Joonisel loodud stsenaariumis elemendi Random Order Controller sisse on paigutatud kaks HTTP päringut ja üks ooteperiood. Käivitamisel lõimed täidavad need käsud juhuslikus järjekorras.



Lisa 1 joonis 7 element View Result Tree avatud JMeter-i graafilises kasutajaliideses. Joonisel käivitati 3 lõime ning element View Result Tree salvestas 6 päringu tulemust, kuna iga lõim tegi 2 päringut.



Lisa 1 joonis 8 element Aggregate Report avatud graafilises kasutajaliideses. Joonisel kujutatud testjuhtumis on 3 HTTP päringut, kuid 2 erinevat sihtkohta: www.delfi.ee ja www.neti.ee.



Lisa 1 joonis 9 element Response Time Graph avatud graafilises kasutajaliideses. Joonisel kujutatud testjuhtumis on kaks sihtkohta päringutele: www.google.ee ja www.ria.ee. Graafikul on mõlema sihtkoha jaoks eraldi joon.

Lisa 2 – X-Tee turvaserveri imiteerimisel kasutatav näidisvastus

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
      ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xforms="http://www.w3.org/2002/xforms"
      xmlns:ehr="http://ehr.ee.x-road.ee/producer/"
      xmlns:xhtml="http://www.w3.org/1999/xhtml"
      xmlns:events="http://www.w3.org/2001/xml-events" xmlns:SOAP-
      ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xtee="http://x-tee.riik.ee/xsd/xtee.xsd"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"
      xmlns:xrd="http://x-road.ee/xsd/x-road.xsd"
      xmlns:exf="http://www.exforms.org/exf/1-0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xehr="http://ehr.ee.x-road.ee/producer/">
  <SOAP-ENV:Header>
    <xrd:consumer></xrd:consumer>
    <xrd:producer>ehr</xrd:producer>
    <xrd:userId></xrd:userId>
    <xrd:id>f6f9091f68f5890157e25bde0002305a6a275b4e</xrd:id>
    <xrd:service>ehr.ehitiseAndmeteParing.v1</xrd:service>
    <xrd:position/>
    <xrd:issue/>
    <xrd:authenticator>ID_CARD</xrd:authenticator>
    <xrd:userName></xrd:userName>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <xehr:ehitiseAndmeteParingResponse>
      <request>
        <ehrKood>120661354</ehrKood>
        <andmevektor>111111111111</andmevektor>
      </request>
      <response>
        <ehitiseAndmeteParingVastus>
          <ehitis>
            <version>1.0</version>
            <ehitiseAndmed>
```

```
    <ehrKood>120661354</ehrKood>
    <ehitId>3681038</ehitId>
    <verTekkAeg>2014-10-09T12:04:46.020016</verTekkAeg>
    <nimetus>Ridaelamu</nimetus>
    <kaosIdPeamine>11221</kaosIdPeamine>
    <seisund>EHITIS_SEISUND_KASUTUSEL</seisund>
    <seisundTxt>kasutusel</seisundTxt>
    <rajatisHoone>H</rajatisHoone>
    <rajatisHooneTxt>hoone</rajatisHooneTxt>
    <esmaneKasutus>2014</esmaneKasutus>
    <adsOid>EE03206555</adsOid>
    <taisaadress>Harju maakond, Keila linn, Pargi tn
        38</taisaadress>
    <katastriyksused>29601:002:0289</katastriyksused>
    <url>https://www.ehr.ee/app/link/b/120661354</url>
  </ehitiseAndmed>
  <ehitisePohiandmed>
    <omandiLiik>EHITIS_OMANDI_LIIK_KINNIS</omandiLiik>
    <omandiLiikTxt>kinnisasi</omandiLiikTxt>
    <maxKorrusteArv>2</maxKorrusteArv>
    <korgus>6.80</korgus>
    <laius>11.40</laius>
    <pikkus>60.80</pikkus>
    <ehitisalunePind>784.00</ehitisalunePind>
    <suletudNetopind>900.70</suletudNetopind>
    <mahtBruto>3610.00</mahtBruto>
    <koetavPind>900.70</koetavPind>
    <ehAlustKp>2012-11-29</ehAlustKp>
    <maakond>37</maakond>
    <omavalitsus>296</omavalitsus>
    <lahiaadress>Pargi tn 38</lahiaadress>
  </ehitisePohiandmed>
  <ehitiseAadressid>
    <aadress>
      <taisaadress>Harju maakond, Keila linn, Pargi tn
        38</taisaadress>
    <lahiaadress>Pargi tn 38</lahiaadress>
    <olekviit>
    <olek>K</olek>
    <olekTxt>kehtiv</olekTxt>
    <adrId>121836</adrId><koodaadress>
        372960000000000CV00001F6O0000</koodaadress>
  </olekviit>
```

```
….

          </ehitis>
        </ehitiseAndmeteParingVastus>
      </response>
    </xehr:ehitiseAndmeteParingResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Algusest elementidest *<xrd:consumer>*, *<xrd:userId>* ja *<xrd:userName>* identifitseeriv informatsioon eemaldatud. Kompaktsuse huvides vahelt 1007 rida välja jäetud.

Lisa 3 - Andmebaasi algolekusse viimise SQL skript

```
DELETE FROM building_prices WHERE building_price.building_id > 0;
DELETE FROM building_renovation WHERE building_renovation.id > 0;
DELETE FROM renovation_report WHERE renovation_report.id > 0;
DELETE FROM interested WHERE interested.interested_id > 0;
DELETE FROM ordereval WHERE ordereval.order_id > 0;
DELETE FROM person WHERE person.person_id > 0;
DELETE FROM full_address WHERE full_address.f_address_id > 0;

INSERT INTO full_address VALUES (1, 1, 4, 399, 1, 'Pae', '12');
INSERT INTO full_address VALUES (2, 1, 4, 399, 1, 'Pae', '43a');
INSERT INTO full_address VALUES (3, 1, 4, 399, 1, 'Pae', '43b');
INSERT INTO full_address VALUES (4, 1, 4, 399, 1, 'Murru', '12');
INSERT INTO full_address VALUES (5, 1, 4, 399, 1, 'Murru', '23');
INSERT INTO full_address VALUES (6, 1, 4, 399, 6, 'Sinilille', '5');
INSERT INTO full_address VALUES (7, 1, 4, 53, 14, 'Vase', '65');
INSERT INTO full_address VALUES (8, 1, 4, 53, 14, 'Vase', '12');
INSERT INTO full_address VALUES (9, 1, 4, 53, 14, 'Vase', '3');
INSERT INTO full_address VALUES (10, 1, 4, 53, 14, 'Vase', '44');
SELECT pg_catalog.setval('full_address_f_address_id_seq', 1, true);


INSERT INTO building_price VALUES (1, 1, 231.2300, 12.3200,
current_date);
INSERT INTO building_price VALUES (2, 2, 278.2000, 23.4000,
current_date);
INSERT INTO building_price VALUES (3, 3, 32.4300, 23.3200,
current_date);
INSERT INTO building_price VALUES (4, 4, 100.2200, 101.3200,
current_date);
INSERT INTO building_price VALUES (5, 5, 100.2200, 101.3200,
current_date);
SELECT pg_catalog.setval('building_price_building_id_seq', 5, true);

INSERT INTO building_renovation VALUES (1, 1, true, current_date, 1);
INSERT INTO building_renovation VALUES (2, 2, true, current_date, 1);
INSERT INTO building_renovation VALUES (3, 3, true, current_date, 1);
INSERT INTO building_renovation VALUES (4, 4, true, current_date, 2);
INSERT INTO building_renovation VALUES (5, 5, true, current_date, 3);
SELECT pg_catalog.setval('building_renovation_id_seq', 5, true);

INSERT INTO renovation_report VALUES (1, 'id1', 1, current_date);
INSERT INTO renovation_report VALUES (2, 'id2', 1, current_date);
INSERT INTO renovation_report VALUES (3, 'id3', 1, current_date);
INSERT INTO renovation_report VALUES (4, 'id1', 2, current_date);
INSERT INTO renovation_report VALUES (5, 'id4', 3, current_date);
SELECT pg_catalog.setval('renovation_report_id_seq', 5, true);
```

```
INSERT INTO person VALUES (1, 'Andres', 'Kala', 'email@test.ee',
'4005006', 1);
INSERT INTO person VALUES (2, 'Malle', 'Sammal', 'email2@test.ee',
NULL, 2);
INSERT INTO person VALUES (3, 'Kati', 'Ämblik', 'email3@test.ee',
NULL, NULL);
INSERT INTO person VALUES (4, 'Ivan', 'Aas', 'email4@test.ee',
'4324232', NULL);
INSERT INTO person VALUES (5, 'Ivan', 'Aas', 'email5@test.ee', NULL,
5);
SELECT pg_catalog.setval('person_person_id_seq', 5, true);


INSERT INTO interested VALUES (1, 1, 1, '{2,3}', 'kommentaar', true,
1, current_date);
INSERT INTO interested VALUES (2, 1, 2, '{2,3}', 'kommentaar', true,
1, current_date);
INSERT INTO interested VALUES (3, 2, 2, '{4,1}', NULL, true, 2,
current_date);
INSERT INTO interested VALUES (4, 2, 1, '{+}', NULL, false, 2,
current_date);
INSERT INTO interested VALUES (5, 3, 1, '{2,3}', NULL, false, 3,
current_date);
SELECT pg_catalog.setval('interested_interested_id_seq', 5, true);


INSERT INTO ordereval VALUES (1, 1, 1, 'kommentaar', 1, current_date);
INSERT INTO ordereval VALUES (2, 1, 2, NULL, 1, current_date);
INSERT INTO ordereval VALUES (3, 1, 2, NULL, 1, current_date);
INSERT INTO ordereval VALUES (4, 2, 2, NULL, 2, current_date);
INSERT INTO ordereval VALUES (5, 2, 2, NULL, 2, current_date);
SELECT pg_catalog.setval('ordereval_order_id_seq', 5, true);

VACUUM ANALYSE building_renovation;
VACUUM ANALYSE full_address;
VACUUM ANALYSE building_price;
VACUUM ANALYSE renovation_report;
VACUUM ANALYSE person;
VACUUM ANALYSE interested;
VACUUM ANALYSE ordereval;
REINDEX TABLE building_renovation;
REINDEX TABLE full_address;
REINDEX TABLE building_price;
REINDEX TABLE renovation_report;
REINDEX TABLE person;
REINDEX TABLE interested;
REINDEX TABLE ordereval;
```