TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Sander Pärn

Suurte failide puhverdamine Nortali eHealth projekti näitel

Bakalaureusetöö (9 EAP)

Juhendaja: Artjom Lind

Tartu 2019

Suurte failide puhverdamine Nortali eHealth projekti näitel

Lühikokkuvõte:

Käesolevas töös uuriti Nortali loodud haigla infosüsteemi printimislahendusega seotud mäluhalduse probleeme. Mälukasutust analüüsides tuvastati mäluleke ja sessiooni liigne suurus. Mäluhalduse efektiivsemaks muutmiseks kasutati optimeerimisvõteteks puhverdamist ja voogedastust. Töö lõpus võrreldi eelnevate ja uue printimislahenduste mälukasutust.

Võtmesõnad:

Puhverdamine, mälupõhine andmebaas, mäluhaldus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Caching large files in Nortal's eHealth project

Abstract:

In this thesis, memory management of Nortal's hospital information system's printing solution was studied. Memory leak and session's excessive size were detected. Optimisation techniques like caching and streaming were used to improve efficiency of memory management. At the end of this thesis, memory usage of different solutions were compared.

Keywords:

Caching, in-memory database, memory management

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

1. Sissejuhatus

eHealth projekt on Nortali poolt arendatav haigla infosüsteem, mis on kasutuses Tartu Ülikooli Kliinikumis, Ida-Tallinna Keskhaiglas, Valga Haiglas ja Põlva Haiglas. Süsteemi on arendatud alates 2003. aastast ning see on olnud kasutuses alates 2006. aastast. Haigla infosüsteem muudab arstide ja muu personali töö efektiivsemaks. Infosüsteemi vahendusel saab tellida uuringuid ja analüüse, lugeda eelnevate uuringute andmeid, trükkida välja kõikvõimalikke meditsiiniga seotud dokumente ning palju muud [1], [2].

Haigla infosüsteemid on kriitilised tänapäeva haiglate töös. Juhul kui infosüsteemis esineb tõrge, võib tõrke tulemusena aeglustuda haigla töö. Rakenduse ressurssid, seal hulgas ka mälu, on piiratud. Üheks põhjuseks, miks programmides tekib tõrkeid, on rakenduse ressursside ammendumine. Suure mälukasutuse tõttu rakendus aeglustub või jookseb kokku, juhul kui mälu saab otsa. Aastate jooksul on infosüsteemi kasutajate hulk ja rakenduse funktsionaalsus suurenenud, mille tõttu kasutatakse üha rohkem ressursse. Üheks suurimaks mäluressursside kasutajaks on PDF (Portable Document Format) failidega seotud toimingud. Bakalaureusetöö eesmärgiks on optimeerida PDF-failidega seotud töövoo mälukasutust. Töö on jaotatud kolmeks osaks. Esimeses osas analüüsitakse eHealth infosüsteemi PDF-failide töövoogu ja tuvastatakse mälukasutusega seotud probleemid. Teises osas otsitakse lahendusi esimeses osas leitud probleemidele. Kolmandas osas tehakse ülevaade tulemustest.

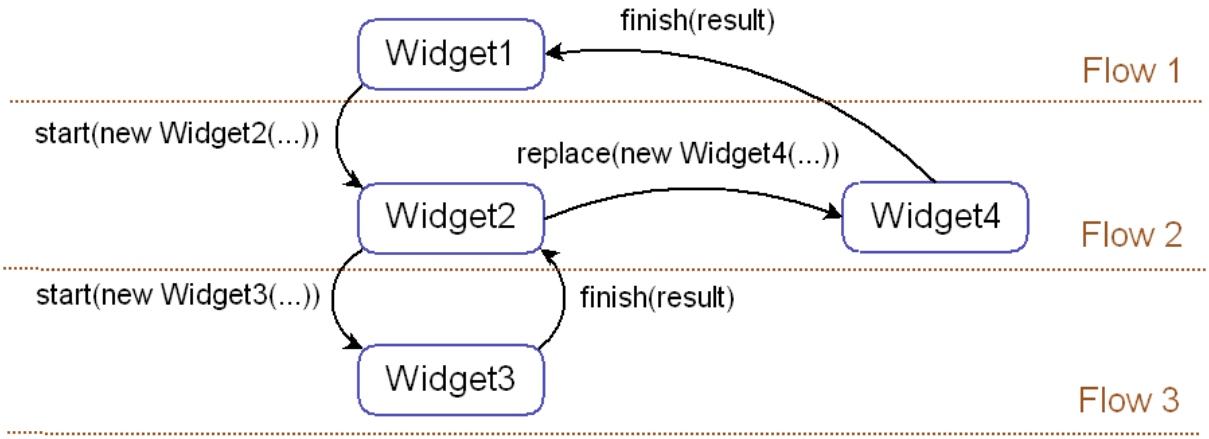
2. Olemasoleva PDF-failide töövoo analüüs

Nortali loodud haigla infosüsteemi on arendatud aastast 2003 ning koodibaas sisaldab üle miljoni koodirea. Koodibaasi vanuse tõttu ei saa väga suuri muudatusi teha, kuna see võib rakenduse lihtsasti katki teha. Selle tõttu peab arvestama eelnevalt kirjutatud funktsionaalsusega ja tehnoloogiatega. Tähtsamad kasutatavad tehnoloogiad on Aranea, Spring MVC, Spring Boot, JavaServer Pages, AngularJs, Angular, Oracle andmebaas, Apache Tomcat ja Jersey. Käesolevas peatükis käsitletakse PDF-failide töövoo arhitektuuri, seotud tehnoloogiaid ning probleemi olemust.

2.1 Aranea raamistik

Haiglainfosüsteem eHealth rakendus on suuremalt jaolt tehtud Aranea raamistikuga. Aranea on Javas kirjutatud mudel-vaade-kontroller veebiraamistik, mis pakub lihtsat viisi luua taaskasutatavaid veebirakenduse komponente ning üldist graafilise kasutajaliidese loogikat [3]. Antud veebiraamistik põhineb Java Enterprise Edition-i Java Servlet tehnoloogial [4]. Esitluskihis kasutatakse üldjuhul dünaamilise veebisisu tekitamiseks JSP (JavaServer Pages) tehnoloogiat. Aranea 1.0 loodi 2006. aastal Nortali toetusel [5]. Antud raamistikku aktiivselt enam edasi ei arendata [3].

Aranea raamistiku tähtsamateks komponentideks on *service*, *widget* ja *router*. *Service* on kontroller, mis töötleb HTTP (Hypertext Transfer Protocol) päringu andmeid ja tekitab samale päringule vastuse. *Widget* on graafilise kasutajaliidese komponent, millel on *service*-i omadused ning on olekuga. *Router* on komponent, millel on palju alamkomponente ning mille ülesandeks on suunata päring õigesse alamkomponenti, võttes arvesse päringu sisendandmed [6].



Joonis 1. Aranea graafilise kasutajaliidese komponentide voo illustratsioon [7].

Graafilise kasutajaliidese komponentide voogu saab juhtida kolmel erineval viisil. Voogu on võimalik lisada uus komponent, vahetada komponent teise vastu välja või lõpetada käimasoleva komponendi töö. Graaflise kasutajaliidese komponentide voogu on hea ettekujutada pinumäluna. Joonis 1 näites alustatakse voogu *Widget1*-st. Edasi tekitatakse uus komponent ning töö voog jätkub *Widget2*-s. Pärast seda on võimalik tekitada järgnev komponent või vahetada olemasolev komponent teise vastu välja. Kui tekitada uus komponent juurde, siis jätkub töö voog *Widget3*-s. Kõik eelnevad komponendid hoitakse mälus isegi, siis kui ainult üks on aktiivne. Käimasoleva komponendi tööd on võimalik lõpetada ning tagasi liikuda eelnevasse komponenti ehk *Widget2*-e. Tagasi liikudes eemaldatakse komponent mälust. Kasutajaliidese poolelt vaadates suunatakse kasutaja esialgu pealehele, kus kasutaja saab edasi liikuda järgnevatele kuvadele ning siis ka tagasi liikuda.

Kuna graafilise kasutajaliidese komponendid on olekuga ning nende vahel on võimalik edasi ja tagasi liikuda, siis rakendus peab hoidma mälus kõiki käimasolevaid komponente. Antud veebiraamistiku arhitektuur lisab suurt koormust serveri mälule ning arendaja peab selle tõttu jälgima, et olekuga komponendid talletaksid ainult vajaminevaid andmeid ning võimalikult efektiivselt. Samuti peab arendaja jälgima, et voog graafiliste komponentide vahel oleks lihtne ning ei sisaldaks ringsõltuvust, muidu võib tekitada lõpmata palju komponente.

2.2 eHealth rakenduse infrastruktuur

eHealth rakendus on jaotatud mitmeks mooduliks, mis on omakorda jaotatud kaheks: põhimoodulid ja abistavad moodulid ehk mikroteenused. Rakenduse põhifunktsionaalsus on põhimoodulites, mida iseloomustab pikk ajalugu, tohutu funktsionaalsuste hulk ja monoliitne arhitektuur. Abimoodulid on koodiridade poolest väikesed programmid, mis tegelevad väga spetsiifilise funktsionaalsusega. Igal moodulil on erinev ülesanne. Näiteks põhimoodul *reception* tegeleb patsiendi vastuvõttudega ning põhimoodul *treatment* tegeleb patsiendi ravimisega seotud toimingutega. Abimooduliks on näiteks PDF-faili tekitamise moodul. Kõik moodulid pakitakse WAR (Web Application Archive) failideks ning käidatakse Apache Tomcat serveris. Moodulid saavad omavahel suhelda REST (Representational State Transfer) teenuste vahendusel.

2.3 Mäluhalduse probleemi avastamine

Probleemide õigeaegseks avastamiseks tuleb rakendust püsivalt seirata. Java rakenduste mälukasutuse seiramiseks on mõistlik kasutada kuhja analüüsijat. Antud juhul kasutati Eclipse Memory Analyzer-it.

Class Name	Shallow Heap	Retained Heap	Percentage
org.araneaframework.framework.router.StandardTopServiceRouterService @ 0x64a237bc0	72	555 798 416	26,04%
java.util.Collections$SynchronizedMap @ 0x64a237c88	32	555 798 344	26,04%
java.util.LinkedHashMap @ 0x64a237ca8	56	555 798 312	26,04%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0x67cd4bd18	104	5 007 600	0,23%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0x67c23eb28	104	3 577 408	0,17%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0x671b09bf8	104	3 500 520	0,16%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0x65dde4e58	104	3 387 776	0,16%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0x670cb7878	104	2 484 344	0,12%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0x6bd1c8d98	104	2 354 864	0,11%

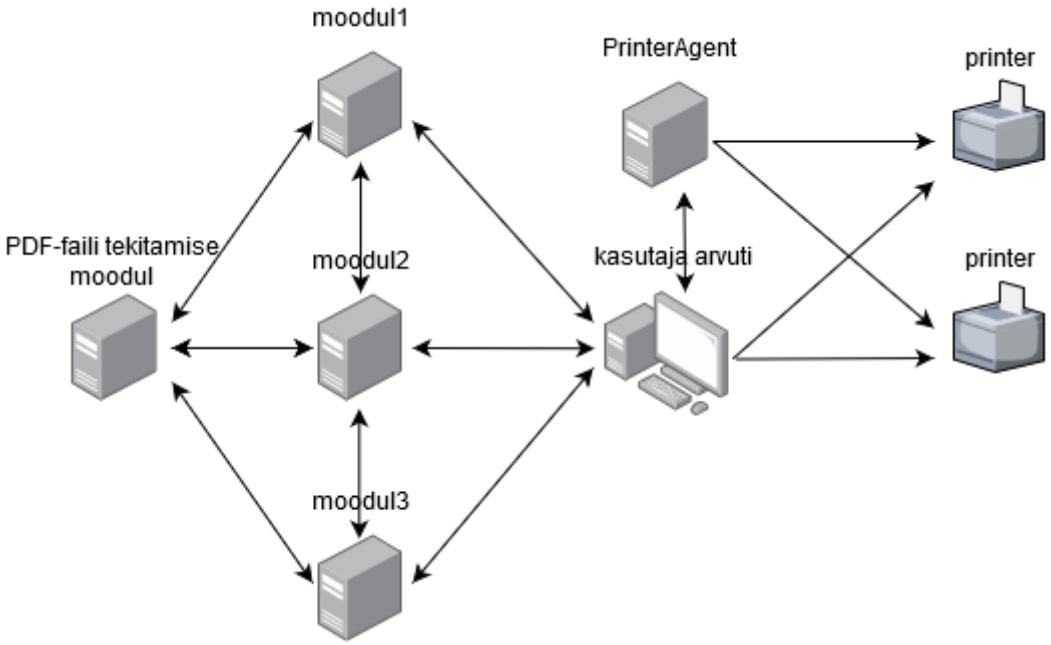
Joonis 2. Rakenduse kuhja analüüs.

Joonisel 2 oleva kuhja analüüsi kokkuvõttes on näha, et umbes 556 MB ehk 26% rakenduse kuhja mälust kasutab *StandardTopServiceRouterService*, mille alamkomponentideks on mitu *PdfPrintWidget*-it. *PdfPrintWidget* on graafilise kasutajaliidese komponent, mille

ülesanne on kasutajale PDF-faili sisu näitamine brauseris ning selle faili saatmine printerisse.

2.4 PDF-failide voog rakenduses

Patsiendiga seotud uuringud, diagnoosid, ravitulemused ja palju muud tuleb haigla personali poolt protokollida, kasutades selleks haigla infosüsteemi. Samuti väljastatakse osa infost ka patsiendile paberkujul. Selleks suundub haigla töötaja pärast vajalike andmete sisestamist veebileheküljele, kus on võimalik kõigepealt näha vastloodud PDF-faili sisu. Kui töötaja on veendunud, et failil on õiged andmed, siis saadetakse see edasi printerile. Seda kõike on võimalik teha vaid mõne nupuvajutusega.



Joonis 3. Lihtsustatud illustratsioon PDF-faili vooga seotud osapooltest.

Kasutaja vaatepunktist on kõik väga lihtne. Tagataustal toimub aga palju enamat. Joonisel 3 on välja toodud PDF-faili vooga seotud osapooled. Kui kasutaja töövoog nõuab dokumendi printimist, siis kõik algab ühes põhimoodulis vajalike andmete kokkukogumisega, mille järel tehakse REST päring PDF-faili tekitamise moodulile, andes sisendiks dokumendi struktuuri malli ning andmed, mida tuleks dokumenti kirjutada. PDF-faili tekitanud moodul tagastab dokumendi baitide massiivina. Vajadusel tekitatakse mitu erinevat PDF-faili ning liidetakse need kokku. Edasi tekitatakse *PdfPrintWidget* ning antakse sisendiks vastloodud dokument baitidena. Antud sisend talletatakse klassi isendimuutujana:

```
public class PdfPrintWidget extends BaseWidget {
  private byte[] pdf;
  private String documentContent;
  // ...
}
```

Edasi tekitatakse esitluskihis kuva, mille üheks osaks on HTML (Hypertext Markup Language) Inline Frame element ehk iframe. Tegemist on elemendiga, mille abil on võimalik HTML leheküljel sisse põimida teine HTML lehekülg. Antud elemendi üheks atribuudiks on src, mille parameetriks on URL. *PdfPrintWidget* tekitab URL-i, mida pärides tagastatakse PDF-fail. Vastav URL lisatakse iframe-i src parameetriks. Loodud lehekülg saadetakse edasi kasutajale.

Kasutajal on võimalik tekitatud dokument printida. Selleks on kaks erinevat võimalust: brauseri vahendusel printimine või PrinterAgent [8] rakenduse vahendusel printimine. Selleks tekitatakse ajutine koopia olemasolevast PDF-failist ning listatakse JavaScript printimise skript, mille tulemusena dokumenti avades tekib printimise dialoog. Kui kasutaja prindib brauseri vahendusel, siis avaneb brauseri printimisdialoog, kus kasutaja peab valima sobiva printeri. Kui aga kasutatakse PrinterAgent lahendust, siis tekitatakse failist koopia, Base64 kodeeringus sõne, ning talletatakse isendimuutujana. Edasi saadetakse see konfigureeritud printerile. PrinterAgent-i eeliseks on see, et kasutaja ei pea tegema valikut mitme printeri vahel vaid õige printer on eelnevalt konfigureeritud. See tähendab, et kasutaja saab printida ühe nupuvajutusega. Loodud dokumendid on ajutised, kuna graafilise kasutajaliidese komponentide voogu mööda tagasiminnes kaob viide komponendile.

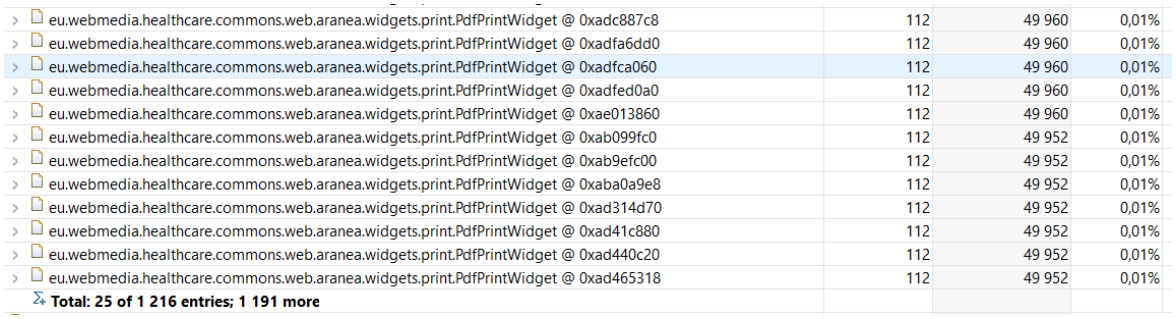
2.5 Probleem 1: mäluleke

Rakenduses on tuhandeid komponente. Ühe väga spetsiifilise komponendi mälukasutus on ebaharilikult kõrge. Kuna rakenduses tekitatud PDF-failid on ajutised, siis tekib küsimus, kas rakenduses eksisteerib mäluleke.

Sellele hüpoteesile on võimalik vastata lihtsa katse läbi. Esiteks tuleb käivitada server nullist. Järgmisena tuleb tekitada uus PDF-fail, liikuda vastavale kuvale ning siis sulgeda kuva. Viimast protsessi tuleb korrata N korda. Kui esineb mäluleke, siis kõik N faili peaksid mälus eksisteerima isegi siis, kui kasutaja on oma tegevuse lõpetanud.

Antud hüpoteesi testiti katsega, mis hõlmas 1200 PDF-faili tekitamise automatiseeritud viisil, kasutades Selenium veebilehitseja pluginat. Plugina vahendusel tuli üks kord kasutajaliideses voog läbida ning siis sai seda protsessi korrata nii palju kui vaja. Kasutajaliideses tekitati 50 KB PDF-fail, mille järel mindi tagasi algse kuva peale. Seda

protsessi korrati 1200 korda. Pärast katset tehti kuhja analüüs Eclipse Memory Analyzer-iga.



eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xadc887c8	112	49 960	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xadfa6dd0	112	49 960	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xadfca060	112	49 960	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xadfed0a0	112	49 960	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xae013860	112	49 960	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xab099fc0	112	49 952	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xab9efc00	112	49 952	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xaba0a9e8	112	49 952	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xad314d70	112	49 952	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xad41c880	112	49 952	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xad440c20	112	49 952	0,01%
eu.webmedia.healthcare.commons.web.aranea.widgets.print.PdfPrintWidget @ 0xad465318	112	49 952	0,01%
Total: 25 of 1 216 entries; 1 191 more			

Joonis 4. *PdfPrintWidget* mälu kasutuse analüüs. Joonisel on näidatud ainult väike osa tekitatud komponentidest.

Joonisel 4 on näha, et kõik 1200 *PdfPrintWidget*-it on mälus alles. See tähendab, et rakenduses esineb mäluleke. Mäluleke on tahtmatu mälutarbimise viis, mille tõttu ei õnnestu rakendusel vabastada eraldatud mälu, kui seda enam vaja ei lähe [9]. Kuna nüüd on teada, et rakenduses esineb mäluleke, siis tekib küsimus, mis on selle tagajärjed.

Rakenduse mälu on piiratud ressurss. Java objekte luuakse Java kuhja peale, mille suurust on võimalik määrata *-Xmx* ja *-Xms* argumentidega, kus *mx* on kuhja ülempiir ja *ms* on kuhja algne suurus. Juhul kui Java kuhi hakkab täis saama, siis mälukoristuse peale kulutatakse palju aega, mille tõttu võib rakendus ajutiselt hanguda. Juhul kui mälukoristaja ei suuda piisavalt palju mälu vabastada, hakatakse lehekülgi saalima, mis muudab rakenduse veelgi aeglasemaks. Kui saalida enam ei õnnestu, siis rakenduses tekib *OutOfMemoryError* erind, mille tõttu rakenduse töö peatatakse.

Kontrolliks prooviti tekitada lõpmatult palju PDF-faile. Prügikoristuse täpsema info logi genereerimiseks lisati Java argumenditesse *-verbose:gc*. Iga prügikoristuse etapi lõpus genereeriti logisse info:

[GC (Allocation Failure) 782076K->264169K(1959424K), 0.0540107 secs]

Antud prügikoristuse etapiga peatati rakenduse töö 0,06 sekundiks, mis on Java rakenduste puhul täiesti ootuspärane. Kui aga mälu hakkas täis saama, siis prügikoristus toimus tihedamini ning suurenes ka prügikoristuse peale kulunud aeg:

[Full GC (Ergonomics) 1952162K->1950354K(2022400K), 1.5511253 secs]

[Full GC (Ergonomics) 1952162K->1949001K(2022400K), 5.4081949 secs]

[Full GC (Ergonomics) 1951844K->1949263K(2022400K), 1.5470660 secs]

[Full GC (Ergonomics) 1951844K->1949127K(2022400K), 10.2255405 secs]

Lõpuks tekkis rakenduses *OutOfMemoryError* erind:

Critical exception occured:

java.lang.OutOfMemoryError: GC overhead limit exceeded

eHealth rakenduse puhul töödeltakse kõiki erindid, mille tõttu rakendust ei peatata. Rakendus muutus aina aeglasemaks kuni hangus täielikult. Tegemist on ekstreemse näitega, kuid teoorias siiski võimalik, kui mälukasutust ei jälgita.

2.6 Probleem 2: suur sessioon

Peatükis 2.4 toodi välja, et iga *PdfPrintWidget*-i puhul tekitatakse üks PDF-fail. PrinterAgent-iga printimise puhul tekitatakse samast failist Base64 kodeeringus koopia, mis on mälukasutuse poolest suurem kui originaalne PDF-fail. Mõlemad failid talletatakse isendimuutujatena. Aranea raamistiku puhul tähendab see suurte failide talletamist veebisessioonis.

Kui näiteks rakendust kasutab 10 kasutajat ning iga üks tekitab 10 MB suuruse PDF-faili, siis talletatakse kõigi nende PDF-failid ja Base64 kodeeringus koopia kasutajate sessioonides. Ühe 10 MB suuruse faili talletamine baitide massiivina hõivab 10 MB rakenduse mälu. Base64 kodeeringus sõne pikkus on $4\left\lceil\frac{n}{3}\right\rceil$, kus n on baitide arv [10]. Seega 10 MB suuruse faili suurus Base64 kodeeringus on 13,3 MB. Kuna Java sõne üks täht võtab kaks baiti ruumi mälus, siis Base64 sõne hõivab mälus 26,6 MB. Kokku hoitakse kümne kasutaja poolt sessioonis umbes 367 MB jagu mälu. See aga tähendab, et mida rohkem on kasutajaid, seda suuremad on sessioonid. See tekitab probleeme skaleerivusega ning lisab suuremat koormust prügikoristusele, mille tõttu muutub rakendus aeglasemaks.

3. Realisatsioon

Eelnevas peatükis käsitleti eHealth rakenduse PDF-failide vooga seotud probleeme. Käesolevas peatükis keskendutakse nende probleemide lahendamisele. Peatüki alguses leitakse viis, kuidas vabaneda *PdfPrintWidget*-i isendimuutujatest, mis talletavad suuri faile. Leitud viisi põhjal luuakse uus ja parem lahendus. Lahenduse loomisel arutatakse ka tähtsamate koodilõikude üle.

3.1 Failide eraldamine *PdfPrintWidget*-i isendimuutujatest

Peatükis 2.4 väljatoodud PDF-failide töövoo kirjelduses oli mainitud, et PDF-faili tekitav moodul tagastab põhimoodulile dokumendi baitide massiivi kujul, mis talletatakse *PdfPrintWidget* isendimuutujana. Baitide massiivina talletamine isendimuutujasse on mugav ja lihtne, kuid lisab koormust mälukasutusele. Kuna mälukasutuse probleemiks on suurte failide talletamine isendimuutujatena, siis oleks tarvis nendest vabaneda. Selleks on vaja muuta suhtlus moodulite vahel selliseks, et baitide massiivi pole vaja talletada põhimoodulites.

Üks võimalus oleks massiv saata kliendi brauserisse, ilma et seda isendimuutujasse talletatakse. Selline faili saatmise viis lahendab töös püsitatud eesmärgid, kuid on ebaefektiivne ning keeruline töödelda. Palju efektiivsem lahendus on voogedastamine (ingl. keeles *streaming*). Voogedastus on tükkhaaval andmete saatmise viis ühest punktist teise. Java rakendustes kasutatakse voogedastamiseks I/O Streams-e. Voogedastamise puhul on eeliseks see, et objekti töödeltakse tükkhaaval, mille tõttu on võimalik töödelda ka nii suuri faile, mis ei mahu rakenduse mällu. Baitide massivi puhul tuleb hoida kogu faili mälus.

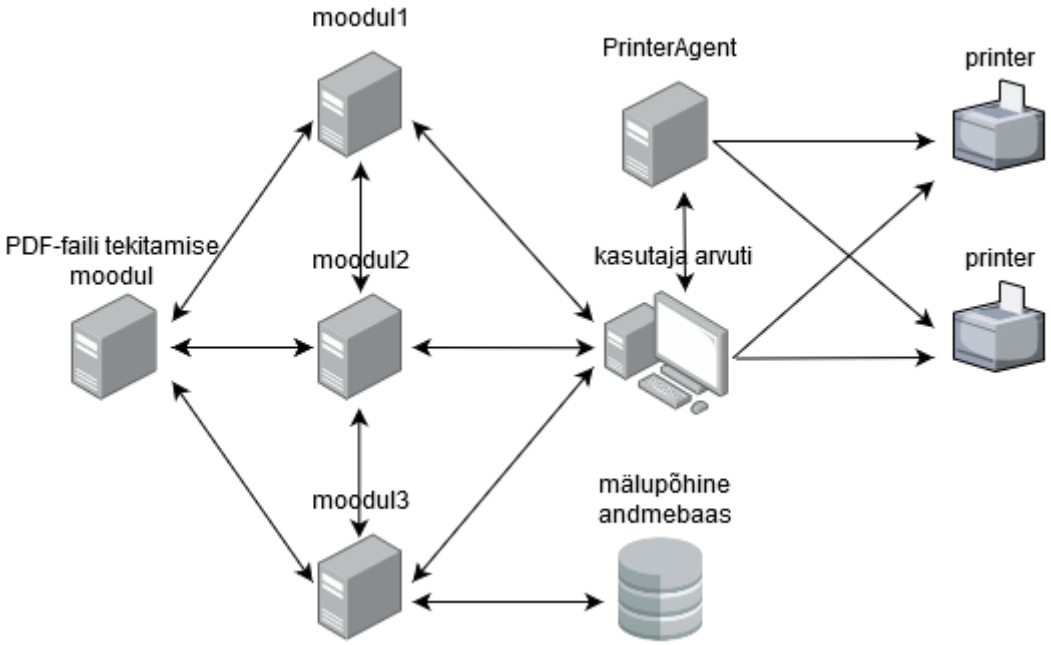
3.2 Puhverdamine

Voogedastus lahendab ära mälukasutusega seotud probleemid, kuid tekitab juurde uued probleemid. Kuna sama dokumenti on vaja mitu korda kasutada ning töödelda erinevatele kujudele, siis tuleb ka sama palju kordi teha päring PDF-faili tekitava mooduli vastu. See aga tähendab sama sisuga faili genereerimist mitu korda, mis lisab juurde ajakulu ning suurendab võrguliikluse koormust. Palju efektiivsem oleks samade sisendandmete põhjal genereerida ainult üks fail ning see talletada ajutiselt mällu või ketta peale. Sellist nähtust nimetataks puhverdamiseks. Oluline on siinjuures, et faile säilitatakse väljaspool põhimoodulit, et vähendada põhimooduli sessioonide mälukasutust.

Puhverdada on võimalik nii ketta peale kui ka mällu. Failide ketta peale puhverdamise plussiks on see, et selle jaoks pole eraldiseisvat tarkvara vaja. Miinuseks aga see, et ketta kiirus on aeglasem kui mälu kiirus ning puudub viis kuidas ajutisi faile automaatselt koristada. Mälus puhverdamiseks kasutatakse üldiselt mälupõhist andmebaasi. Osadel mälupõhistel andmebaasidel on võimalik määrata andmebaasi talletamise objekti eluiga. Kuna eHealth rakenduse puhul on tegemist ajutiste failidega, siis eluiga jääb sekunditesse või minutitesse, mille järel pole faili enam vaja ning võib mälust kustutada.

3.3 Lahendus

Lahenduseks on kasutada failide saatmiseks ja töötlemiseks voogedastamist ning failide puhverdamiseks mälupõhist andmebaasi. Voogedastamise ja puhverdamise kombineerimisel vabanetakse *PdfPrintWidget*-i suurte failide isendimuutujatest, jäädes hea jõudluse juurde.



Joonis 5. Uus PDF-faili voog arhitektuur.

Joonisel 5 on peatükis 2.4 välja toodud PDF-faili voo arhitektuur, millele on juurde lisatud mälupõhine andmebaas. Mälupõhiseks andmebaasiks valiti Redis populaarsuse ja kiiruse tõttu [11]. Mälupõhise andmebaasiga suhtluseks valiti Java teegiks Redisson, kuna selle abil on lihtne kasutada puhverdamist, voogedastamist ning objekti eluea määramist.

Java rakenduses tuleb luua ühendus rakenduse ja mälupõhise andmebaasi vahel. Redise andmebaasi ühenduse tekitamiseks kasutati Redisson-i teegi Redisson.create meetodit.

Ühenduse tekitamisel tuleb määrata vähemalt ühe Redis-e serveri IP-aadress. Juurde lisati kompressimine, mille tõttu kasutab andmebaas vähem mälu. Redisson-iga on võimalik kasutada kaht erinevat kompressimisalgoritmi: LZ4 ja Snappy. Testides kompressiti sama 5,3 MB fail. LZ4 kompressimisalgoritm tekitas 2,2 MB suuruse faili ning Snappy kompressimisalgoritm tekitas 2,6 MB suuruse faili. Sellest tulenevalt otsustati kasutada LZ4 algoritmi, kuna see tekitas väiksema faili. Juhul kui tekib viga, näiteks serveri IP aadress on vigane, siis genereeritakse logisse hoiatus:

```
@Configuration
public class RedisConfig {
  @Value("${redis.addresses}")
  private String addresses;

  @Bean
  RedissonClient redisson() {
    try {
      Config config = new Config();
      config.useClusterServers()
            .addNodeAddress(addresses.split(","));
      config.setCodec(new LZ4Codec());
      return Redisson.create(config);
    } catch (Exception e) {
      LOG.warn(e.getMessage());
      return null;
    }
  }
}
```

Järgnevalt luuakse Java *interface*, mis defineerib vajalikud operatsioonid mälupõhise andmebaasiga suhtluseks. Selleks luuakse kaks meetodit. Meetod put lisab mälupõhisesse andmebaasi PDF-faili ning tagastab unikaalse võtme, millega saab lisatud faili andmebaasist alla laadida. Meetod get tagastab võtme põhjal PDF-faili:

```
public interface RedisCacheService {
  InputStream get(String key);
  String put(InputStream value);
}
```

Faili võtmiseks andmebaasist teostatakse RedisCacheService *interface*-i get meetod. Meetodis kasutatakse konfiguratsiooni klassi poolt tekitatud RedissonClient objekti, mille kaudu leitakse võtmele vastav RBinaryStream tüüpi objekt. Antud objekti kaudu voogedastatakse vastav PDF-fail:

```
@Override
public InputStream get(String key) {
  RBinaryStream stream = redisson.getBinaryStream(key);
```

```
    return stream.getInputStream();
  }
```

Faili lisamiseks andmebaasi teostatakse put meetod. Meetodis tekitatakse unikaalne võti. Võtme kaudu luuakse RBinaryStream tüüpi objekt, millesse on võimalik talletada igat objekti, mille suurus on väiksem kui 512 MB. Vastavasse RBinaryStream objekti voogedastatakse put meetodi argumendina antud PDF-fail. Pärast objekti mälupõhisesse andmebaasi sisestamist, määratakse objektile aegumise aeg, mille möödudes fail kustutatakse. Oluline on määrata objekti aegumine pärast objekti lisamist andmebaasi, muidu aegumise määramise operatsioon ebaõnnestub:

```
@Override
public String put(InputStream value) {
  String key = createUniqueId();
  RBinaryStream stream = redisson.getBinaryStream(key);
  OutputStream outputStream = stream.getOutputStream();
  try {
    IOUtils.copy(value, outputStream);
    stream.expire(TIME_TO_LIVE, TimeUnit.MINUTES);
  } catch (IOException e) {
    throw new RedisCacheException(e);
  } finally {
    IOUtils.closeQuietly(value);
  }
  return key;
}
```

4. Realisatsiooni tulemused

Antud töö eesmärgiks on optimeerida PDF-failidega seotud töövoo mälukasutust. Sellest tulenevalt on tulemuste analüüsimisel põhirõhk mälukasutusel. Samas on oluline, et kasutaja vaatepunktist töötab uus lahendus täpselt samamoodi nagu vana lahendus. Käesolevas peatükis kontrollitakse, kas PDF-faili tekitamise voog töötab täpselt sama moodi nagu vanas lahenduses ning kontrollitakse, kas teises peatükis väljatoodud probleemid on lahendatud.

4.1 Tarkvara kvaliteet

Põhiline muudatus on mälupõhise andmebaasi kasutusele võtmine. Sellest tulenevalt tuleb kontrollida, mis juhtub olukorras, kui PDF-fail aegub ja kasutaja üritab seda faili pärida. Lisaks tuleb kontrollida olukorda, kus Redis-e rakenduse mälu saab täis.

Mälupõhises andmebaasis talletamisel määratakse objekti aegumise aeg. Kuna dokumendi ülevaatamine ja printimine võtab mõned sekundid või mõni minut, siis mõistlikuks aegumise ajaks oleks määrata umbes 10 minutit. See on piisav aeg PDF-faili ülevaatamiseks ja printimiseks, millele on lisatud mõistlik aeg puhvrit. Samas võib kasutaja dokumendi printerisse saata ka 10 minuti möödudes. Sellisel juhul on PDF-fail aegunud ja mälupõhisest andmebaasist eemaldatud. Printimine ebaõnnestub ning kasutajale kuvatakse viga, mille järel peab kasutaja eelneva kuva peale minema ning siis uuesti printimiskuva avama. Selle protsessi käigus tekitatakse uus fail ning kasutajal on võimalik see printida.

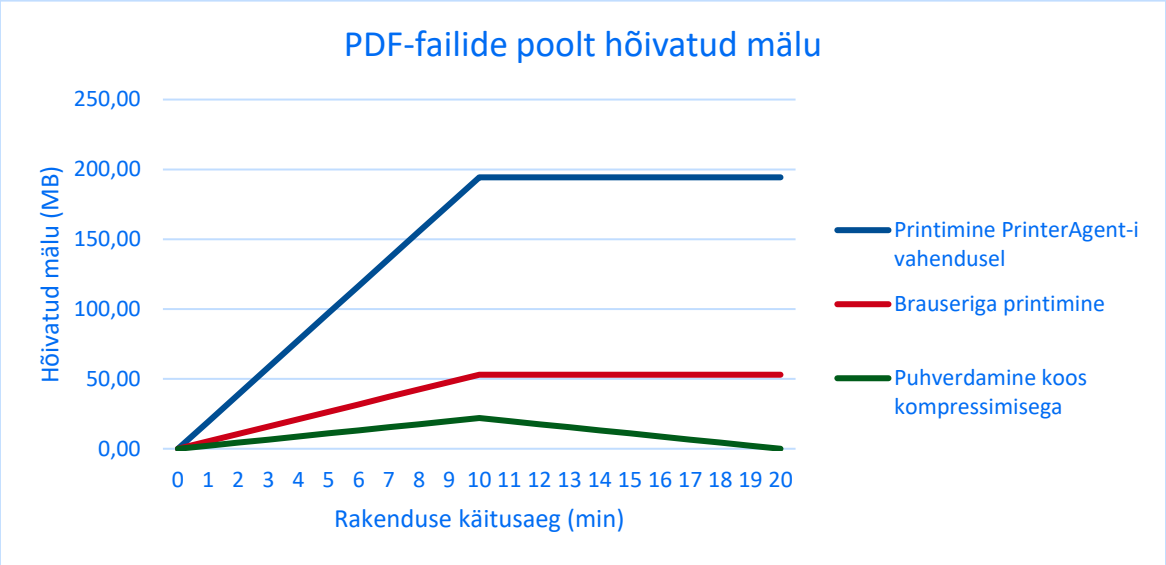
Kuna Redis rakenduse ressurssid on piiratud, siis tekib küsimus, mis juhtub olukorras, kui Redis-e poolt hõivatud mälu saab täis. Sellises olukorras muutub rakendus aeglasemaks, mälu saalitakse ketta peale, rakendus jookseb kokku või peatatakse operatsioonisüsteemi poolt. Positiivseks punktiks on see, et ülejäänud eHealth rakendus jääb püsima. Siiski on Redis rakendusele võimalik määrata reeglistik, mis juhtub olukorras, kui mälu limiit saab täis. Selleks tuleb määrata *maxmemory-policy*-le soovitud käitumisviis. Kuna antud lahenduse juures on igale objektile määratud aegumise aeg, siis kõige efektiivsem oleks kasutada *volatile-ttl* reeglit, mis kustutab kõige esimesena aeguva faili andmebaasist [12], [13].

4.2 Mälukasutus

Rakendus eHealth on suur ja keeruline. Täpse mälukasutuse mõõtmine rakenduses on keeruline, kuna pidevalt luuakse uusi objekte ning käidatakse prügikoristust. Sellest tulenevalt ei ole korrektne rakenduse protsessi mälumõõtmine. Lihtsuse mõttes mõõdeti

ainult PDF-failide peale kuluvat mälukasutust, kuna see moodustas suurema osa huvipakkuvast mälukasutusest.

Peatükkis 2.4 toodi välja, et rakenduses on kaks erinevat viisi printimiseks. Brauseri vahendusel printimine tekitab ajutise koopia kuid ei talleta seda isendimuutujana. PrinterAgent-iga printimisel tekitatakse PDF-failist Base64 sõne ja talletatakse isendimuutujana. See tähendab, et mälukasutuse vaatepunktist on PrintAgent ebaefektiivsem kui brauseri vahendusel printimine.



Joonis 6. PDF-faili printimise lahenduste mälukasutuse võrdlus.

Joonisel 6 on uue lahenduse ja eelnevate lahenduste võrdlus. Testi käigus tekitati iga minuti tagant üks 5,3 MB suurune PDF-fail, kokku 10 faili. Siis oodati veel 10 minutit, et kontrollida, kas tekitatud failid kustutatakse mälust. Jooniselt on näha, et kõige vähem PDF-failide poolt hõivatud mälu kasutab puhverdamine koos kompressimisega lahendus. Iga fail hõivas umbes 2,3 MB mälu. Brauseriga printimise puhul kasutas iga fail umbes 5,3 MB mälu. PrinterAgent lahendus oli kõige ebaefektiivsem. Ühe 5,3 MB suuruse PDF-faili tekitamisel hõivati umbes 19,4 MB mälu.

Pärast kümnendat minutit uusi faile ei tekitatud. PrinterAgent-i ja brauseriga printimise lahenduse mälukasutus jäi konstantseks mälulekke tõttu. Puhverdamine koos kompressimise lahenduse puhul määrati faili elueaks 10 minutit. Joonisel 6 on näha, et pärast kümnendat minutit, iga minuti tagant väheneb mälukasutus. 20 minuti möödudes on kõik failid mälust kustutatud.

Oluline punkt mälupõhise andmebaasi lahenduse puhul on see, et dokumente ei hoita enam rakenduse sessioonides, vaid eraldiseisvas Redis-e serveris. Jooniselt on puudu Redise mälupõhine andmebaas. Olenevalt sätetest ja serverite arvust võib see varieeruda. Ühe serveri jooksutamisel Docker-is võtab umbes 30 MB mälu. Kuna haiglates kasutatakse enamasti PrinterAgent-i lahendust, siis uus lahendus koos eraldiseisva serveriga kasutaksid kokkuvõttes vähem mälu.

5. Kokkuvõte

Käesoleva töö käigus analüüsiti Nortali loodud eHealth rakenduse PDF-failide vooga seotud mälukasutuse probleeme. Uurimise käigus selgus, et rakenduses esineb mäluleke ning suured failid hõivavad liiga suurt osa rakenduse sessioonidest. Sellest tulenevalt püstitati töö eesmärgiks PDF-failide vooga seotud mälukasutuse optimeerimine.

Probleemi põhjuseks oli suurte failide ebaefektiivne talletamine isendimuutujates, mis oli tingitud Aranea raamistiku eripärast. Töö käigus vabaneti probleemsetest isendimuutujatest, kasutades failide liigutamiseks ja töötlemiseks voogedastamist. Kuna üht faili kasutati mitu korda ning voogedastamise kaudu saab ainult üks kord töödelda faili, siis tekkis probleem, et üht faili pidi genereerima uuesti. See probleem lahendati failide puhverdamisega mälupõhises andmebaasis.

Lahenduses kasutati mälupõhist andmebaasi Redis. Java rakenduse ja Redis-e vahelise suhtluse lihtsustamiseks kasutati Java teeki Redisson, mille abil oli lihtne kasutada puhverdamist, voogedastamist, kompressimist ning objekti eluea määramist. Lahenduse kvaliteeti testides kontrolliti, et dokumendi aegumise tagajärjel ja mälu täissaamisel töötaks rakendus edasi. Mälukasutuse testimisel selgus, et voogedastamise ja puhverdamise lahendus on efektiivsem kui eelnev lahendus.

6. Viidatud kirjandus

[1] Tartu Ülikooli Kliinikum saab elektroonilise haigusloo. https://www.postimees.ee/2047027/tartu-ulikooli-kliinikum-saab-elektroonilise-haigusloo (14.01.2019)

[2] Tartu Ülikooli Kliinikumi ja Põlva Haigla koostöös algab uus ajastu. http://www.polvahgl.ee/?p=1979 (25.04.2019)

[3] Aranea framework. http://nortal.github.io/araneaframework/ (14.01.2019)

[4] A Web Development and Integration Framework. https://dspace.ut.ee/bitstream/handle/10062/2544/kabanov_jevgeni.pdf (05.05.2019)

[5] Aranea Web Framework 1.0 Final Release. https://www.theserverside.com/discussions/thread/42637.html (14.01.2019)

[6] Aranea—Web Framework Construction and Integration Kit. https://www.ioc.ee/~tarmo/tday-voore/myrk-kabanov-pppj06.pdf (14.01.2019)

[7] Graafilise kasutajaliidese komponentide voog. http://nortal.github.io/araneaframework/doc/1.1/reference/html_single/index.html (14.01.2019)

[8] PrinterAgent repositoorium. https://github.com/nortal/printeragent (28.04.2019)

[9] OWASP Memory Leak. https://www.owasp.org/index.php/Memory_leak (28.04.2019)

[10] Base64 length calculation. https://stackoverflow.com/questions/13378815/base64-length-calculation (09.05.2019)

[11] Redis System Properties. https://db-engines.com/en/system/Redis (09.05.2019)

[12] What happens if Redis runs out of memory? https://redis.io/topics/faq (09.05.2019)

[13] Using Redis as an LRU cache. https://redis.io/topics/lru-cache (09.05.2019)

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Sander Pärn,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Suurte failide puhverdamine Nortali eHealth projekti näitel,

   mille juhendaja on Artjom Lind,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Sander Pärn

10.05.2019