Tartu Ülikool

Loodus- ja täppisteaduste valdkond

Tehnoloogiainstituut

Kristofer Kurvits

Andmete reaalajas kogumise võrdlemine kasutades Apache NiFit ja Pythonit

Bakalaureusetöö (12 EAP)

Arvutitehnika eriala

Juhendaja:

Pelle Jakovits, PhD

Tartu 2019

Resümee

Viimastel aastatel populaarsust kogunud *DevOps*i kultuur on jõudnud andmeteaduse valdkonda, mida kutsutakse *DataOps*iks. Selle tõttu on hakatud ehitama andmetorusi, et kontrollida andmete kogu elutsükklit. Kui andmete maht on kasvanud väga suureks, siis *DataOps*i eesmärk on parandada suhtlust, koostööd, automatiseerimist ja integratsiooni erinevate tiimide vahel, näiteks andmeteadlaste ja andmeanalüütikute. Selle tõttu on hakatud ehitama andmetorusi, et kontrollida andmete kogu elutsükklit. Käesoleva bakalaureusetöö eesmärk on ehitada andmetoru kasutades tarkvara Apache NiFi ning võrrelda seda tavapärase skriptilise lähenemisega kasutades programmeerimiskeelt Python. Andmetoru on loodud temperatuuri mõõtvate seadmete, mis töötavad Raspberry Pi 3 arvutitel ning Tartu Ülikooli pilves olevate masinate vahele.

CERCS: P170

Võtmesõnad: Automatiseerimine, andmed, *DevOps*, *DataOps*, andmetoru

Abstract

In the last years DevOps culture has gained popularity and has applied on the field of data science, which is called DataOps. It is because of that the creation of data pipelines has begun to have control over data lifecycle. When the volume of data has become huge, DataOps aims to improve communication, cooperation, automation and integration between different teams for example data scientists and data analysts. The purpose of the thesis is to build a data pipeline with a software intended for that purpose, Apache NiFi and to compare it to scripting approach using programming language Python. The data pipeline is created between temperature measuring devices which are working on Raspberry Pi 3 computers and University of Tartu's cloud environment instances.

CERCS: P170

Keywords: Automation, data, DevOps, DataOps, data pipeline

Mõisted

DevOps (*Development and Operations***)** - Arendus ja haldus

DataOps (*Data and Operations*) - Andmed ja haldus

TÜ - Tartu Ülikool

Versioonimine (*Versioning*) - Toote või teenuse versioonide loomine ja haldus

Andmetoru (*Data pipeline*) - Eri allikatest pärit andmete struktureerimise, töötluse ja teisendamise protsessistik

Regulaaravaldis (*Regular expression*) - Määratletud tähistuse ja ehitusega avaldis otsitava teksti malli ja käsitlusviisi spetsifitseerimiseks

Prognooshooldus (*Predictive maintenace*) - Hooldusvorm, mis põhineb objekti tegelikul seisundil

VPN (*virtual private network*) - Virtuaalne privaatvõrk

vCPU - Virtuaalprotsessor

XML (*Extensible Markup Language)* - Platvormist sõltumatu märgistuskeel

YAML (*YAML Ain't Markup Language*) - Inimloetav andmete jadastuse keel

Saalimine (*Swapping*) - Mingi põhimäluala sisu vahetus mingi välismäluala sisuga

SQL (structured query language) - Struktureeritud päringute keel

JVM (*Java virtual machine*) - Java virtuaalmasin

MQTT (*Message Queuing Telemetry Transport*) - Sõnumijärjekorraga telemeetriatransport

NTP (*Network Time Protocol***) -** Võrguaja protokoll

SSH (*Secure Shell*) - Võimaldab turvalist krüpteeritud kaugpöördust

1. Sissejuhatus

Viimastel aastatel on üha rohkem populaarsust kogunud tarkvaraarenduse metoodika DevOps (*Development and Operations*) [1]. Siim Vene [2] defineeris enda magistritöös DevOpsi tähendust järgmiselt: DevOps on kultuur või praktika, mis käsitleb eneses nii arenduse kui halduse kompetentside koondamist omavahel kas läbi parema kommunikatsiooni või tihedalt seotud meeskondade. Protsesside automatiseerimise abil võimaldab tarkvara arendamist, testimist ning juurutamist teha kiiremini ja sagedamalt. Toetab teenuste arendamist vastavalt kiiresti muutuvatele turunõuetele, tehes seda odavamalt ning vähemate katkestustega.

Andmed on tihtipeale võtmetähtsusega tänapäeva ärimaailmas, kuna need aitavad teha paremaid otsuseid. Tehes otsuseid kogutud andmete põhjal, on võimalik suurendada ettevõtte kasumit [3]. Tänapäeval mõne ettevõtte ärimudel seisnebki andmete analüüsis, näiteks Eesti ettevõtte MindTitani.

Üha enam populaarsust koguvas andmeteaduse valdkonnas on hakatud kasutama *DevOps*i põhimõtteid. Sellest on ka välja kujunenud uus termin *DataOps*, mis kujutab endast keskpunkti andmetehnika, -integratsiooni, -turvalisuse ja -kvaliteedi vahel [4]. Kui andmete maht on kasvanud väga suureks, siis *DataOps*i eesmärk on parandada suhtlust, koostööd, automatiseerimist ja integratsiooni erinevate tiimide vahel, näiteks andmeteadlaste ja andmeanalüütikute. Selle tõttu on hakatud ehitama andmetorusi, et kontrollida andmete kogu elutsükklit. Andmetoru kujutab endast süsteemi liigutamaks andmeid lähtekohast sihtkohta. Apache NiFi on tarkvara, mis lahendab need probleemid ning mis on loodud *DevOps* andmetoru ehitamiseks. Apache NiFi tarkvara sai valitud selle tõttu, et see on avatud lähtekoodiga, sellega on võimalik andmeid töödelda reaalajas ning on tõrkeid taluv.

*DataOps*i tavade kasulikkus tuleb välja näiteks prognooshoolduse valdkonnas (*predictive maintenance*). Ajakirjas Reliableplant [5] kirjeldatakse prognooshooldust kui hoolduse liiki, kus jälgitakse töötava seadme jõudlust ja seisukorda, et saada seadme riknemisele jälile enne, kui see tegelikult juhtub.

Järgmistes alampeatükkides tutvustab autor probleemi, töö eesmärki ning hindamiskriteeriumeid. Järgmises peatükis räägib autor *DevOps*i olemusest ja kus seda

rakendada. Kolmandas peatükis tutvustab autor kasutatud tarkvara, neljandas räägib kasutusloost ning ehitatud andmetorust. Viiendas analüüsib tulemusi lähtudes hindamiskriteeriumitest ja kasutusloost ning viimases peatükis võtab tehtud töö kokku.

1.1 Probleemi tutvustus

Andmetorude loomisel on kõige olulisem andmete terviklikkus. Kui mõne ettevõtte ärimudel koosneb suuresti protsessidest andmetega, on andmete korrektne olemasolu ettevõtte edukuse jaoks tähtis. Andmete kogumine ja edasi suunamine on võimalik ellu viia üsna kiiresti lähenedes skriptiliselt, tihtipeale aga selline lähenemine ei ole jätkusuutlik. Andmetorude ehitamisel on tihtipeale suureks katsumuseks näiteks internetiühenduse katkemine ja andmete kogumise prioriteetide muutumine. Samuti on oluline, et andmetoru oleks võimalik kiiresti üles seada ning muutused kiiresti ellu viia. Neid probleeme adresseerides on hakatud looma erinevaid programme, et andmetorude loomine ja haldamine oleks võimalikult murevaba ja efektiivne. Konkreetses lõputöös võetakse kasutusele Apache NiFi ja võrreldakse seda tavapärase skriptlise lähenemisega.

1.2 Töö eesmärk

Apache NiFi on tarkvara, mis on loodud suurte koguste andmete liigutamiseks. NiFi eesmärk on automatiseerida seda protsessi lähteseadmetelt kuni soovitud seadmeteni. Käesoleva bakalaureusetöö eesmärk on ehitada andmetoru kasutades tarkvara Apache NiFi ning võrrelda seda tavapärase skriptilise lähenemisega. Andmetoru on loodud temperatuuri mõõtvate seadmete, mis töötavad Raspberry Pi 3 arvutitel ning Tartu Ülikooli pilves olevate masinate vahele. Tartu Ülikooli pilv on majutatud OpenStack keskkonnas, millele pääseb ligi läbi TÜ sisevõrgu. Andmetoru loomine põhineb kasutuslool, millest täpsemalt räägitakse peatükis 4.

Eelnevalt kirjeldatud tarkvara efektiivsust võrdlen järgmises alampeatükis kirjeldatud hindamiskriteeriumite ning kasutusloo järgi.

1.3 Hindamiskriteeriumid

Hindamiskriteeriumid on koostatud autori ja juhendaja poolt. Autor hindab kasutatavat tarkvara järgmiste hindamiskriteeriumite alusel.

Andmetoru ülesseadmise keerukus - autori hinnang kui keeruline on andmetoru üles seada konkreetse lähenemisega.

Dokumentatsiooni põhjalikkus - autori hinnang dokumentatsiooni põhjalikkusele ja internetist leiduvate materjalide rohkusele.

Saatmiskiirus - andmete kogumishetkest kulunud aeg lähteseadmelt (Raspberry Pi 3) läbi Tartu Ülikooli pilves asuva serveri kuni samas pilves, kuid erinevas serveris asuvasse andmebaasi. Lähteseadmelt keskserverisse saatmiseks kasutatakse 50MB üles- ja allalaadimiskiirusega internetiühendust.

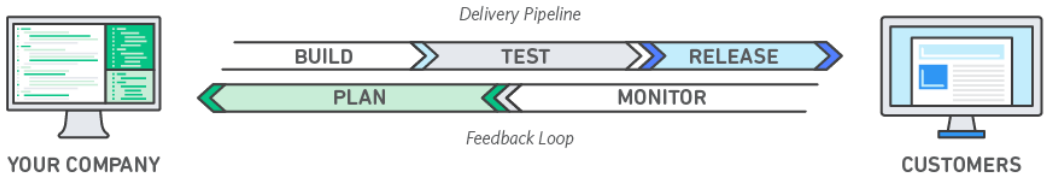
Tarkvara ressursinõudlus lähteseadmel - autor võrdleb lähteseadmel protsessori ja mälu hõivatust keskserveriga suhtlusel.

2. Kirjanduse ülevaade

2.1 *DevOps*

Siim Vene kirjeldab oma magistritöös "*Devops* juurutamine suurettevõttes" [2] *DevOps*i kui praktikat, kus halduse ja arenduse insenerid töötavad koos terve teenuse elutsükli jooksul, disaini faasist läbi arenduse kuni toodangukeskkonda paigaldamiseni. Lisaks käsitletakse *DevOps*i ka kui arendustehnikate kasutamist halduse inseneride poolt nagu versioonihalduse ning testimise printsiipide rakendamine.

Amazoni kodulehel [6] avaldatud artikklis räägitakse, et *DevOps*i praktika juures on oluline sage muudatuste elluviimine, kuid võimalikult väikeste, sest selline lähenemine vähendab riski uuenduste toodangukeskkonda paigaldamisel. Tagasiside aitab toodet parandada ja tarkvaraarendajatel kiiremini vigu üles leida.



Joonis 1. *DevOps*i voog [6]

Joonisel 1 on välja toodud *DevOps*i voog aitamaks eelnevat teksti illustreerida.

2.2 *DataOps*

S. Distefano ja teised kirjutasid oma uurimuses [7], et *DataOps* on uus lähenemine parandamaks andmete elutsükkli kvaliteeti. *DataOps*i eesmärk on parandada suhtlust, koostööd, automatiseerimist ja integratsiooni erinevate tiimide vahel, näiteks andmeteadlaste ja andmeanalüütikute. *DataOps*i tavad pärinevad *DevOps*i praktikast ning selle eesmärk on *DevOps*i kasulikkus andmeteaduse maailma tuua. Andy Palmeri sõnul [4] *DevOps*i loomus hõlmab vajadust hallata mitmeid andmeallikaid ja mitmeid andmetorusi ning valmisolekut muutusteks. Järgnevad lõigud tuginevad Pat Pattersoni artikklile [8].

Üheks *DataOps*i tiimi vastutuseks on andmetorude loomine. DevOpsi tavasi järgides, on andmetoru ulatus võimalikult väike. Andmetoru lähtekohas kogutakse andmeid, olgu selleks näiteks mõni seade koos sensoritega. Andmetoru käigus võidakse andmeid filtreerida või neid mingil määral manipuleerida, enne kui need lõpp punktis hoiustatakse, näiteks andmebaasis. Loodud vooge tuleks hoida topoloogiatena, mis võimaldab lihtsama vaevaga andmete teekonda jälgida. *DataOps* lähenemisel on rõhk pigem konfiguratsioonil kui koodil, suurendades taaskasutust ja seeläbi ka vähendades ajakulu. *DataOps* tööriistad võimaldavad andmetorusi luua ka inimestel, kes pole programmeerimisega varem kokku puutunud.

Kui loodud andmetoru jõuab tootmisfaasi, on oluline valida selleks sobiv keskkond. Kui ettevõte omab oma infrastruktuuri, suurendab see paindlikkust, kuid sel juhul läheb rohkem ressurssi infrastruktuuri haldamisele. Tänapäeval on andmete hoiustamiseks populaarsed erinevad pilveteenused. Mõndade pilvekeskkondade puhul võib aga tekkida tarnija lukustus (*vendor lock-in*) ehk tekib sõltuvus konkreetsest teenusepakkujast ja andmete teise kohta liigutamine on raskendatud.

Kui süsteem on kasutusele võetud, on oluline andmevooge jälgida. Automatiseeritud monitoorimine võimaldab *DataOps* tiimil muutustele kiirelt reageerida, samuti on võimalik suurenenud koormuse puhul süsteemi automaatselt skaleerida. Andmetorude loomisel tuleks alustada väikselt ning vältida kompleksust enne kui seda on vaja.

Eelnevalt kirjeldatu elluviimiseks on mitmeid viise, konkreetses lõputöös proovib autor seda teha tarkvaraga Apache NiFi.

2.3 Prognooshooldus

*DataOps*i tavade kasulikkus tuleb välja näiteks prognooshoolduse valdkonnas (*predictive maintenance*). Järgnevad lõigud tuginevad Bryan Christianseni artikklile [9].

Viimaste aastatega on prognooshoolduse turg tõusutrendis püsinud. Uurimuse kohaselt on tõus tingitud suurenenud tähelepanust tegevuskulude ja seadmete seisakute vähenemisele.

Prognooshooldus on hooldusvorm, kus proovitakse seadme riknemist ette näha ning hooldustööd vahetult enne seda tehtud saada. Seadme riknemise ettenägemine baseerub

seadmete seisukorral, mille kohta kogutakse infot erinevate sensoritega. Prognooshoolduse eesmärk on ootamatute riknemiste minimeerimine ja seadmete töökindluse suurendamine. Sinna alla kuulub ka tegevuskulude vähendamine, kuna hooldus viiakse läbi ainult siis kui see on vajalik, mille tõttu hoitakse rohkem tööjõudu ja aega kokku. B. Christiansen on oma teises artikklis [10] välja toonud, et prognooshooldus on kuluefektiivsem võrreldes kahe sarnase hooldusvormiga, reaktiivne hooldus (*reactive maintenance*) ja ennatlik hooldus (*preventive maintenance*). Reaktiivse hoolduse puhul sooritatakse hooldustöid alles siis, kui seade on juba riknenenud ja ennatliku hoolduse puhul lähtutakse seadme eeldatavast elueast.

Prognooshoolduse puhul jälgitakse seadmete seisukorda reaalajas. Võttes kasutusele ennustusvalemid, on võimalik suurendada tähelepanu probleemsematele piirkondadele.

Üks prognooshoolduse rakendamise asukohtadest on tehased. Tehastesse paigaldatakse hulk erinevaid sensoreid, et seadmete seisukorda jälgida. Samuti tasuks jälgida ka andmeid koguva seadme seisukorda. Tehase keskkonda arvestades, ei pruugi masina tervist väljaspoolt näha, oluline informatsioon võib tihtipeale peituda masina sees. Selleks, et see info kätte saada, on võimalik kasutada näiteks vibratsiooni, temperatuuri või müra sensoreid sõltuvalt masinast.

3. Kasutatavad tööriistad

3.1 Apache NiFi

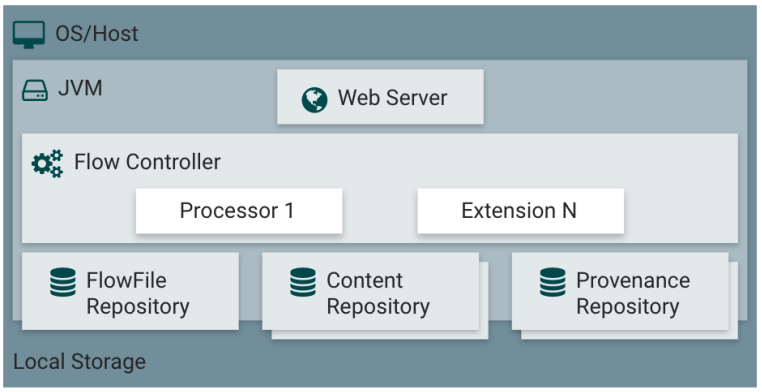
Käesolev alampeatükk tugineb Apache NiFi dokumentatsoonile [11]. NiFi on andmevoosüsteem, mis põhineb voopõhise programmeerimise kontseptsioonidel. Sellel on veebipõhine kasutajaliides, kus on võimalik vooge koostada, kontrollida ja monitoorida. Lisaks on see väga konfigureeritav. Andmevoogude koostamine töötab suunatud graafide põhimõttel, ehk kasutaja defineerib suuna kahe punkti vahel ning nende kahe punkti vahel saavad andmed ainult selles suunas liikuda. NiFi pakub võimalust jälgida andmete päritolu ja nendega tehtud toiminguid.

NiFi voo tähtsaimad komponendid on FlowFile, FlowFile Processor, Connection ja Flow Controller.

- FlowFile - Kujutab endast objekti, mis liigub läbi süsteemi. NiFi hoiab võti-väärtus paari selle objekti atribuutide ja sisu kohta, attribuut on võrreldav tavalise muutujaga. Objektiks on protsessi poolt tagastatud väljund, näiteks faili lugemise korral faili sisu ning attribuutideks näiteks faili nimi ja millal fail loodi.
- FlowFile Processor - Nendel on ligipääs FlowFile atribuutidele ja sisule ning need teevad tegelikku tööd, kokku on neid üle 200 erineva tüübi. FlowFile Processorid tegelevad näiteks faili lugemisega, programmide käivitamisega ja tekstitöötlusega (vt. Joonis 9). Võimalus on ka koguda infot välistest allikatest, näiteks mõnelt veebilehelt või andmebaasist.
- Connection - Ühendus FlowFile protsessorite vahel. Toimib kui järjekord, mis lubab andmeid suunata erinevate protsessorite vahel. Kasutajal on võimalik kontrollida, kui suure koguse andmeid ühendus hoiab. Kui järgnev protsessor ei suuda piisavalt kiiresti või üldse andmeid vastu võtta, kogutakse andmed järjekorda. Ühendus koosneb ühest või mitmest suhtest, näiteks ühest Processorist teise jõuavad andmed ainult siis, kui teatud tingimus on täitunud. Kõige levinumad tingimused on “success” ja “failure”, ehk andmed suunatakse edasi, kui protsess sooritab oma töö edukalt või siis mitte (vt. Joonis 4).

- Flow Controller - Töötab vahendajana, mis kontrollib kõiki lõimesid, mida protsessorid kasutavad. Hoiab infot selle kohta, kuidas protsessid omavahel ühendatud on ja haldab lõimesi, mida protsessid kasutavad. Töötab vahendajana hõlbustades FlowFile'de vahetust Processorite vahel.

NiFit on võimalik jooksutada võimsates serverites klastris kui ka üksikus sülearvutis. Vajalik riistvara sõltub defineeritud andmevoost, NiFi ise võtab ruumi 1,2GB jagu. Kui NiFi andmeid töötleb, hoitakse koopiat andmetest ka andmekandjal. NiFi töötab Windowsi ja unixi masinatel ning nõuab kas Java 8 või 11 olemasolu. Kuna NiFi kasutab veebipõhist kasutajaliidest, siis brauseritest toetavad selle edukat töötamist Microsoft Edge, Mozilla Firefox, Google Chrome ja Safari viimased ning nendest ühe võrra eelnevad versioonid.



Joonis 1. NiFi arhitektuur [11]

NiFi töötab Java virtuaalmasinas. NiFi arhitektuuri (vt. Joonis 1) peamised komponendid on:

- Veebiserver - Selle kaudu käib suhtlus välismaailmaga.
- Flow Controller - Töötab vahendajana, mis kontrollib kõiki lõimesid, mida protsessorid kasutavad. Hoiab infot selle kohta, kuidas protsessid omavahel ühendatud on ja haldab lõimesi, mida protsessid kasutavad. Töötab vahendajana hõlbustades FlowFile'de vahetust Processorite vahel.
- FlowFile Repository - Hoitakse infot aktiivsete FlowFile'de hetkeseisu kohta. Kõik muutused kirjutatakse automaatselt sellesse repositooriumisse enne muutuste elluviimist. See võimaldab süsteemil katkestuste korral jätkata kohast, kus töö pooleli jäi.

- Content Repository - Hoitakse aktiivse ja eelmiste FlowFile'de sisu.
- Provenance Repository - Hoitakse infot FlowFile'de ajaloo kohta. Iga kord, kui FlowFile luuakse või muudetakse, tehakse uus sissekanne sellesse repositooriumisse. Sissekandesse kopeeritakse FlowFile'i attribuudid, viide selle sisule ja suhe teiste FlowFile Processoritega. See võimaldab näiteks vaadata, millal andmed saadeti, kust saadeti ja mida saadeti ning vajadusel uuesti saata, kui järgmine lüli pole andmeid kätte saanud.

NiFi puhul ei ole unustatud ka turvalisust. Erinevate masinate vahel saadetud andmed on võimalik krüpteerida, et volitamata isikud andmetele ligi ei pääseks. Sama kehtib ka kasutaja ja NiFi vahelise suhtluse kohta, kus näiteks kõik sisestatud paroolid on samuti krüpteeritud. NiFi seadmed suhtlevad omavahel läbi NiFi Site-to-Site (S2S) protokolli, mis lubab luua kas sokkliühenduse või suhelda üle HTTP(S) protokolli.

3.2 Apache MiNiFi

MiNiFi on NiFi alamprojekt. MiNiFi töötab samal põhimõttel nagu NiFi ainult, et on mõeldud nõrgema jõudlusega seadmetele, keskendudes andmete kogumisele selle lähtekohas. MiNiFi üks eelistest on see, et MiNiFi koos NiFiga võimaldab kogu süsteemil skaleeruda. Omades kümneid või sadu lähteseadmeid on ajakulukas kõikide seadmete koodi kallale minna, eriti kui seadmed asuvad geograafiliselt erinevates asukohtades. MiNiFi ja NiFi kombinatsioon võimaldab lihtsa vaevaga andmete kogumises muutusi teha. Selleks on loodud programm MiNiFi c2, mis hoiab koostatud voogude konfiguratsioonifaile ning suhtleb lähteseadmega, kus asub MiNiFi. Kuna MiNiFil puudub kasutajaliides, siis on võimalik voog defineerida NiFi kasutajaliidesel, teine variant on kirjutada konfiguratsioonifail manuaalselt. Kui voog on defineeritud, tuleb sellest luua mall. NiFis loodud mall on XML tüüpi, kuid MiNiFi aksepteerib YML tüüpi konfiguratsioonifaile. Vajaliku transformatsiooniga aitab MiNiFi toolkit. Tagastatud konfiguratsioonifail tuleb asetada MiNiFi c2 kausta, kust MiNiFi seade seda automaatselt küsib üle HTTP protokolli. Kuidas seda kõike üles seada, on kirjeldatud peatükis 4.1.1. Erinevalt NiFist on võimalik kasutada lisaks Java versioonile ka C++ versiooni. Java versioon toetab veidi üle 60 protsessori, C++ versioon toetab 33 protsessorit [9].

3.3 Python

Et võrrelda MiNiFi ressursinõudlikkust lähteseadmel luuakse samasugune andmetoru kasutades programmeerimiskeelt Python 3.7.3. Python on üks levinumaid programmerimiskeeli ning paljude avalike materjalide tõttu Pythonist pikemalt ei räägita.

4. Kasutuslugu

Andmetoru loomine on ülesehitatud kasutusloole, mis realiseeritakse Apache NiFi ja MiNiFi tarkvaraga ning võrreldakse seda skriptlise lähenemisega. Kasutusloos on lähtutud hoolduse vormist prognooshooldus (*predictive maintenance*). Selleks sai valitud lähteseadmeteks Raspberry Pi arvutid, mille külge on võimalik ühendada sensoreid ja samas võimaldab koguda ka andmeid seadme enda kohta.

Kasutuslooks on temperatuuri mõõtja, millega jälgida tehastes tööruumide temperatuuri. Lähteseadmed ja keskserver koos andmebaasiga asuvad geograafiliselt erinevates kohtades ning erinevates võrkudes. Et lähteseadmed saaksid ühenduse keskserveriga, mis asub TÜ sisevõrgus, on loodud VPN ühendus tarkvaraga OpenVPN. Kuna keskserver asub masinaga, millel töötab andmebaas, samas võrgus, pole nendevahelise suhtluse jaoks lisameetmeid vaja rakendada. Kasutusloos paika pandud süsteem peab olema tõrkeid taluv ja vajadusel skaleeritav. Samuti on oluline, et andmete kogumise viisis muudatusi tehes oleks neid võimalik kiiresti lähteseadmetel rakendada.

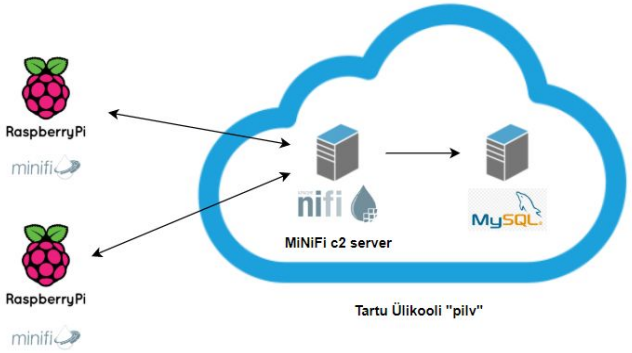
Temperatuuri mõõdetakse Dallas DS1820 sensoritega ning mõlemad on ühendatud erinevate Raspberry Pi 3 arvutitega. Tarkvarana on kasutatud NiFi 1.7.0, MiNiFi 0.5.0, MiNiFi toolkit 0.5.0, MiNiFi c2 0.6.0 ning MySQL 5.7.3. Samasugune andmetoru on ehitatud võrdluseks kasutades ainult Python programmeerimiskeelt. Pilves asuvate masinate operatsioonisüsteemiks on Ubuntu 18.04 ning lähteseadmetel Raspbian GNU/Linux 10 (buster).

4.1 Teostus

Kasutusloo realiseerimiseks kasutatakse kokku nelja seadet. Lähteseadmeteks, ehk andmete kogujateks on kaks Raspberry Pi 3 B+ arvutit, millel on nelja tuumaline 1.4Ghz protsessor, muutmälu 1GB ning andmekandjaks SD-kaart U1 klassist, mis mahutab kuni 32GB andmeid.

Lähteseadmed suhtlevad Tartu Ülikooli pilves asuva keskserveriga, millel on 8 vCPUd, 16GB muutmälu ja 20GB salvestusruumi. Keskserver jooksutab NiFi ja MiNiFi c2 tarkvara.

Keskserver omakorda suhtleb samas pilves teise virtuaalmasinaga, millel on 2 vCPUd, 4GB muutmälu ja 20GB salvestusruumi. Sellel masinal asub MySQL andmebaas (vt Joonis 2).



Joonis 2. Seadmete topoloogia

Loodud andmetoru töötab järgmiselt: lähteseadmed kasutavad keskserveriga suhtluseks Apache MiNiFi tarkvara, keskserver võtab vastu ja suunab andmeid edasi läbi Apache NiFi ning samuti hoiab töös MiNiFi c2 serverit, mis varustab lähteseadmeid konfiguratsioonifailiga. Andmetoru lõpp punktiks on eraldiseisev MySQL andmebaas.

4.1.1 MiNiFi voog

Esmalt on vaja defineerida MiNiFi voog, kuidas andmeid koguda. Kuna MiNiFil puudub graafiline liides, siis on võimalik seda teha NiFi kasutajaliidesel. MiNiFi toetab 63 andmetöötlusprotsessorit, NiFi üle 150, seega oluline on enne MiNiFi voo defineerimist uurida olemasolevate protsessorite kohta [12][13]. NiFis loodud MiNiFi voost on vaja luua mall. Koostatud mall on XML tüüpi, kuid MiNiFi aktsepteerib YAML tüüpi konfiguratsioonifaile. Selleks on loodud MiNiFi toolkit, mis aitab vajaliku transformatsiooniga. MiNiFi toolkiti jooksutades on vaja kaasa anda 3 argumenti, milleks on märksõna “transform”, XML fail ja kasutaja defineeritud nimega YML fail. Joonisel 3 on näidatud eelnevalt kirjeldatud protsess.

```
ubuntu@server-kurvits:~/minifi-toolkit-0.5.0/bin$ ./config.sh transform minifi.xml config.yml

Java home: /usr/lib/jvm/java-1.8.0-openjdk-amd64
MiNiFi Toolkit home: /home/ubuntu/minifi-toolkit-0.5.0


No validation errors found in converted configuration.
ubuntu@server-kurvits:~/minifi-toolkit-0.5.0/bin$
```

Joonis 3. NiFi malli muutmine MiNiFi konfiguratsioonifailiks

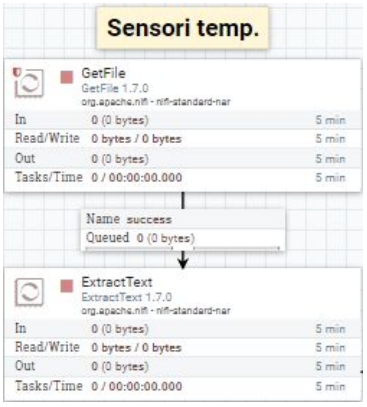
Raspberry Pi ja temperatuuri sensoriga temperatuuri mõõtmiseks on vaja lubada One-Wire liides Raspberry Pi peal. Selleks on vaja esmalt */boot* kaustas olevasse Config.txt faili lisada rida "dtoverlay=w1–gpio" ning teha Raspberry Pile taaskäivitus. Peale seda on vaja lisada gpio ja therm moodulid linuxi kernelisse käskudega "sudo modprobe w1–gpio" ja "sudo modprobe w1-therm". Temperatuur kirjutati autori puhul kaustas */sys/bus/w1/devices/10-0008031ad55b* olevasse W1_slave faili (vt. Joonis 4). Failitee viimane kaust on igal kasutajal erinev. [14]

```
pi@client1:~ $ cat /sys/bus/w1/devices/10-0008031ad55b/w1_slave
2e 00 4b 46 ff ff 0c 10 00 : crc=00 YES
2e 00 4b 46 ff ff 0c 10 00 t=23000
```

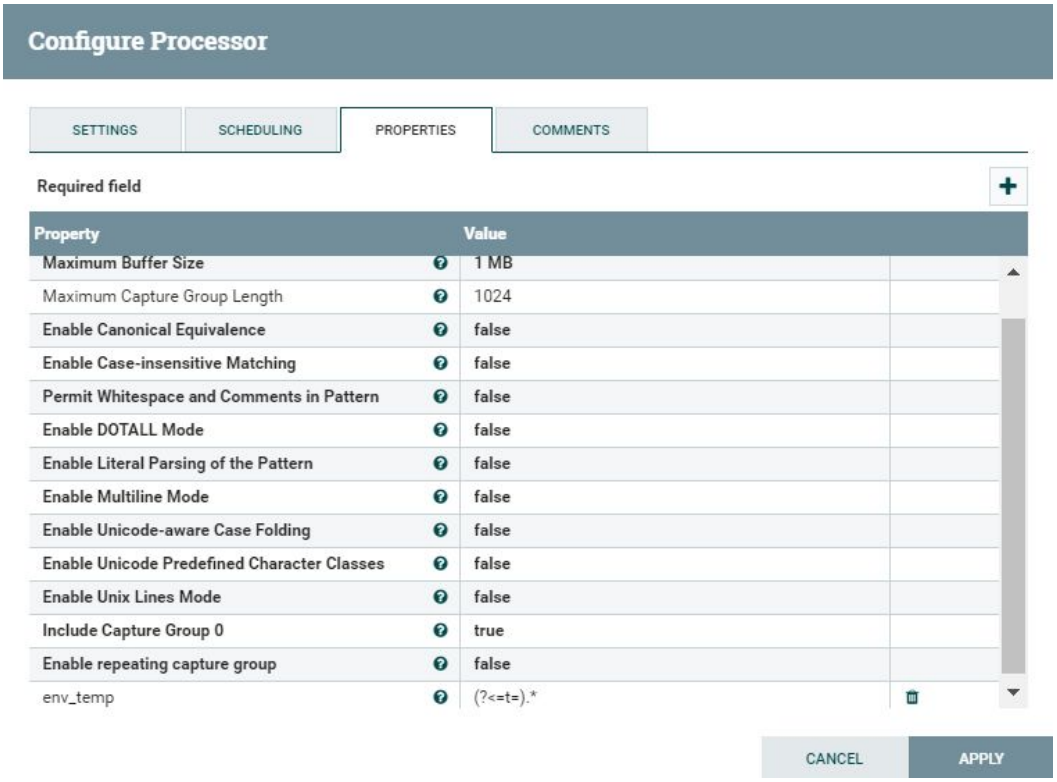
Joonis 4. W1_slave faili sisu. Temperatuur on pärast t= sümbolit, hetkel 23°C

Lähteseadmelt mõõdetud keskkonna temperatuuri kättesaamiseks on võimalik kasutada GetFile protsessorit, mis loeb kasutaja poolt määratud faili sisu. GetFile protsessor on ühendatud ExtractText protsessoriga (vt. Joonis 5), mis võimaldab regulaaravaldisega vajaliku info kätte saada. Joonisel 6, viimasel real, on välja toodud regulaaravaldis temperatuuri eraldamiseks teistest andmetest, eraldatud andmed muudetakse attribuutideks. Paremal tulbas on regulaaravaldis ning vasakus tulbas attribuut, millesse eraldatud väärtus salvestatakse. Ülejäänud parameetrid on autor jätnud vaikeväärtusteks.

Atribuut on võrreldav tavalise muutujaga ja need omistatakse FlowFile'le. Atribuudid võimaldavad näiteks marsruutida andmeid RouteOnAttribute protsessoriga, lisaks saavad mõned protsessorid, näiteks PutFile, võtta vastu otsuseid olenevalt attribuudi väärtusest. PutFile puhul võidakse FlowFile atribuudi väärtusest sõltuvalt faile erinevatesse kohtadesse salvestada. [8]



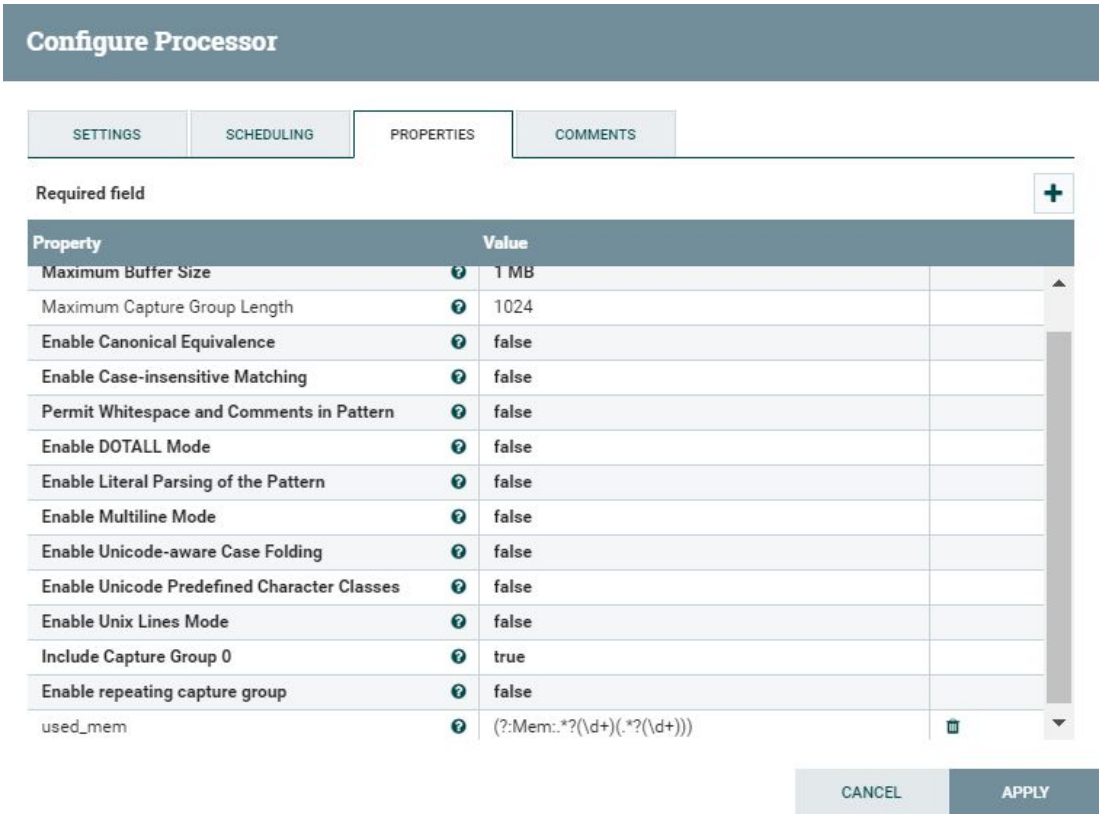
Joonis 5. Sensori andmete kogumise voog



Joonis 6. Regulaaravaldis temperatuuri eraldamiseks

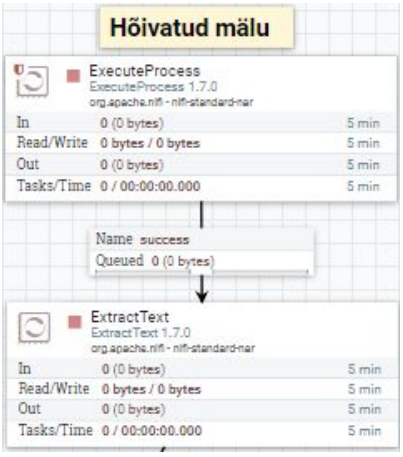
Samasugust loogikat on järgitud ka Raspberry Pi protsessori temperatuuri kogumiseks ja seadme nime salvestamiseks. Lisaks keskkonna ja protsessori temperatuurile kogutakse veel ka andmeid lähteseadme mälu kohta, milleks on hõivatud mälu ning vaba saalimisruum. Samuti kogutakse aja hetkeväärtust, et mõõta, kui kiiresti andmed lähteseadmelt keskserverisse jõuavad. Infot mälu ja aja kohta küsitakse läbi unixi programmide Free

argumendiga -m, et tagastada väärtused megabaitides ja Date argumendiga +"%Y-%m-%d %T.%3N", et tagastada kuupäev ning kellaaeg millisekundi täpsusega. Seda võimaldab teha ExecuteProcess protsessor (vt. Joonis 8). Soovitud andmed eraldatakse jällegi ExtractText protsessoriga.



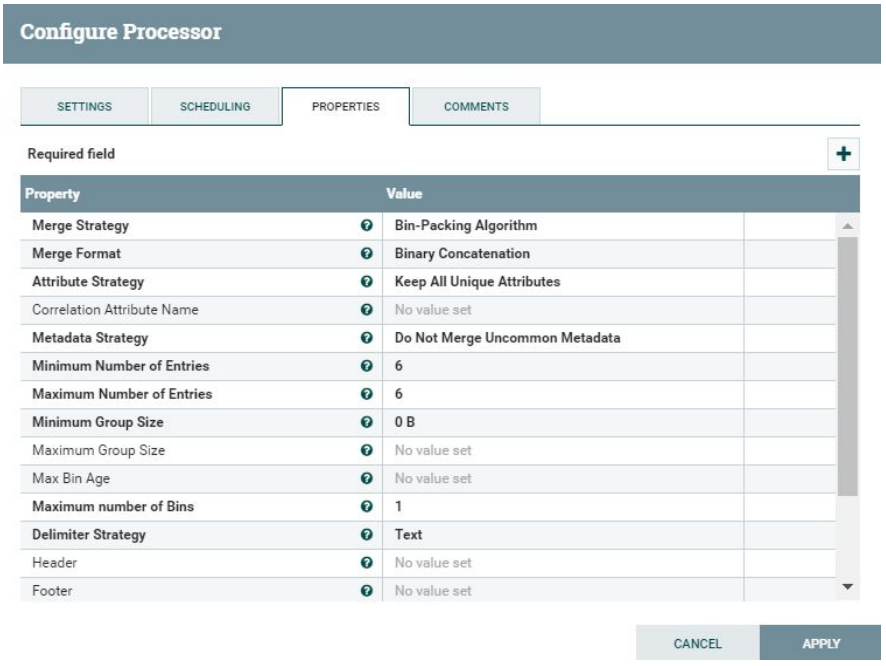
Joonis 7. Hõivatud mälu eraldamiseks mõeldud regulaaravaldis

Kõik GetFile ja ExecuteProcess käivitatakse samaaegselt ning pannakse tööle iga sekundi tagant, et andmebaasi mitte ületäita. Kogutud andmed salvestatakse FlowFile atribuutideks, et neid oleks hiljem lihtsam andmebaasi saata.



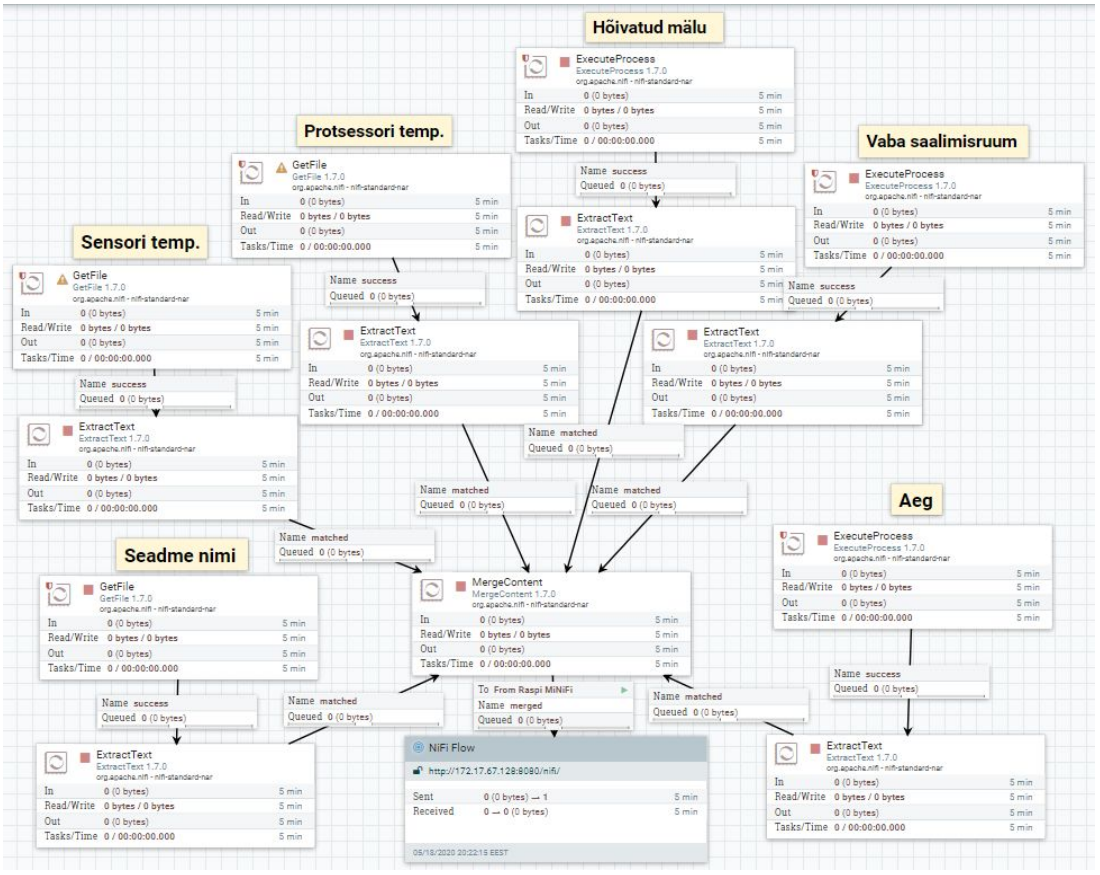
Joonis 8. Hõivatud mälu kogumise voog

Kahe protsessori vahel on nendevaheline seos, ehk mis tingimusel saadetakse andmed edasi, tingimuse mittetäitumise korral kustutatakse andmed mälust. Kui järgmine protsessor ei jõua andmeid nii kiiresti vastu võtta, kui eelmine sellele saadab, pannakse andmed järjekorda, et vältida andmekadu.



Joonis 9. MergeContent protsessori seaded

Kokkukogutud andmed on suunatud MergeContent protsessorisse, et need ühildada. Kuna vaikimisi MergeContent protsessor ei oota kõikide sissetulevate ühenduste FlowFile'de saabumist, tegi autor selle probleemi lahendamiseks ümbersõidu (*workaround*). Kõikide MergeContent protsessorisse sissetulevate ühenduste järjekorra suuruseks pandi 1 FlowFile. MergeContent protsessor paneb sissetulevad FlowFile'd konteineritesse ja samuti saab määrata, mitu sisendit saab ühel konteineril olla. Autor määras konteineri suuruseks 1 ning maksimaalseks ja minimaalseks sisendiarvuks 6 (vt. Joonis 9). See tähendab, et MergeContent protsessor ei tee enda tööd enne, kui tal on 6 sisendit. Järjekorra suurus 1 tagab selle, et kõik sissetulevad FlowFile'd jõuavad MergeContent protsessorisse. Vastasel juhul võib mõni protsess kiiremini töötada kui teised ja MergeContent konteineri sisendarv täidetakse ilma, et kõik FlowFile'd oleks MergeContent protsessorisse jõudnud. Kui andmed järjekorda ei mahu, jäävad nad eelmisesse järjekorda ootama.Viimaks saadetakse kokkukogutud andmed üle HTTP protokolli keskserverisse, kus andmed võtab vastu NiFi. Joonisel 10 on välja toodud kogu MiNiFi voog.



Joonis 10. MiNiFi voog

NiFi ja MiNiFi kooslus võimaldab süsteemi teha skaleeritavaks. Selle asemel, et MiNiFi seadmele manuaalselt konfiguratsioonifail asetada, saab lähteseadme ise konfiguratsioonifaili küsima panna. See võimaldab lähteseadmete arvu suurendada vastavalt soovile ning iga muudatus andmete kogumise viisis kandub peaaegu üheaegselt kõikidele lähteseadmetele. Selleks on appi võetud MiNiFi c2 tarkvara, mis töötab keskserveril. Lähteseade peab teadma,

```
# Hostname on which to pull configurations from
nifi.minifi.notifier.ingestors.pull.http.hostname=172.17.67.128
# Port on which to pull configurations from
nifi.minifi.notifier.ingestors.pull.http.port=8200
# Path to pull configurations from
nifi.minifi.notifier.ingestors.pull.http.path=/c2/config
# Query string to pull configurations with
nifi.minifi.notifier.ingestors.pull.http.query=class=iot-minifi-raspberry-agent
# Period on which to pull configurations from, defaults to 5 minutes if commented out
#1min
nifi.minifi.notifier.ingestors.pull.http.period.ms=60000
```

Joonis 11. Faili “bootstrap.conf” sisu

kust see vajaliku konfiguratsioonifaili saab, seega on vaja MiNiFi seadmel kaustas *conf*, failis “bootstrap.conf” defineerida parameetrid, mis on näidatud joonisel 11. Määrates MiNiFi c2 kuulama sama porti, kuhu lähteseade ühendub, saab lähteseade küsida konfiguratsioonifaili. Selleks tuleb MiNiFi c2 kaustas *conf*, failis “C2.properties” serveri pordiks määrata sama pordi number, mis sai sisestatud MiNiFi “bootstrap.conf” faili. Konfiguratsioonifaili hoitakse *~/minifi-c2-0.6.0-SNAPSHOT/files/iot-minifi-raspberry-agent* kaustas. Oluline on, et failitee viimane kaust ühtiks MiNiFi “bootstrap.conf” failis *nifi.minifi.notifier.ingestors.pull.http.query=class=* väärtusega.

4.1.2 NiFi voog

NiFi tegeleb andmete vastuvõtmise ning lihtsama töötlusega. Pärast MiNiFi andmete vastuvõtmist suunatakse voog RouteOnAttribute protsessorisse. NiFil on oma väljenduskeel ning seda on võimalik kasutada ka RouteOnAttribute protsessoris, mis võimaldab andmeid kasutaja defineeritud tingimustel edasi erinevatesse protsessoritesse suunata. Joonisel 12 on näidatud NiFi väljenduskeelt. Üheks tingimuseks on see, et keskkonna temperatuur on kõrgem kui 27°C. Selleks on kasutatud märksõna *divide*, mis võimaldab väärtust jagada.

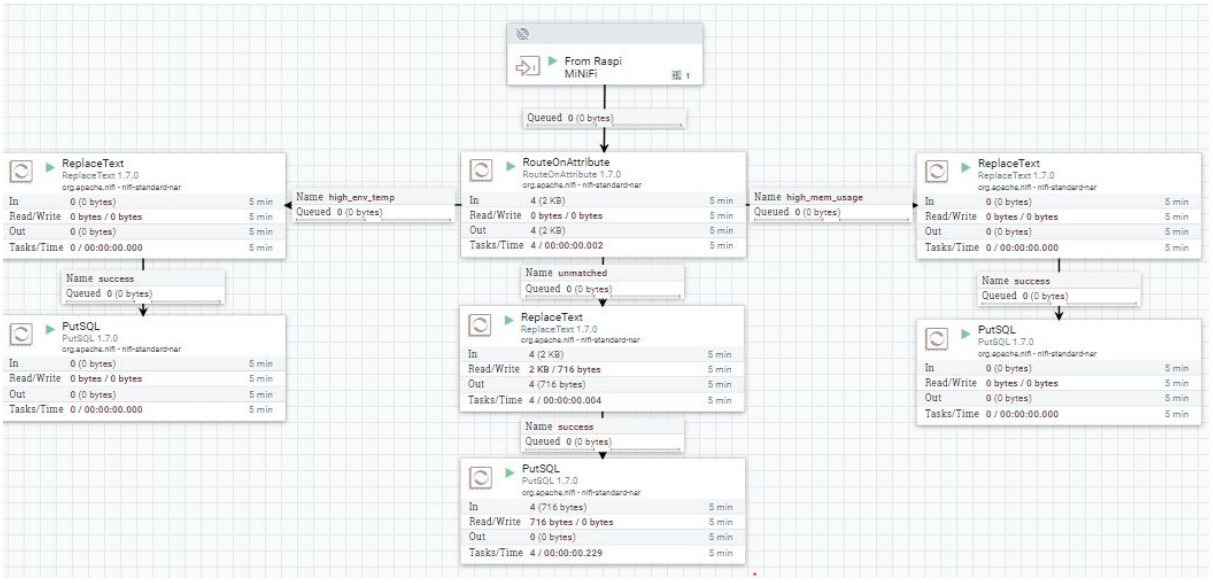
Autor on jaganud keskkonna temperatuuri arvuga 1000, et konverteerida temperatuur Celsiuse skaalale. Gt märksõna tähendab *greater than* ehk suurem kui. Teiseks

Joonis 12. RouteOnAttribute protsessori marsruutimise tingimused

tingimuseks on see, kui lähteseade on kasutanud rohkem mälu kui 600MB. Sellisel juhul piisas ainult gt märksõnast, sest MiNiFi seadmel käivitati programm Free argumendiga -m, mis konverteerib väärtused megabaitidesse. Tingimuse täitumise korral suunatakse andmed selleks ettenähtud tabelisse. Kui kumbki eelnevatest tingimustest pole täidetud suunatakse andmed üldtabelisse, et siiski jälgida kogutud infot. Pärast suunamist koostatakse ReplaceText protsessoris SQL päring andmete tabelisse sisestamiseks ning viimaks PutSQL protsessor suhtleb andmebaasiga. Päring andmete sisestamiseks üldtabelisse nägi välja selline:

INSERT INTO data
(device,time_sent,time_recieved,environment_temp_C,cpu_temp_C,used_memory_MB,free_swap_MB) VALUES ('${device.0}', '${time.0}', NOW(3),
'${env_temp.0:toNumber():divide(1000)}', '${cpu_temp.0:toNumber():divide(1000)}',
'${used_mem.3:toNumber()}', '${swap_free.1:toNumber()}')

Joonisel 14 on välja toodud NiFi voog.

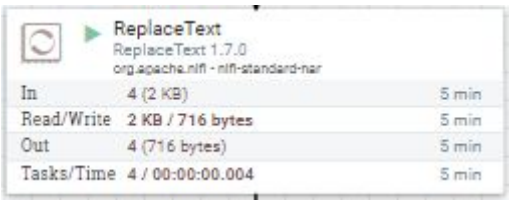


Joonis 14. NiFi voog

id	device	time_sent	time_recieved	environment_temp_C	cpu_temp_C	used_memory_MB	free_swap_MB
1	client1	2020-05-10 23:21:26.517	2020-05-10 23:21:28.287	22	51	348	99
2	client1	2020-05-10 23:21:28.707	2020-05-10 23:21:30.094	22	51	356	99
3	client1	2020-05-10 23:21:30.769	2020-05-10 23:21:32.129	22	49	359	99
4	client1	2020-05-10 23:21:32.831	2020-05-10 23:21:34.169	22	49	359	99
5	client1	2020-05-10 23:21:34.887	2020-05-10 23:21:37.310	22	49	360	99
6	client1	2020-05-10 23:21:35.906	2020-05-10 23:21:38.372	22	48	360	99
7	client1	2020-05-10 23:21:38.955	2020-05-10 23:21:40.411	22	48	360	99
8	client1	2020-05-10 23:21:40.993	2020-05-10 23:21:42.451	22	48	360	99
9	client1	2020-05-10 23:21:43.035	2020-05-10 23:21:44.548	22	48	369	99
10	client1	2020-05-10 23:21:45.071	2020-05-10 23:21:46.555	22	48	369	99

Joonis 15. Väljavõte üldandmebaasi esimesest 10st reast

Andmebaasina oli kasutatud MySQL andmebaasi ning andmeid hoiustati kujul nagu joonisel 15 on näidatud. Nii andmebaasi, lähteseadmete kui ka keskserveri ajavööndiks oli GMT+3 ning seadmed olid ajaliselt sünkroniseeritud Tartu Ülikooli NTP serveriga.



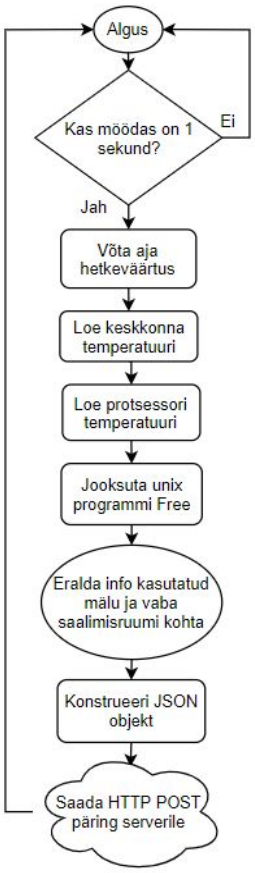
Joonis 16. ReplaceText protsessor

Iga protsessor näitab infot protsessitavate andmete koguse kohta. Täpsemalt, protsessor näitab infot mitu korda on ta infot vastu võtnud, mis koguses andmeid lugenud ja kirjutanud,

mitu korda väljastanud ning viimaks kui kiiresti töötluskorrad lõpule viidi, statistika on viimase 5 minuti kohta. Joonisel 16 on näha ReplaceText protsessori läbitud andmete kogust. Siinkohal tuleb ka mainida, et sissetulevad andmed on kogum MiNiFi GetFile ja ExecuteProcess protsessorite tagastatud andmetest.

4.1.3 Skriptiline lähenemine

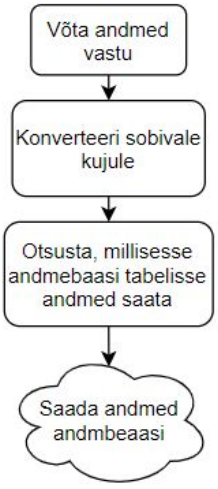
Andmetoru loomiseks on kasutatud ainult programmeerimiskeelt Python. Andmete teekond on täpselt samasugune nagu see oli NiFi ja MiNiFi puhul ehk lähteseadmed suhtlevad keskserveriga, mis omakorda suhtleb andmebaasiga. Lähteseadmete skripti puhul on kasutatud järgmisi teeke: json, os, datetime, requests ja time. Koostatud programm kogub täpselt samu andmeid mida MiNiFi. Nendeks on keskkonna temperatuur, protsessori temperatuur, hõivatud mälu, vaba saalimisruum ning algav kogumishetk, et kogu andmete elutsükkli liikumiskiirust mõõta. Skripti loogika plokkskeem on toodud välja joonisel 17.



Joonis 17. Lähteseadme andmekogumisskripti plokkskeem

Aja mõõtmist alustatakse esimesena, et võrrelda hiljem kui kiiresti kogub ja saadab andmeid skript MiNifiga võrreldes. Andmete kogumine toimub iga sekundi tagant nagu ka MiNiFi puhul. Andmete keskeserverisse saatmine toimub üle HTTP protokolli nagu ka MiNiFi puhul. Andmete saatmiseks testiti ka pealtnäha ebaefektiivsemat varianti, kus andmete kogumine toimus konstantselt ja saadeti neid alles siis, kui möödunud oli 1 sekund. Tulemused selle kohta asuvad peatükis 5.3.

Keskserveris võtab saadetud andmed vastu teegiga Flask ehitatud veebiserver. Programmis tehakse otsus andmete liigutamise osas nagu ka NiFi puhul, andmete teekond otsustatakse 3 faktori põhjal. Kasutaja jaoks defineeritud kõrge keskkonna temperatuur, kasutaja jaoks defineeritud kõrge mälukasutus ning eelmisele kahele tingimusele mittevastamine. Joonisel 18 on toodud eelnevalt kirjeldatud programmi plokkskeem.



Joonis 18. Serveri andmete vastuvõtmise plokkskeem

Kui ükski kasutaja defineeritud tingimus ei täitu, suunatakse andmed andmebaasi üldtabelisse. Esimese kahe tingimuse täitumise korral suunatakse andmed neile vastavasse andmebaasi tabelisse. Mõlemad tingimused on samad, mis MiNiFi-NiFi andmetoru puhul.

5. Analüüs

Analüüsis lähtub autor esimeses peatükis kirjeldatud hindamiskriteeriumitest. Samuti võrdleb kahte erinevat lähenemist lähtudes neljandas peatükis kirjeldatud kasutusloost.

5.1 Andmetoru üles seadmise keerukus

NiFi

NiFi ülesseadmine käib valutult. Piisab tarkvara allalaadimisest ning kasutajal on ligipääs kohe kõikidele funktsionaalsustele. MiNiFi Java versioon töötab samasugustel põhimõtel ning selle ülesseadmine käib sama valutult. Kõik protsessorid, mida toetab MiNiFi, on olemas ka NiFis. Kogu kasutusloo süsteemi ülesseadmine nii valutult aga ei käinud. Kui kasutaja on harjunud probleemidele programmaatiliselt lähenema, on esialgu võõras liigutada plokke ja neid omavahel ühendada. Üldpilti on esialgu keerulisem selgeks saada. Küll aga on kogu voog arusaadavam, kui koodi lugedes. Suunatud graafide põhimõttele tuginedes on selge aru saada, kust algab protsess ja mis protsess järgmisena käivitatakse. Suures pildis NiFi ja MiNiFi kasutamine on kergem, kuna suur hulk tööd tehakse kasutaja eest ära. Kasutaja peab ainult teadma, kuidas konkreetne protsess käib. Programmeerimisoskusi selle tarkvara kasutamine ei nõua. Andmetoru ülesseadmine võttis autoril kaua aega, sest kogu NiFi-MiNiFi süsteemist oli esialgu keeruline aru saada. Suur hulk ajast kulus sünkroonsuse saavutamisele andmete kogumisel, et neid andmebaasi saata. Lisaks nõuab regulaaravaldise efektiivne kasutamine aega ja kannatust, sõltuvalt andmete kogusest, struktuurist ja soovitud tulemusest.

Python

Pythoni, kui ka andmetoru üles seadmine Pythoniga on üsna muretu, kui kasutaja omab programmeerimisoskusi. Skriptiline lähenemine eeldab lisaks programmeerimisoskustele baasteadmisi veebiserveritest ja HTTP protokollist või mõnest muust suhtlusviisist, näiteks sokkliühendus või MQTT. Suureks abiks on ka olemasolevad teegid, mida Pythoni puhul on võimalik alla laadida paketihalduriga Pip. Skriptiliselt lähenedes saab kogu süsteemi

märgatavalt kiiremini püsti panna, küll aga on koodi raskem lugeda eriti, kui selle kallal peaks mitu inimest töötama. Andmetoru ülesseadmine sujus autoril kiiremini, kuna autor teadis, kuidas suhelda erinevate masinate vahel ja kuidas vajalikud andmed kätte saada.

Mõlema lähenemise puhul loeb ka veidi süsteemiadministreerimise oskus, kuna erinevate masinate omavaheliseks suhtluseks on vaja näiteks porte avada või mõndasi teenuseid tööle panna ja hallata.

5.2 Dokumentatsiooni põhjalikkus

NiFi

Dokumentatsioonis on põhjalikult kirjeldatud mida NiFi endast kujutab ja kuidas selle kasutamisega alustada. Lisaks on dokumentatsioonis kõikide protsessorite kui ka protsessorite seadete kohta kirjeldus. Küll aga ei pruugi sellest kuigi palju kasu olla, kui eelteadmised puuduvad. Võrreldes Pythoni kommuuniga pole NiFi oma veel kuigi suur. Selle tõttu on internetist abi otsimine raskendatud ja võib juhtuda, et spetsiifilisele murele niipea lahendust ei leia. Üks suurimaid keskkondi abi otsimiseks on Cloudera foorum [18], kus on üle 70 000 liikme, kuid foorum pole konkreetselt NiFi murede jaoks, vaid pigem üldiselt pilveteenuste ja suurandmete halduse murede jaoks. Stack Overflow foorumis oli seoses NiFiga kokku 3096 küsimust võrreldes Pythoni 1,4 miljoniga. NiFi enda suhtluskeskkonnas Slack oli veidi alla tuhande liitunu.

Python

Kasutatud teekide ja ka Pythoni enda kohta on põhjalik dokumentatsioon. Andmetoru on võimalik luua ka teiste teekidega, millega sama töö ära teha, seega andmetoru loomiseks ei pea ilmtingimata samu teeke kasutama. Hätta jäämise korral võib internetist abi üsna kiiresti leida, kuid see eeldab ka, et kasutaja teab, mida ta otsib.

5.4 Saatmiskiirus

Katsetulemused kajastavad 100 mõõtmise tulemusi. Skriptilises lähenemises hakatakse aega mõõtma vahetult enne andmete kogumist, MiNiFil hakatakse aega mõõtma samaaegselt koos andmete kogumisega, sest MiNiFi voos graafi tippudeks olevad protsessid jooksutatakse samaaegselt. Kõik seadmed on ajaliselt sünkroniseeritud TÜ NTP serveriga.

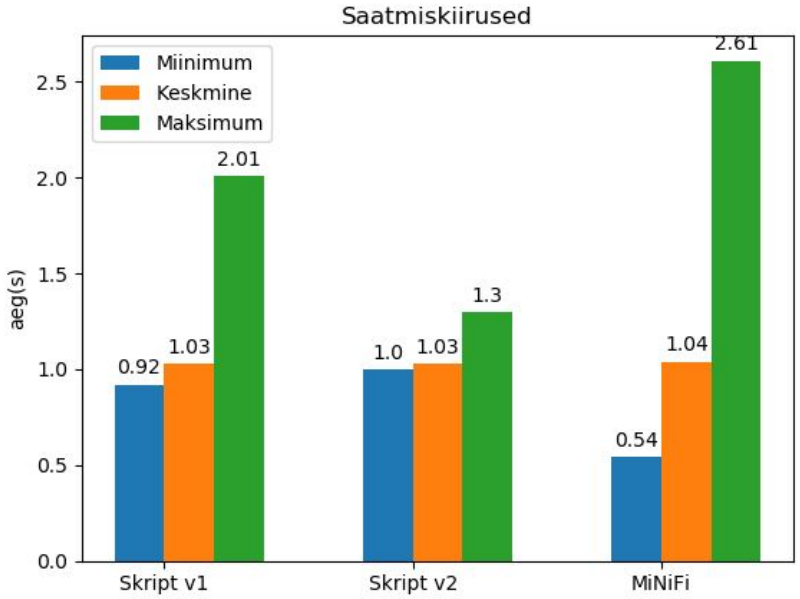
Skript v1 puhul on lähtutud loogikast, mis on näidatud joonisel 17. Skript v2 erineb Skript v1st selle poolest, et andmeid kogutakse konstantselt ja saadetakse edasi ainult 1 sekundi möödumisel. Skript v1 puhul oodatakse kõigepealt ühe sekundi möödumist ja siis kogutakse kõik andmed kokku ning saadetakse edasi.

Viis/Aeg(s)	Miinimum	Keskmine	Maksimum
Skript v1	0.92	1.03	2.01
Skript v2	1	1.03	1.3
MiNiFi	0.54	1.04	2.61

Tabel 1. Saatmiskiiruste tulemused

Tabelis 1 on toodud MiNiFi ja skriptilise lähenemise saatmiskiirused. Saatmishetkel oli MiNiFi puhul andmete suurus 400 baiti, see sisaldab kõikide protsessorite väljundit ja attribuute. Skriptide puhul oli andmete suuruseks 129 baiti. Keskmiste kiiruste järgi võrreldes on erinevus põhimõtteliselt olematu, küll aga oli MiNiFi ja skript v1 puhul saatmiskiirused ebastabiilsemad. Maksimume võrreldes võttis MiNiFil saatmine veidi üle kahe korra rohkem aega võrreldes skript v2ga ja 0,6 sekundit kauem võrreldes v1ga, samas kõige kiirem saatmine mõlemast peaaegu kaks korda vähem aega. Joonisel 19 on tulemuste illustreemiseks graafik.

MiNiFi puhul oli esimese mõõtmise aeg ühtlasi ka maksimum aeg. See võib tuleneda asjaolust, et esmasel käitamisel on süsteem aeglane ja kõikide lõimede töölepanekuks läheb veidi rohkem aega. Pärast MiNiFi tööle panemist, on ajaliselt soodsam lõim pausil olekust tööle panna kui seisatud olekust.



Joonis 19. Saatmiskiirused kasutusloo puhul

Konkreetse kasutusloo puhul ei ole kriitilise tähtsusega, et andmed jõuaksid kiiremini, kui 1 või 2 sekundit. Tihtipeale seadmete riknemine toimub ajapikku ning kõige olulisem on, et andmed oleks üldse olemas. Mõistagi ei ole efektiivne oodata andmeid minuteid, kuid kriitiline ajapiir sõltub valdkonnast.

Autor testis ka võrdluseks 1MB suuruse faili saatmist. Selle testi puhul saadeti andmeid ainult lähteseadme ja keskserveri vahel, andmeid andmebaasi masinasse ei saadetud.

Viis/Aeg(s)	**Miinimum**	**Keskmine**	**Maksimum**
Skript v1	0,8	1,02	2,74
Skript v2	0,95	1,02	2,81
MiNiFi	1,85	2	2,35

Tabel 3. 1MB suuruse faili saatmiskiirused lähteseadme ja keskserveri vahel

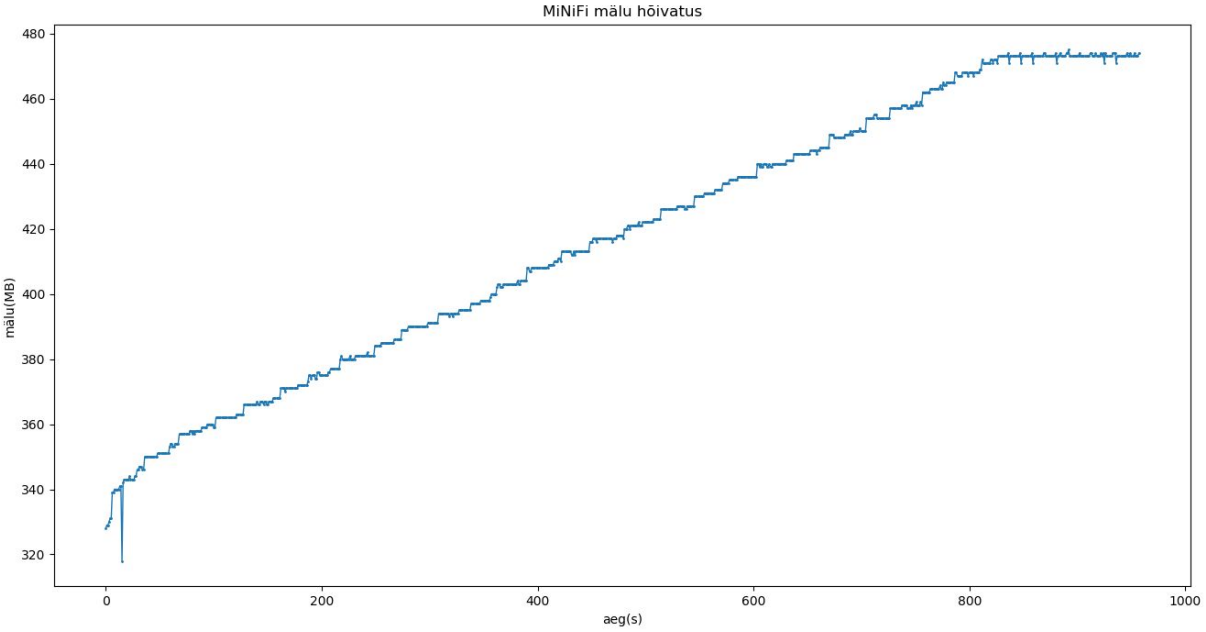
5.3 Ressursinõudlus

Mälukasutust jälgiti unixi programmiga Free. Infot Protsessori temperatuuri kohta saadi failist “Temp”, mis asub kaustas */sys/class/thermal/thermal_zone0.*

Viis/Mälukasutus (MB)	Miinimum	Keskmine	Maksimum
Skript v1	151	152.53	153
Skript v2	138	139.7	140
MiNiFi	318	474	475

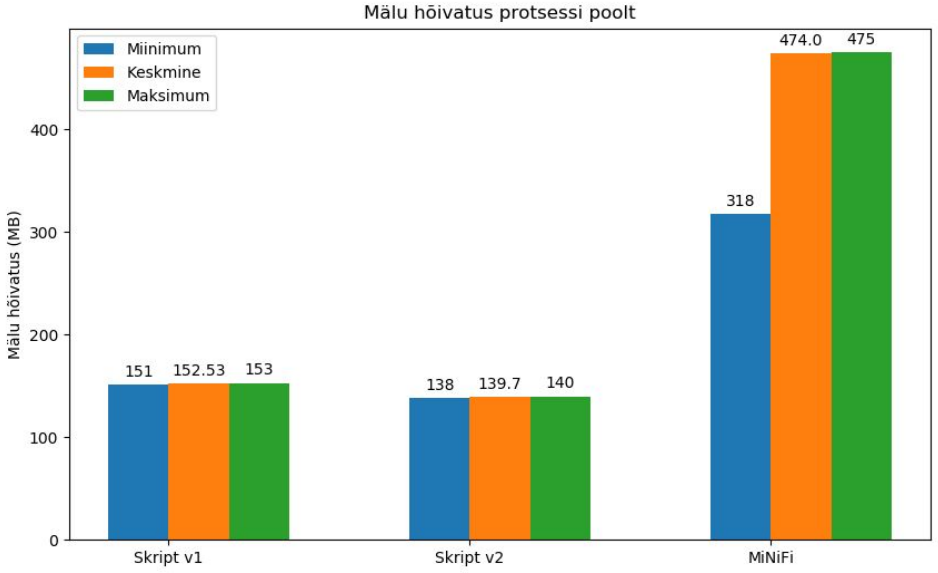
Tabel 4. Mälu hõivatus kasutusloo puhul

Skriptilise lähenemise mõõtmistulemused on saadud 100st mõõtmisest. Kuna MiNiFi hõivatud mälu hulk kasvas lineaarselt ja tõus jätkus pärast sadat mõõtmist, on MiNiFi mõõtmiste puhul lähtutud esmasest käitamisest kuni hetkeni, mil hõivatud mälu hulk enam ei kasvanud, see võttis aega 836 mõõtmist. Hõivatud mälu hulk kulmineerus 475MB juures ja edaspidi kõikus enamasti 475MB ja 473MB vahel, seega keskmiseks tulemuseks võib lugeda 474MB.



Joonis 20. MiNiFi mälu hõivatuse lineaarne kasv

Tabelis 4 on näha mälu hõivatuse mõõtmistulemused. Mälukasutuses tulevad erinevused selgelt välja. Siinkohal on ka oluline meeles pidada, et skriptilises lähenemises kasutati Pythonit ja MiNiFi-NiFi süsteem kasutab Javat. Java on tuntud oma kõrge mälukasutuse poolest. See-eest on Java ka kiirem ja efektiivsem võrreldes Pythoniga, kuna Java on kompileeritud keel ja Python interpreteeritud. Nii maksimumi kui ka keskmise mälu hõivatuse puhul jäi vahe 3.3 kordseks, miinimumi puhul oli vahe 2.3 kordne. Ilma kumbagi programmi jookustamata oli hõivatud mälu hulk 132MB. Joonisel 21 on tulemuste visualiseerimiseks loodud graafik.



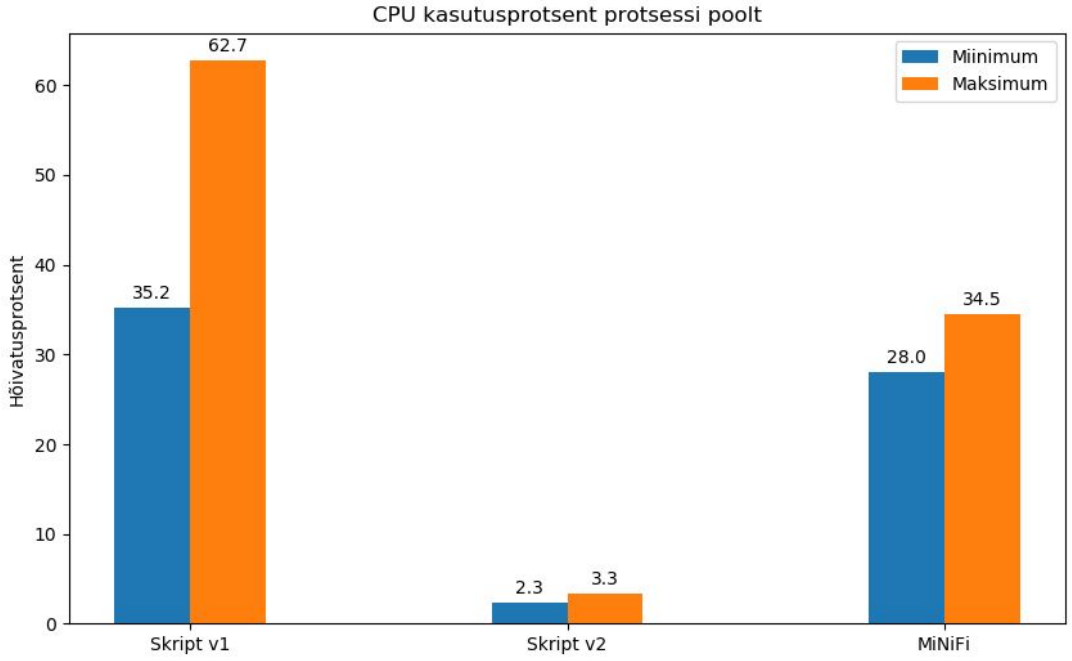
Joonis 21. Mälu hõivatus kasutusloo puhul

Protsessorikasutust jälgiti unixi programmiga Top. Tabelis 5 on välja toodud protsessorikasutuse protsendid. Kõik tulemused kajastavad protsessori ressursinõudlikkust protsessi poolt, mis käivitas kas skripti või MiNiFi. Protsessori hõivatusprotsent kajastab 4 tuuma tööd, ehk maksimaalne võimalik väärtus on 400%, mida väiksem protsent, seda parem. Tulemused olid üllatavad, sest skript v1 teeb pealtnäha vähem tööd, kuigi sai kõige kehvema tulemuse. See tuleneb tõenäoliselt sellest, et tsükklis kontrollitakse ülikiiresti aega, kas 1 sekund on möödunud, mitte ei kasutata näiteks Pythoni teegi *time* funktsiooni *sleep*. Mis seiskab kogu programmi töö kasutaja määratud ajaks.

Viis	Protsessi CPU hõivatus (%)
Skript v1	35,2-62,7
Skript v2	2,3-3,3
MiNiFi	28,0-34,5

Tabel 5. Protsessori hõivatuse protsendid

Protsessori ressursinõudlikkuse erinevus tuli selgelt välja. Siinkohal tuleb lisada, et MiNiFi kasutas lähteseadme kõiki nelja tuuma efektiivsemalt ära, jaotades kogu koormuse nelja tuuma vahel üsna võrdselt. Pythoni puhul kasutati ühte tuuma. MiNiFi puhul jäi protsessori tuumade keskmine hõivatus 28,0-34,5% juurde. Pealtnäha efektiivsema skripti puhul ehk skript v1 puhul oli keskmiseks protsessorihõivatuseks 35,2-62,7%. Skript v2 tulemus oli stabiilsem ja jäi 2,3-3,3% vahele. Joonisel 21 on illustreerimiseks tulemused.

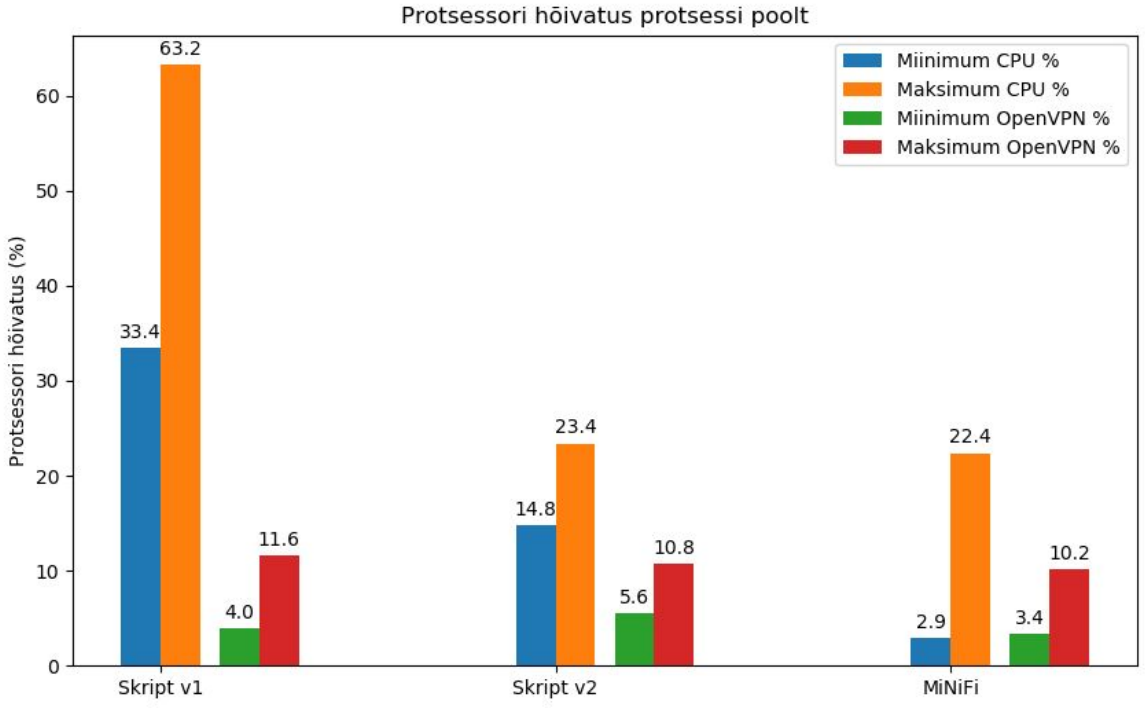


Joonis 21. Protsessori hõivatuse protsendid

Viis	Protsessi CPU hõivatus (%)	OpenVPN CPU hõivatus(%)
Skript v1	33,4-63.2	4-11,6
Skript v2	14,8-23,4	5,6-10,8
MiNiFi	5,9-22,4	3,4-10,2

Tabel 6. Protsessori hõivatuse protsendid 1MB suuruse faili saatmisel

Testides 1MB suuruse faili saatmist, oli ressursikasutuse poolest kõige optimaalsem MiNiFi, mille kõige väiksem CPU kasutus oli 5,9% ja kõige kõrgem 22,4%. Pealtnäha kõige optimaalsem skript, skript v1, nõudis protsessorilt kõige rohkem 33,4-63,2% ja keskmiseks jäi skript v2, mille protsessorikasutus jäi 14,8-23,4% vahele. Saates 1MB suurust faili, tõusis ka OpenVPNi protsessori nõudlus. Ka sel puhul oli MiNiFi optimaalsem võrreldes skriptidega. MiNiFi nõudis 3,4-10,2%, skript v1 4-11,6% ja skript v2 5,6-10,8%. Tulemused on tabelis 6 ja illustreeritud joonisel 22.



Joonis 22. Protsessori hõivatuse protsendid 1MB suuruse faili saatmisel

Protsessori temperatuur jäi mõlemal juhul umbes sama kõrgeks. Kasutusloo tulemused on toodud välja tabelis 7 ja 1MB suuruse faili saatmise tulemused tabelis 8.

Viis/Protsessori temp.(°C)	Miinimum	Keskmine	Maksimum
Skript v1	46,2	47,7	48,7
Skript v2	45,1	46,1	47,2
MiNiFi	46	47,8	53

Tabel 7. Protsessori temperatuur kasutusloo puhul

Keskmine temperatuur erines MiNiFi ja skript v2 puhul 1,7°C võrra. Maksimumide puhul oli vahe suurem, 5,8°C. MiNiFi ja skript v1 puhul olid tulemused põhimõtteliselt samasugused. MiNiFi puhul tõusis temperatuur üle 50°C ainult esimese paari mõõtmise juures. Tavaolekus oli protsessori temperatuur 41°C.

Viis/Protsessori temp.(°C)	Miinimum	Keskmine	Maksimum
Skript v1	48,3	49,3	51,0
Skript v2	48,1	49	50
MiNiFi	49.1	50.2	53,3

Tabel 8. Protsessori temperatuur 1MB suuruse faili saatmisel

Ühe MB suuruse faili saatmisel tõusis keskmine temperatuur alla 3 kraadi. Faili saatmisel tõusid ka skriptide maksimaalsed temperatuurid 50°C piirini. Seekord oli ka MiNiFi keskmine temperatuur veidi üle 50°C, skriptide omad jäid alla selle. Minimaalne temperatuur oli skriptide puhul väga sarnane, skript v1 oma 48,3°C ja skript v2 oma 48,1°C. MiNiFi miinimum temperatuur oli 1°C võrra skript v2 omast kõrgem.

5.4 Lähtudes kasutusloost

Peatükis 4 kirjeldatud kasutusloo puhul on äärmiselt oluline, et andmed ei läheks kaduma ja säiliks andmete terviklikkus. Kuna kogu tehase heaolu sõltub laias laastus andmetest, on vaja garanteerida, et kõik kogutud andmed jõuavad kindlasti lähtekohast sihtkohta. Et seda

võimaldada, on oluline, et kogutakse andmeid lähteseadme kohta, et vältida seadme riknemisest tingitud andmekadu.

5.4.1 MiNiFi-NiFi

MiNiFi-NiFi kooslus on sellist stiili kasutusloo realiseerimiseks hea valik, sest on tõrkeid taluv ja skaleeritav. Selle tarkvara teeb veakindlaks asjaolu, et andmed säilitatakse reeglina igas töötlusetapis välja arvatud juhul, kui kasutaja on teistmoodi defineerinud. Andmeid hoitakse mälus, kuid on võimalik seada konkreetne piir, millal võetakse saalimisruum kasutusele. Lisaks hoitakse töötluse ajal FlowFile'de sisu andmekandjal, et teha mahukate andmete töötlus kiiremaks ja riketele talutavamaks [11]. Kui vahepeal peaks internetiühendus katkema, hoitakse kõik andmed alles kuni riistvara seda võimaldab. NiFi suudab kirjutada ja lugeda andmekandjalt paralleelselt ja samuti käitab kõik esmased protsessid samaaegselt, mis teeb programmi töö kiireks. Teine oluline faktor on see, et MiNiFi-NiFi kooslus võimaldab teha süsteemi skaleeritavaks. Pole oluline, mitu lähteseadet kasutaja soovib lisada, see ei mõjuta seadmete jõudlust ning ajakulu uute seadmete konfigureerimisega on võrdne ühe seadme konfigureerimise ajakuluga, sest kõik lähteseadmed küsivad ise keskserverilt konfiguratsiooni. Tabelis 1 välja toodud saatmiskiiruste tulemused konkreetset kasutuslugu negatiivselt ei mõjuta, kuna kasutusloo puhul ei ole kriitiline kiirus, vaid töökindlus. Mõistagi on seadmete eluea prognoosimine raskendatud, kui andmeid saadetaks näiteks kauem kui minut.

5.4.2 Skript

Skriptiline lähenemine võib sellises valdkonnas puudulikuks jääda. Kasutades skriptimist saab kogu süsteemi väga kiiresti püsti panna. Probleem aga tekib rikketaluvuses ja skeleeritavuses. Internetiühenduse katkedes kaotatakse andmed, mis on antud kontekstis võtmetähtsusega. Kui andmete kogumises soovitakse muudatusi teha, on vaja iga seadme koodi eraldi muuta. Kui aga seadmeid on kümneid või sadu, võtab see kaua aega. Üks võimalustest oleks kasutada näiteks sellist tarkvara nagu Ansible, millega oleks võimalik automatiseerida uuendatud koodi paigaldamine lähteseadmele. Olgugi, et Ansible võimaldab paralleelset seadmete konfigureerimist võtab see kauem aega, kuna kõigepealt on vaja läbi

SSH sisse logida, uuendatud kood alla laadida ning siis käivitada. Maksimaalse paralleelsete seadistamiste arvu piirab Ansible puhul riistvara. NiFi-MiNiFi ressursinõudlikkus on minimaalne, sest piisab ühest päringust MiNiFi c2 serverile.

6. Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli ehitada andmetoru kasutades tarkvara Apache NiFi ning võrrelda seda tavapärase skriptilise lähenemisega. Võrdlemiseks kasutas autor peatükis 1.3 väljatoodud hindamiskriteeriume ning samuti lähtus peatükis 4 kirjeldatud kasutusloost. Töö käigus loodi tarkvaraga Apache NiFi ja MiNiFi andmetoru ning võrreldi seda programmeerimiskeeles Python ehitatud andmetoruga. Andmetoru loodi kokku nelja seadme vahele. Lähteseadmeteks, ehk andmete kogujateks oli kaks Raspberry Pi arvutit ning andmete kogumisel ja hoiustamisel kasutati kahte Tartu Ülikooli pilves asuvat masinat. Lähteseadmed asusid geograafiliselt teises kohas võrreldes Tartu Ülikooli pilves asuvate masinatega ja samuti ka teises võrgus. Nendevaheliseks suhtluseks kasutati VPN ühendust ning lähteseadmetel oli 50MB üles- ja allalaadimiskiirusega internetiühendus.

Lähtudes ainult hindamiskriteeriumitest, sõltub valik sellest kuidas on andmetoru ehitatud. Väiksemate andmemahtude poolest oli nii skriptide kui ka MiNiFi-NiFi keskmised saatmiskiirused võrdsed, andmeid saadeti iga sekundi lähteseadmelt keskserverisse ning sealt omakorda andmebaasi, mis asus eraldi masinas. Skripti puhul oli andmete koguseks 129 baiti ning MiNiFi puhul 400 baiti. Erinevus seisnes selles, et MiNiFi puhul saadeti edasi kõikide protsesside väljundid ning attribuudid. Pythoni puhul oli võimalus selekteerida spetsiifilised infokillud, mida edasi saata. Küll aga nõudsid skriptid oluliselt vähem mälu kui MiNiFi, seda pea 3 korda vähem. MiNiFi kasutas mälu käivitades 318MB jagu ning peale seda hakkas lineaarselt kasvama, 800 sekundit hiljem jäi tulemus 474MB juures pidama, niiet keskmiseks mälukasutuseks arvestatati 474MB. Protsessorikasutuse juures tekkis huvitav tulemus. Mõlema skripti protsessorinõudlikkus erines üle 10 korra. Skriptid erinesid selle poolest, et skript v1 puhul oodati esmalt sekundi möödumist ning siis koguti kõik andmed kokku ja saadeti edasi. Skript v2 puhul koguti andmeid konstanselt ning saadeti alles sekundi möödumisel. Pealtnäha efektiivsem skript v1 saavutas aga halvema tulemuse protsessorihõivatuses. MiNiFi tulemus jäi skriptide tulemuste vahele ning erines ka selle poolest, et MiNiFi kasutas lähteseadme kõiki nelja protsessorit efektiivsemalt ära. Protsessori temperatuur tõusis kõige kõrgemaks MiNiFi puhul, kus keskmiseks temperatuuriks oli 47,8°C. Tavaolekus oli protsessori temperatuur 41°C.

Tulemused olid teistsugused, kui andmete kogust tõsteti. Katse viidi läbi 1MB suuruse failiga ning see test tehti ainult lähteseadme ja kekserveri vahel. Selle katse puhul tekkis saatmiskiirustes suurem vahe sisse. MiNiFi tulemus oli skriptidest ligikaudu sekundi võrra kehvem, skriptide keskmine saatmiskiirus oli 1,02 sekundit ja MiNiFi tulemus 2 sekundit. Mälu kasutus oli MiNiFil endiselt suurem ning skriptidel peaaegu võrdne. MiNiFi ja skriptide mälukasutuse erinevus sõltub suuresti sellest, et need on kirjutatud erinevates programmeerimiskeeltes, MiNiFi Javas ning skript Pythonis. Protsessorikasutuses olid tulemused mõnevõrra üllatavad. Sel korral oli MiNiFi tulemus parim ja pealtnäha efektiivsem skript oli kõige kehvem. Kuna andmemaht oli suurem, tõusis ka võrguliikluse hõivatus. Märgatavalt rohkem hakkas protsessori jõudlust kasutama OpenVPN teenus, millega oli ühendatud Tartu Ülikooli sisevõrku. Kõikide andmetorude puhul olid tulemused üsna võrdsed, kuid MiNiFi andmete kogumise ja saatmise ajal oli tulemus parim.

Eeldades, et kasutaja omab programmeerimisoskusi, on andmetoru loomine Pythoniga lihtsam kui NiFiga, sest esialgu on kogu NiFi-MiNiFi süsteemist raskem aru saada. Olgugi, et NiFi dokumentatsioon on väga põhjalik, ei ole seda ilma eelteadmisteta lihtne mõista ning aru saamiseks peab hulgaliselt erinevaid protsesse läbi proovima. Üldpildis on NiFi kasutamine lihtsam, sest suur osa tööst tehakse kasutaja eest ära. Kasutaja peab lihtsalt teadma, kuidas teatud protsessid töötavad. Internetist abi leidmine on palju raskem NiFi puhul, sest kommuun pole veel kuigi suur. Stack Overflow foorumis on NiFi kohta küsitud veidi üle 2000 küsimuse, millele on aktsepteeritud vastus, Pythoni puhul on see arv veidi üle miljoni.

Lähtudes kasutusloost, on selgelt parem valik NiFi, seda rikketaluvuse ja skaleeritavuse tõttu. Kuna sellise stiiliga kasutusloo puhul on andmed võtmetähtsusega, ei saa lubada andmete kadumist. NiFi-MiNiFi süsteem on tõrkeid taluv, kuna isegi internetiühenduse katkemise puhul säilitatakse andmete olemasolu hoides andmeid mälus või vajadusel andmekandjal, kuni teine osapool on võimeline neid jälle vastu võtma. Andmeid säilitatakse nii kaua, kuni riistvara seda võimaldab. Teine oluline aspekt on skaleeritavus. Kuna aeg-ajalt on vaja andmete kogumise viisis muudatusi teha või üldsegi uus seade süsteemi lisada on oluline, et muudatus viidaks ellu kiiresti. MiNiFi puhul on võimalus panna lähteseadmed ise

konfiguratsioonifaili küsima, mis tähendab, et muutus andmete kogumisviisis on sisuliselt sama kiire, ükskõik kui palju lähteseadmeid on.

Viited

[1] Jürisoo K. On the Role of Agile Software Development Practices in Software Process Improvement. TÜ arvutiteaduse instituudi magistritöö. 2017. https://comserv.cs.ut.ee/home/files/Jurisoo_tarkvaratehnika_2017.pdf?study=ATILoputoo&reference=393C027E6454DD3525C23D0ACC2DA315FD0C5A98 (08.03.2020)

[2] Vene S. DEVOPS Juurutamine Suurettevõttes. TalTech Tarkvarateaduse instituudi magistritöö. 2017. https://digikogu.taltech.ee/en/Download/31989650-996d-4f43-9a6c-c7dedefc0d11 (08.03.2020)

[3] Seddon P.B., Constantinidis D., Tamm T., Dod H. How does business analytics contribute to business value? *Information Systems Journal* Volume 27, issue 3, 2016, 237-269 https://onlinelibrary.wiley.com/doi/full/10.1111/isj.12101 (08.03.2020)

[4] Palmer A. From DevOps to DataOps, 2015

https://www.tamr.com/blog/from-devops-to-dataops-by-andy-palmer/ (08.03.2020)

[5] Predictive maintenance explained, Reliableplant ajakiri (s.a) https://www.reliableplant.com/Read/12495/preventive-predictive-maintenance (02.04.2020)

[6] Amazon Web Servivces, What is DevOps? (s.a)

https://aws.amazon.com/devops/what-is-devops/ (17.05.2020)

[7] Capizzi A., Distefano S., Mazara M. From DevOps to DevDataOps: Data Management in DevOps Processes, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment,* Springer International Publishing, 2020, 52-62

https://www.researchgate.net/publication/338672377_From_DevOps_to_DevDataOps_Data_Management_in_DevOps_Processes (14.03.2020)

[8] Patterson. P DataOps: Applying DevOps to Data - Continuous Dataflows Built With a DataOps Platform (24.05.2018)
https://dzone.com/articles/dataops-applying-devops-to-data-continuous-dataflo (17.05.2020)

[9] Christiansen B. A Complete Guide To Predictive Maintenance (06.02.2019)

https://limblecmms.com/blog/predictive-maintenance/ (18.05.2020)

[10] Christiansen B. 3 Main Types Of Maintenance Strategies (Side-by-side Comparison) (04.12.2018)
https://limblecmms.com/blog/3-main-types-of-maintenance-strategies/ (18.05.2020)

[11] The Apache Software Foundation - NiFi Documentation (18.03.2020)

https://nifi.apache.org/docs/ (10.05.2020)

[12] Spann T., LoPresto A. Intelligently Collecting Data at the Edge – Intro to Apache MiNiFi (18.04.2019)
https://www.slideshare.net/Hadoop_Summit/intelligently-collecting-data-at-the-edge-intro-to-apache-minifi-141236290 (09.05.2020)

[13] Zombrisky C. Apache Nifi Processors in version 1.7.0 (27.06.2018)

https://www.nifi.rocks/apache-nifi-processors-version-1.7.0/ (09.05.2020)

[14] Circuit Basics, RASPBERRY PI DS18B20 TEMPERATURE SENSOR TUTORIAL (s.a)
https://www.circuitbasics.com/raspberry-pi-ds18b20-temperature-sensor-tutorial/ (18.05.2020)

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Kristofer Kurvits**,

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

Andmete reaalajas kogumise võrdlemine kasutades Apache NiFit ja Pythonit,

mille juhendaja on Pelle Jakovits,

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.

Kristofer Kurvits

16.05.2020