TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Kaarel Randorg 142845IAPB

ETL VAHENDITE VÕRDLUS HADOOPI RAAMISTIKUL TELEKOMMUNIKATSIOONIETTEVÕTTE NÄITEL

bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2017

Autorideklaratsioon

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

Autor: Kaarel Randorg

22.05.2017

Annotatsioon

Antud töö eesmärgiks oli võrrelda kolme ETL protsesside tegemise vahendit Hadoopi raamistikul. Aluseks võeti juba realiseeritud lahendus telekommunikatsioonieettevõttele ning pakuti välja kaks alternatiivset lahendust. Olemasolev süsteem on realiseeritud kasutades Apache Falconit ning alternatiivseteks lahendusteks pakuti välja Apache Nifi ning Talendi Open Studio.

Kõik lahendused realiseeriti ning lahendusi hinnata nelja kriteeriumi järgi. Lõpuks võrreldi kolme lahendust omavahel kasutades AHP meetodit ning leiti parim lahendus.

Kokkuvõttes tuli välja, et mõlemad alternatiivsed lahendused on paremad, kui praegune lahendus. Parim lahendus oli Apache Nifi ning paremuselt teine oli Talendi Open Studio, kuid on ka juhtumeid, kus Talendi lahendus võib olla sobilikum, kui Nifi.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 32 leheküljel, 6 peatükki, 12 joonist, 11 tabelit.

Abstract

Comparison of ETL tools in the Hadoop framework on the example of a telecommunications company

The aim of this thesis is to compare three different tools to create an ETL process in the Hadoop framework. The original solution is based on a already implemented system for a telecommunications company. Two alternative solutions are proposed, one which uses Talend Open Studio and another one, which uses Apache Nifi.

All of the solutions are implemented in this thesis and then evaluated based on four criteria: how hard it is to implement this solution, how hard it is to debug this solution, how good is the performance of this solution and how much functionality this solution has. Finally all three solutions are compared to each other using AHP technique to find the best solution.

It turned out, that both of the alternative solutions were better than the current solution, which uses Apache Falcon. The best overall solution was Apache Nifi and the second best was Talend Open Studio. But in some cases, it is recommended to use Talend Open Studio.

The thesis is in Estonian and contains 32 pages of text, 6 chapters, 12 figures, 11 tables.

Lühendite ja mõistete sõnastik

API	„*Application Programming Interface* ehk rakendusliides on reeglistik olemasoleva valmisprogrammiga suhtlemiseks” [19] .
CSV	„*Comma-separated values* e komaga eraldatud väärtused on faililiik, mis sisaldav tabelikujulisi andmeid tavalises tekstifailis” [30] .
DAG	Directed Acyclical Graphs, suunatud atsüklilised graafid ehk graaf, kus suvalisest tipust ei leidu teed samasse tippu tagasi ehk tsükliteta graaf
DDL	*Data definition language* või *data description language* e andmete defineerimise keel, mida kasutatakse andmebaasides andmebaasi skeemi määramiseks [23] .
DML	*Data manipulation language* e andmete manipuleerimise keel, mida kasutatakse andmete valimiseks, sisestamiseks, kustutamiseks või uuendamiseks andmebaasides [22] .
ETL	*Extract, Transform, Load*. Andmebaasides kasutatav protsess, kus laaditakse andmed välisest andmeallikast (*Extract*), siis muundatakse selliseks, et neil oleks kindel struktuur (*Transform*) ja siis laaditakse andmebaasi (*Load*) [26] .
HDFS	Hadoop Distributed File System, Hadoopi poolt kasutatav failisüsteem.
HQL	Hive Query Language, SQL-laadne keel Hive'is päringute tegemiseks
HTTP	*Hypertext Transfer Protocol* e hüperteksti edastusprotokoll.
JDBC	*Java Database Connectivity* on API Java keelele, mis määrab, kuidas klient saab andmebaasiga suhelda.
JSON	*JavaScript Object Notation* on lihtsustatud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal [29] .
Schema on read	Andmete skeem määratakse andmete andmebaasist lugemisel ehk andmed on juba andmebaasi salvestatud ilma kindla skeemita [8] .
Schema on write	Andmete skeem määratakse andmete andmebaasi kirjutamisel [8] .
SQL	*Structured Query Language* e struktuurpäringukeel on andmebaasi päringukeel

TCP	*Transmission Control Protocol* e edastusohje protokoll
URL	*Uniform resource locator* e universaalne ressursilokaator e internetiaadress
Web crawler e veebiroomaja	Robotprogramm, mis otsib veebis kindla ja korrapärase meetodiga uusi veebidokumente ja lisab leitud tulemused erinevatesse andmebaasidesse[1] .
XML	*Extensible Markup Language* e laiendatav märgistuskeel.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Telekommunikatsioonieettevõtetel on palju erinevaid süsteeme, mida kasutavad nii töötajad kui ka kliendid. Süsteemide kasutamisel logitakse maha ka kasutaja tegevused, et hiljem oleks ülevaade ning saaks tõestada, kes mida tegi. Kuid kuna süsteeme on palju ning logimise viis ja struktuur võivad süsteemi lõikes erineda, siis nõuab iga kasutaja tegevuse logiandmetest ülesleidmine pingutust.

1.1 Töö eesmärk

Siin töös antakse ülevaade probleemi lahendusest, mis on realiseeritud Hadoopi raamistikul. Kuna Hadoopi maailm on kiiresti arenev ning aasta tagasi valitud lahendused võivad tänaseks juba aegunud olla ning parem variant valikus olla, siis on tähtis olla uute tehnoloogiatega kursis. Siin töös uuritakse kas praegusele lahendusele, mille arhitektuur mõeldi välja juba üle aasta tagasi, on tekkinud paremaid ning mugavamaid alternatiive.

1.2 Metoodika

Töös realiseeritakse nii praegune lahendus, mis kasutab Apache Falconit kui ka kaks alternatiivset lahendust, millest üks kasutab Apache Nifit ja teine Talendi Open Studiot. Kõik kolm eelnimetatud komponenti võimaldavad teha ETL (*Extract*, *Transform*, *Load*) protsesse, millest koosnevad lahendused. Pärast lahenduste realiseerimist hinnatakse lahendusi nelja kriteeriumi põhjal: realisatsiooni keerukus, vea otsimise keerukus, jõudlus ja funktsionaalsus. Kui kõik lahendused on realiseeritud ja hinnatud, siis võrreldakse kõiki lahendusi omavahel kasutad AHP meetodit.

1.3 Ülevaade tööst

Töö koosneb viiest osast.

Esimeses osas kirjeldatakse lühidalt kõiki komponente, mida töös kasutatakse ning antakse ka ülevaade mis asi on Hadoop ning räägitakse selle ajaloost. Pannakse paika, milline on näiteülesanne, mida kõik lahendused peavad realiseerima.

Töö teises osas tutvustatakse praeguse lahenduse realisatsiooni ning hinnatakse antud lahendust. Kolmandas ja neljandas osas tehakse sama, vastavalt Talendi Open Studio ja Nifi kohta.

Töö viimases ehk viiendas osas võrreldakse kõiki kolme lahendust kasutades AHP meetodit.

2 Ülevaade tehnoloogiatest ja näiteülesandest

Siin peatükis tutvustatakse Hadoopi erinevaid komponente, mida siin töös kasutatakse. Lisaks antakse ülevaade ka näiteülesandest, mille põhjal kõik lahendused realiseeritud on.

2.1 Mis on Hadoop

„Hadoop on avatud lähtekoodiga, Java-põhine programmeerimise raamistik, mis võimaldab töödelda ja hoiustada väga suuri andmehulki hajusarvutamise keskkonnas [2] .” Hadoopis on suur andmestik jagatud väikesteks osadeks ehk plokkideks, mida töötlevad paralleelselt mitu arvutit[3] . Need plokid on replitseeritud mitme arvuti peale, seetõttu kui mõnes arvutis tekib rike, siis ei teki andmekadu. Kuna Hadoopis olevad andmed ei ole organiseeritud, see tähendab et kui tavapärases andmebaasis on rea ja veeru alusel kõik andmed struktureeritud, siis Hadoopis on võimalik hoiustada nii struktureeritud ja mittestruktureeritud faile [3] .

2.2 Hadoopi ajalugu

Hadoop loodi Doug Cuttingu poolt, kes on ka Apache Lucene looja. Hadoop sai alguse Apache Nutchi projektist, mis on avatud lähtekoodiga otsingumootor. Nutchi projekt algas 2002. aastal ning töötavad veebiroomaja (web crawler) ja otsingusüsteem olid peagi valmis. Kuid nad avastasid, et nende arhitektuur ei skaleeru miljarditele veebilehtedele. Google avaldas 2003. aastal arhitektuuri kirjelduse oma hajusast failisüsteemist GFS'ist. Just sellelaadne failisüsteem aitaks Nutchil hoiustada suuri andmeid, mida tekitas nende veebiroomaja ja indekseerimise protsess. 2004. aastal hakkaski Nutchi projekt realiseerima sellist avatud lähtekoodiga failisüsteemi, mille nimeks sai Nutch Distributed Filesystem (NDFS) [18] .

2004. aastal avaldas Google ülevaate ka MapReduce'i arhitektuurist ning Nutchi tiim hakkas hakkas ka seda realiseerima. 2005. aasta keskpaigaks oli Nutchi tiimil see valmis ning kõik Nutchi algoritmid tehti ümber nii, et need töötaks NDFS'i ja MapReduce'i peal. Saadi aru, et NDFS ja MapReduce on kasutatavad ka väljaspool otsingutööriistu ning seetõttu tehti nende jaoks 2006. aasta veebruaris eraldi projekt Hadoop, mis tol ajal oli veel Nutchi alamprojekt. 2008. aasta jaanuaris sai Hadoopist eraldiseisev Apache projekt [18] .

2.3 Probleemi lahendava süsteemi arhitektuur

Andmejärve tuuakse kõik rakenduste logid, kus on olemas isikuandmed. Logist leitakse üles tegutseja (isik) ning tegevus, mida ta tegi. Kui isikuks on ettevõtte töötaja ning tegevuseks on mõne kliendi andmete vaatamine, siis tuvastatakse logidest ka klient. Kui on vaja tõestada, et mingi isik tegi mingi tegevuse (näiteks tellis lisa internetimahtu või vaatas kliendi andmeid, mis polnud tööülesandega seotud), siis on sellisest süsteemist lihtne isiku järgi otsida kõik ta tegevused üle kogu rakenduste ning sellepõhjal juba sama kirje leida muutmata kujul alglogiandmetest, millel on tõestusväärtus.

Selline süsteem täidab kahte andmebaasidega seotud funktsiooni, nii andmejärve kui ka andmelao oma. Kõik andmed tuuakse muutmata kujul HDFS'i, kuhu need jäävad ettemääratud ajaks. Neid andmeid seejärel muundatakse ühtsele kujule ning sisestatakse tabelisse, mille peale saab juba aruandlustabeleid teha. Sellist protsessi nimetatakse ETL protsessiks.

ETL tuleneb kolmest ingliskeelsest sõnast *Extract, Transform* ja *Load*. Need kolm sõna moodustavadki ETL protsessi kolm komponenti. *Extract* faasis laaditakse andmed välisest süsteemist kohalikku süsteemi, siin töös vaadeldava protsessi juures HDFS'i (*Hadoop Distributed File System*). *Transform* faasis muundatakse andmed ühtsele kujule ning *Load* faasis laaditakse muundatud andmed andmebaasi tabelisse, siin töös Hive'i tabelisse [26] .

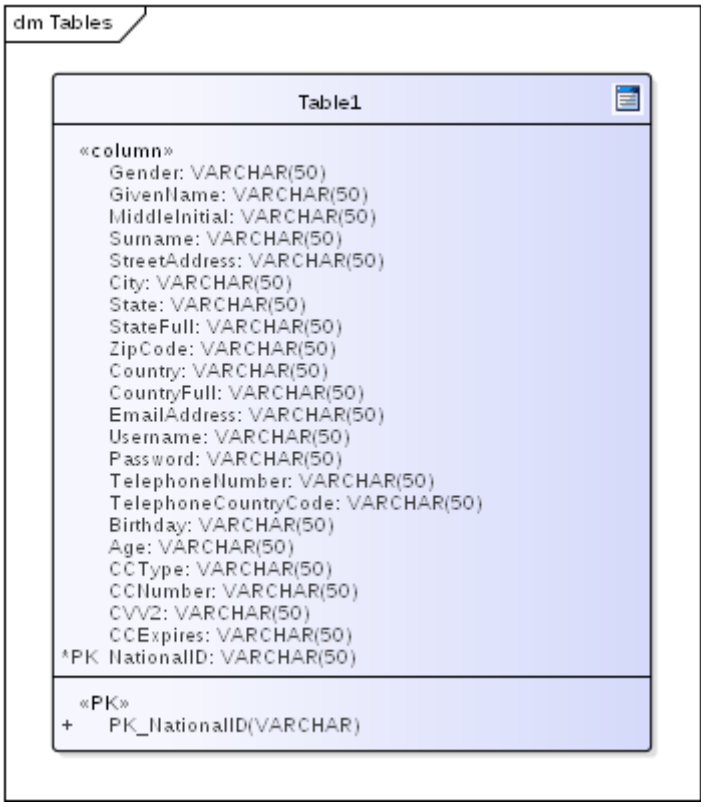
2.4 Näidislahendus, mille põhjal lahendusi võrreldakse

Näidisprotsessiks mõeldi välja võimalikult sarnane protsess peatükis 2.3 kirjeldatud protsessile. On üks faktitabel ning 4 dimensioonitabelit. Algandmed saadakse nii Syslogi[1] käest, relatsioonilisest andmebaasist ning ka staatilistest CSV (*Comma-separated values*) formaadis failidest.

2.4.1 Algandmed

Kokku on 5 algallikat, kust andmeid saadakse.

Klientide andmed tuuakse sisse MySql andmebaasist. Selles tabelis on 23 veergu ning 49973 rida. Testamdmed saadi leheküljelt www.fakenamegenerator.com/, mis genereeris 22 veergu 23'st. Autori poolt lisati andmetele isikukoodi veerg, mis genereeriti soo, sünnikuupäeva ja sünniasukoha alusel.



Joonis 1: Tabeli clients sisu

[1] Syslog on sõnumite logimise standard, see võimaldab eraldada tarkvara, mis tekitab logisõnumeid, süsteemi, mis salvestab neid ja tarkvara, mis analüüsib neid [17] .

Andmed erinevate süsteemide ja sündmuste kohta ning ka töötajate kohta saadakse CSV failist. Süsteemide tabel koosneb 5 veerust, kus on määratud unikaalne identifikaator, süsteemi kood, süsteemi kirjeldus, lisakommentaar ning süsteemi keskkond. Sündmuste tabel koosneb samuti 5 veerust, kus on määratud unikaalne identifikaator, sündmuse kood, sündmuse kirjeldus, kommentaar ning süsteemi kood, millise süsteemi alla see käib. Töötajate tabelis on 20 veergu, enamus veergudest on täidetud andmetega, mis pärinevad samalt leheküljelt, mis klientide andmed ning on lisatud esinduse nimi ning töönimetus.

Rakenduse logi andmed saadakse kuulates TCP (*Transmission Control Protocol*) porti, kuhu tulevad Syslogi RFC 5424 formaadis logiandmed. Logis on näha ajatempel, süsteemi nimi, saatja masina nimi ja sõnum. Sõnumis on määratud töötaja isikukood, kliendi isikukood, sündmuse kood, lehekülje URL (*Uniform resource locator*) ning HTTP (*Hypertext Transfer Protocol*) vastuse kood. Logiandmed on suvaliselt genereeritud autori poolt. Kokku on 15790474 sündmust, mis on jaotatud kümne päeva peale.

2.4.2 Lõpptabeli andmed

Lõpptabelid kasutavad täheskeemat, mis koosneb ühes faktitabelist ning neljast dimensioonitablelist. Dimensioonideks on klientide dimensioon, töötajate dimensioon, sündmuste dimensioon ja süsteemide dimensioon. Faktitabeli sisu on näha tabelist 3.

Tabel 1: Faktitabeli sisu

Veerg	**Alati täidetud**	**Kommentaar**	**Allikas**
id	Jah	Unikaalne identifikaator	Genereeritud
calendar_date	Jah	Sündmuse kuupäev	Rakenduse logi ts veerg
datetime	Jah	Sündmuse kuupäev ja kellaaeg	Rakenduse logi ts veerg
event_id	Jah	Sündmuse identifikaator	Rakenduse logist väärtus x, kus x on user_id_code=”x”
system_id	Jah	Süsteemi identifikaator	
actor_code	Jah	Töötaja isikukood	Rakenduse logist väärtus x, kus x on user_id_code=”x”

customer_code	Ei	Kasutaja isikukood	Rakenduse logist väärtus x, kus x on customer_id_code =”x”
phone_number	Ei	Kasutaja telefoninumber	Clients tabelist veerg telephonenumber.
description	Ei	Sündmuse kirjeldus	„Külastas lehte: ” || Rakenduse logist lehekülje URL

2.4.3 Protsessid

Iga lahendusega tehakse 4 toimingut.

- Logiandmete HDFS'i toomine.
- Andmebaasist klientide andmete laadimine HDFS'i.
- Hive's olevate dimensioonide tabelite täitmine.
- Hive's oleva faktitabeli täitmine.

2.5 Kasutatavad tehnoloogiad

Selles töö osas antakse lühiülevaade töös kasutatavatest tehnoloogiatest.

2.5.1 Apache Flume

„Apache Flume on hajus, töökindel ja rakendatav teenus, millega saab tõhusalt koguda, koondada ja liigutada suurtes kogustes logiandmeid erinevatest allikatest tsentraliseeritud andmehoidlasse [15] ”.

Flume koosneb agentidest, millel on kolm komponenti: source, channel ja sink. Source tegeleb sündmuste vastuvõtmisega erinevatel viisidel ning saadab need ühte või mitmesse channelisse. Flume on näiteks võimeline võtma vastu Avro[1] sündmuseid või sündmuseid Kafkast või Twitterist [5] .

[1] Avro on andmete serialiseerimise süsteem[28] .

Channelis hoitakse sündmuseid seni, kuni sink need sealt eemaldab. Sink eemaldab channelist sündmuseid ning viib need edasi järgmisele agendile, nagu näiteks Avro sink või siis sündmuse lõppsihtkohta, näiteks HDFS sink [5] .

2.5.2 Apache Sqoop

Sqoop võimaldab importida ja eksportida andmeid struktureeritud andmebaasidest, nagu näiteks relatsioonilisest andmebaasist Hadoopi failisüsteemi. Sqoopi töid on võimalik Oozie's ajastada [6] .

2.5.3 Apache Hive

Apache Hive on Hadoopi peal olev andmelao tarkvara, mis võimaldab lugeda, kirjutada ja hallata andmeid HDFS'is. Seda kõike saab teha standardse SQL keeles [7] .

Hive kasutab *schema on readi*, mis tähendab seda, et andmete struktuur defineeritakse nende lugemisel. Traditsioonlistes relatsioonilistes andmebaasides kasutatakse *schema on write*'i ehk siis andmete struktuur määratakse siis, kui neid andmeallikasse kirjutatakse. *Schema on readi* eelisteks on suurem paindlikkus struktuuri defineerimisel, võimalus alles jätta originaalandmed ning võimalus laadida andmed oma süsteemi, teadmata, mida sa nendega tegema hakkad. *Schema on write*'i eelisteks on selle efektiivsus ning teadmine, et su andmed on korrektsed [8] .

Hive's on olemas kahte erinevat tüüpi tabeleid: Hive'i poolt hallatud (managed) tabelid ja välised (external) tabelid. Vaikimisi teeb Hive managed tabeleid, mille puhul on failid, metaandmed ja statistika hallatud Hive enda sisemiste protsesside poolt. Managed tabelite puhul kaovad kõik failid ja metaandmed, kui see tabel kustutada [9] .

Välise tabeli puhul peab tabeli tegemisel määrama andmete asukoha ja metaandmed (näiteks skeemi) ning välise tabeli kustutamise puhul jäävad kõik failid alles. Välist tabelit kasutatakse näiteks siis, kui andmed tulevad kuskilt teisest süsteemist, näiteks Sqoopiga laaditud andmed, ning nende andmete peale tahetakse päringuid teha [9] .

2.5.4 Apache Oozie

Apache Oozie võimaldab teha töövoogusid ehk Oozie Workflowsi, mis koosnevad erinevatest Hadoopi töödest, mida nimetatakse *node*'deks, nagu näiteks Sqoopi importimisest ja Hive'i päringu käivitamisest. Oozie töövood on suunatud atsüklilised graafid (Directed Acyclical Graphs ehk DAG). Oozie koordinaator ehk Oozie Coordinator käivitab Oozie Workflowsi mingi teatud ajavahemiku tagant või andmete olemasolu korral [10] .

2.5.5 Apache Falcon

Apache Falconit kasutatakse siin töös Oozie Workflow'de ajastamiseks. Falcon genereerib ise Oozie Coordinatori ning käivitab selle [11] .

2.5.6 Apache Nifi

Apache Nifi automatiseerib andmete liikumist erinevate andmeallikate ja süsteemide vahel, tehes andmete sissetoomise lihtsaks, kiireks ja turvaliseks [12] . Nifi koosneb erinevatest protsessoritest, mis teevad kokku ühtse andmevoo. Igal protsessoril on oma ülesanne, näiteks failide HDFS'i panemine või Avro faili JSON failiks tegemine.

Andmete ühest protsessorist teise saatmiseks kasutatakse FlowFile'i. FlowFile'l on atribuudid, mis on *map key/value* paaridest ning null või enam baidiline sisu. Protsessorid on omavahel seotud Connectionitega, mis käituvad nagu järjekorrad. Connection'itel saab määrata ka prioriteete ning limiite, palju sündmusi võib järjekorras olla [12] .

Protsessorid ja nendevahelised Connectionid saab jagada ka Process Groupideks. Need võimaldavad ka andmeid sisse võtta ning välja saata. Niimoodi on võimalik Process Groupidest teha uued taaskasutatavad komponendid, mis on moodustatud juba olemasolevatest komponentidest [12] .

3 Praegune lahendus kasutades Apache Falconit ja Apache Ooziet

Praeguse lahenduse juures on kasutuses Apache Falcon ja Apache Oozie. Oozie võimaldab teha töövoogusid Hadoopi töödest. Näiteks käivitada Hive'i päringu ja seejärel Sqoopi töö.

3.1 Protsessid

Andmete sissetoomiseks kasutatakse praeguses lahenduses Sqoopi ja Flume'i. Flume võtab vastu TCP pordile tulevat Syslogi formaadis logiandmeid ning salvestab need HDFS'i ning Sqoopiga imporditakse MySQL andmebaasist andmeid HDFS'i. Juba Hadoopi failisüsteemis olevaid andmeid töödeldakse ja hallatakse Apache Hive'ga. Töövoogude haldamiseks kasutatakse Apache Ooziet ja Apache Falconit.

3.1.1 Andmete sissetoomine

Nagu mainitud, kasutatakse andmete sissetoomiseks Flume'i ja Sqoopi. Need kaks komponenti ongi Hadoopis mõeldud andmete sissetoomiseks.

Sqoopiga andmete sissetoomiseks on vaja teha Sqoopi Job, mida saab teha serveri käsurealt, kuhu Sqoop paigaldatud on. Kuna meil on klastris mitu arvutit ning me ei tea, millisest arvutist Oozie selle Jobi käivitab, siis salvestame selle Sqoopi *metastore*'i. See võimaldab meil selle jobi ükskõik millisest serverist käivitada, millel on ühendus Sqoopi metastore'iga.

Tabel 2: Sqoopi job'i parameetrite seletus [20]

Võti	**Tähendus**
--create	Käsk, mida antud job'ile teha, millele järgneb jobi nimi, millele käsk rakendub.

--meta-connect	Sqoopi metastore'i JDBC aadress
import	Mis sorti job teha. Siin võib olla veel –delete, --show, --exec
--connect	Andmebaasi JDBC (*Java Database Connectivity*) URL, kust andmeid laadima hakatakse
--table	Andmebaasi tabeli nimi, kust andmeid laaditakse
--delete-target-dir	Määrab, et enne uute andmete kirjutamist kustutatakse ette antud –target-dir'i kaust ära
-m	Määrab ära, mitu mitu paralleelset ühendust andmebaasi tehakse.
--as-avrodatafile	Määrab, et andmed salvestatakse Avro formaadis
--target-dir	HDFS'i kaust, kuhu andmed maha kirjutatakse

Tabelis 2 kirjeldatud, mida tähendavad parameetrid, mida Sqoopi jobi tegemisel kasutame. Sqoopi jobi enda definitsiooni on võimalik vaadata jooniselt 2.

```
sqoop job
-Dmapreduce.job.user.classpath.first=true
--create load-clients
--meta-connect "jdbc:hsqldb:hsql://kaarel-
lap.local:16000/sqoop"
--
import
--connect "jdbc:mysql://kaarel-
desk:3306/users?user=sqoop&password=sqoop"
--table clients
--delete-target-dir
--target-dir '/ds_falcon/users/clients'
-m 1
--as-avrodatafile
```

Joonis 2: Sqoopi jobi defineerimine

Flume'i puhul on vaja luua agent. Agendi konfiguratsiooni näeb Lisa 1 all. Kuna ridahaaval selle konfiguratsiooni kommenteerimine oleks liiga pikk, seetõttu kommenteerin seda loogiliste osade kaupa..

```
a1.sources = r1
a1.channels = c1
a1.sinks = k1
```

Joonis 3: Flume'i agendile source'i, channel'i ja sinki määramine

„a1” on agendi nimi ning siin määran ära agendi Source'd, Channelid ja Sinkid. „r1” on agendi Source, „c1” on agendi Channel ning „k1” on agendi Sink (vt joonis 3).

```
a1.sources.r1.type = syslogtcp
a1.sources.r1.port = 5140
a1.sources.r1.host = localhost
a1.sources.r1.keepFields = true
a1.sources.r1.channels = c1
a1.sources.r1.interceptors = i1
```

Joonis 4: Flume'i Source'i konfiguratsioon

Agendi Source'i konfiguratsioon. Määran ära Source'i tüübi, milleks on „syslogtcp” ning „hosti” ja „pordi”, kust andmeid kuulata. „keepFields” võti määrab ära, kas jätta alles kogu saadud info või ainult sõnumiosa kogu logist [15] . Järgmine rida näitab, et

sellest Source'st saadetakse andmed edasi „c1” Channelisse. See Source kasutab interceptorit „i1”. (vt joonis 4)

```
a1.sources.r1.interceptors.i1.type =
regex_extractor
a1.sources.r1.interceptors.i1.regex =
\\S+\\s(\\d{4})\\-(\\d{2})\\-(\\d{2})
a1.sources.r1.interceptors.i1.serializers = s1
s2 s3
a1.sources.r1.interceptors.i1.serializers.s1.n
ame = year
a1.sources.r1.interceptors.i1.serializers.s2.n
ame = month
a1.sources.r1.interceptors.i1.serializers.s3.n
ame = day
```

Joonis 5: Flume'i Source'le Interceptori määramine

Interceptor „i1” eesmärgiks on igast vastuvõetud sõnumist regulaaravaldisi kasutades välja sõeluda päev, kuu ja aasta, mis lisatakse sõnumi päisesse (vt joonis 5). Need kolm väärtust on igal sõnumil Syslogi headeris olemas. Väärtusi kasutatakse Sinkis, et määrata, millisesse HDFS'i kataloogi sündmus salvestatakse.

```
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /flume-
dirs/test-channel/checkpoint
a1.channels.c1.dataDirs = /flume-dirs/test-
channel/data
```

Joonis 6: Flume'i Channeli konfiguratsioon

Kasutame *File Channelit*, mis kirjutab vastuvõetud sündmused serveri kettale ning Sink saab need sealt kätte [15] . File Channeli defineerimine on võrdlemisi triviaalne, määrata tuleb kataloogid, kuhu andmed serveri kettal salvestatakse (vt joonis 6).

```
a1.sinks.k1.type = hdfs
a1.sinks.k1.channel = c1
a1.sinks.k1.hdfs.path = /datasource/flume-
test/%{year}/%{month}/%{day}
a1.sinks.k1.hdfs.filePrefix = events
a1.sinks.k1.hdfs.rollInterval = 360
a1.sinks.k1.hdfs.rollSize = 0
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.batchSize = 10000
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.useLocalTimeStamp = true
```

Joonis 7: Flume'i Sinki konfiguratsioon

Lõpuks defineerima Sinki. Sinki tüübiks on „hdfs”, mis tähendab, et sellesse Sinki jõudvad andmed salvestatakse HDFS'i [15] . Andmed saadakse Channelist „c1”. Nagu näha, siis pathis kasutame eelnevalt Interceptoris „i1” leitud muutujaid „year”, „month” ja „day”. „rollInterval” määrab sekundites ära intervalli, kui tihti faile HDFS'i kirjutatakse [15] . „rollSize” määrab, faili suuruse, mille saavutamise puhul see HDFS'i kirjutatakse ja „rollCount” sündmuste arvu, mille saavutamisel kirjutatakse fail HDFS'i [15] . Kui need väärtused on 0, siis neid eiratakse ehk siis selle konfiguratsiooniga kogutakse 360 sekundit sündmusi ja siis kirjutatakse need HDFS'i ühe failina [15] . „fileType” määrab ära, kas andmeid HDFS'i kirjutamisel pakitakse kokku või mitte; „DataStream” tähendab seda, et ei pakita [15] . (vt joonis 7)

3.1.2 Andmete töötlemine

Andmete töötluseks kasutatakse Hive'i. Kokku on kolm erinevat andmete sisestamise päringut: faktitabeli täitmine, sündmuste ja süsteemide dimensiooni täitmine. Kõik Hive'i päringud kutsutakse välja Oozie töövoogudes, millest tuleb juttu järgmises alajaotuses.

3.1.3 Protsesside ajastamine

Protsessid ajastatakse kasutades Apache Falconit. Falconis on määratud ära töövoo käivitamise vahemik, kui tihti seda seal vahemikus käivitatakse ja millist töövoogu käivitatakse.

Kokku on kaks Falconi protsessi: „load-and-transfer-dimensions” ja „transfer-fact”. Esimene neist käivitab Oozie töövoo, kus on Sqoopi job, mis laadib andmebaasist klientide info ning Hive'i päringud dimensioonide laadimiseks ehk siis kõik vajaliku, et dimensioonid saaks laaditud. „transfer-fact” protsess laadib andmed faktitabelisse. Hive'i jaoks vajalike parameetrite „year”, „month”, „day” käib Falconi XML'is (*Extensible Markup Language*), selleks võetakse Oozie parameeter „nominalTime” ning lahutatakse üks päev maha, et eelmise peäva andmed faktitabelisse laadida. „nominalTime” parameetri väärtus on see kuupäev, millal Oozie protsess esialgselt ajastatud on ehk see ei muutu ning seda saab kasutada ka suvalisel päeval just selle

päeva protsessi uuesti laadimiseks [16] . Protsesside XML failid leiab Lisa 1 ja Lisa 2 jaotuste alt.

3.2 Lahenduse hindamine

Siin peatükis hinnatakse lahendusi eelmainitud aspektide alusel.

3.2.1 Realisatsiooni keerukus

Põhiline keerukus selle lahenduse juures oli see, et nii Falconi, kui ka Oozie protsessid on defineeritud XML failidena. Apache Falconil on küll olemas ka veebiliides, mis teeb Falconi protsessi tegemise mõnevõrra lihtsamaks, kuid laias laastus on see lihtsalt graafiline XML'i täitmine, kus on vajalikud elemendid ette antud.

Oozie'l seevastu graafiline kasutajaliides töövoogude tegemiseks puudub. See tähendab, et kogu DAG'ist peab olema ettekujutus peas olemas ning pikemate töövoogude juures, kus võib olla hargnemis- ja otsustussõlmesid, muutub see keeruliseks.

3.2.2 Vea otsimise keerukus

Arendamise käigus tuleb vahepeal ka kindlasti vigu ning kui nende põhjuste ülesleidmine on lihtne, teeb see arendamisega kergemaks. Siin peatükis keskendume põhiliselt Oozie'le, sest kõik töövooga seotud vead läbi Oozie veebiliidese. Oozie kasutajaliides kuvab informatsiooni töövoogude ja koordinaatorite kohta. Sama informatsiooni saab kätte ka kasutades käsurida, kuid siinkohal on veebiliides mugavam.

Kasutajaliides tundub esmasel kasutusel üpriski segane. Et jõuda Oozie veebiliidese pealehelt mõne protsessi logideni tuleb teha minimaalselt kaheksa hiirevajutust. Positiivne pool selle juures on see, et pärast kaheksat hiirevajutust suunab Oozie otse Yarni logidesse, kus on kõige detailsem informatsioon selle kohta, mis valesti läks.

3.2.3 Jõudlus

Jõudluse osas võrdleme võimekust logikirjeid vastu võtta ehk siis Flume'i võimekust. Selleks saadame *netcati* nimelise programmiga andmeid porti, mida Flume kuulab. Tulemust mõõdame käsurealt käsuga, mis kuulab porti 5140 ning loeb kuulatud sõnumite ridade arvu. Porti kuulatakse 10 sekundit.

```
timeout 10s sudo tcpdump -i any -A port 5140 |
awk '{printf "\r%lu", NR}'
```

Katset korrati viis korda ning tulemused on näidatud tabelis 3.

Tabel 3: Flume'i jõudlus

Katse number	Ridade arv 10 sekundi jooksul
1	3787
2	3541
3	3219
4	3803
5	3201

3.2.4 Funktsionaalsus

Positiivne asi selle lahenduse juures on see, et kuna kõik protsessid defineeritakse XML'idena ja käivitatakse käsurealt, siis saab maksimaalselt ära kasutada kõigi komponentide funktsionaalsust. Kasutajaliidestega pole piiratud tegevusi, mida teha saab.

Tabel 4: Oozie ja Falconi funktsionaalsus

Funktsionaalsus	Olemasolu
Logiandmete sissetoomine	Jah
Andmebaaside tabeli terviklik sissetoomine	Jah
Andmebaaside tabeli inkrementaalne sissetoomine	Jah
Võimalus ETL protsess kindla päeva kohta uuesti käivitada	Jah
Võimalus anda Hive'i päringule parameetreid	Jah

4 Lahendus kasutades Talendi

Esimene alternatiivne lahendus olemasolevale on Talendi poolt pakutavad Big Data lahendused. Siin töös kasutatakse Talendi tasuta tarkvara Open Studio for Big Data, kuid olemas on ka ettevõtetele mõeldud tasuline Talend Big Data Integration, kus on lisatud ettevõtetele vajalikku funktsionaalsust, näiteks tehniline tugi ning paremad administreerimise ja monitoorimise vahendid [13] . Siin töös pole lisafunktsionaalsust vaja ning seetõttu valiti tasuta versioon.

Talend on töölaua rakendus, mille saab käivitada ka Hadoopi klastrist väljaspool, kui käivitatavalt arvutil on võimalus luua ühendus Hadoopi klastriga. Talendi üks eeliseid on see, et kõik protsessid saab valmis teha ja ajastada ühest rakendusest. Talend võimaldab „drag and drop" põhimõttel luua töövoogusid ETL protsesside jaoks. Kui töövoog on valmis, siis genereerib Talend sellest käivitatava Java JAR faili.

4.1 Protsessid

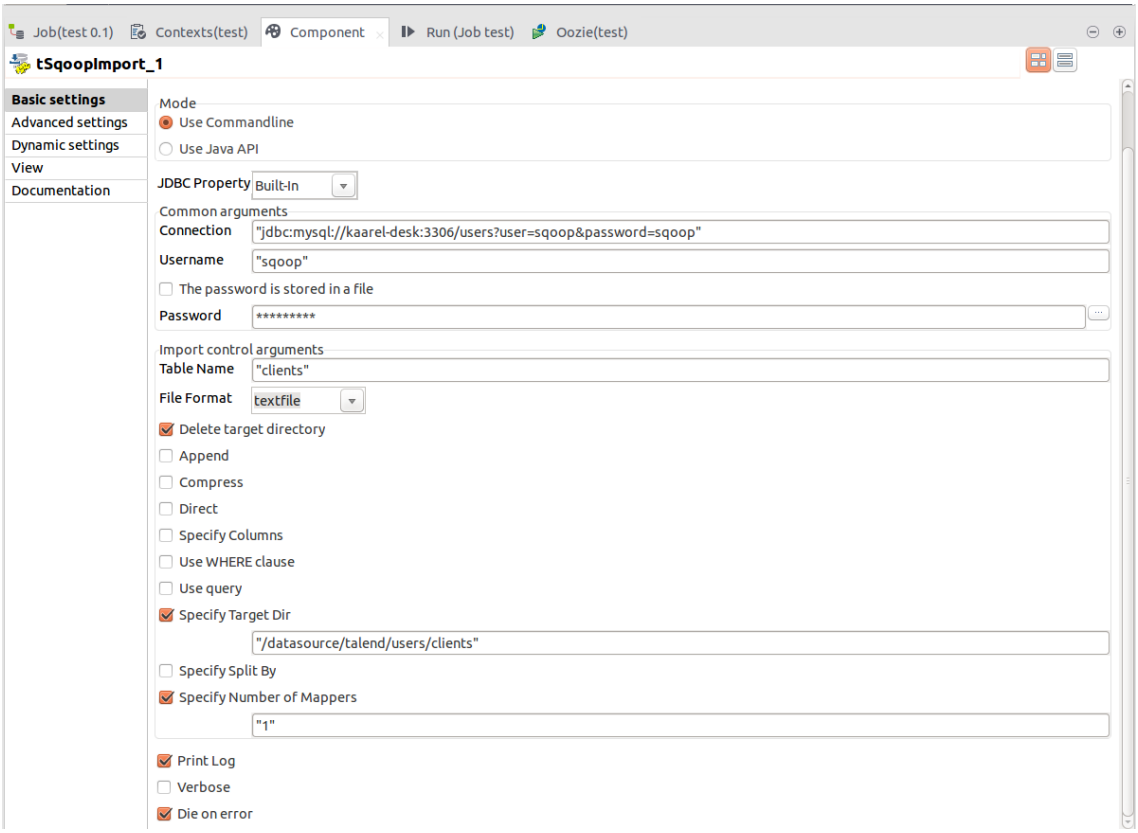
Siin peatükis kirjeldatakse, kuidas protsessid on realiseeritud kasutades Talendi Open Studiot.

4.1.1 Andmete sissetoomine

Kuna Talendi Open Studios pole eraldi logiandmete sissetoomise jaoks komponenti, siis kasutame ka selle lahenduse juures Flume'i, mille kirjelduse leiab peatüki 3.1.1 alt.

Andmebaasidest andmete toomiseks kasutab Talend Sqoopi, kuid erinevus eelmise lahendusega on see, et Talendis on tehtud kasutajaliides, mis annab kõik Sqoopi valikud ette, mis teeb arendaja elu kergemaks (vt joonis 8). Talend võtab sisestatud andmed ning genereerib ise nende põhjal käsu ning käivitab selle kas API (*Application Programming Interface*) kaudu või käsurealt. Kui valitud on käsurida, siis tähendab see, et samas arvutis, kus Talend käivitati peab olema ka paigaldatud Sqoop [4] .

Talend genereerib „sqoop import” käsu ja käivitab selle, mitte ei tekita Sqoopi job'i, nagu eelmise lahenduse juures tehti. See teeb parameetrite muutmise kergemaks, sest Sqoopi job'i parameetrit muutes peab kõige pealt vana job'i ära kustutama ja siis uue tegema. Sellega kaasneb ka oluline piirang. Nimelt on Sqoopis võimalik teha ka inkrementaalne import, mis tähendab seda, et iga kord laaditakse ainult need andmed, mida eelmine kord ei laaditud ning seda kas kasvava unikaalse identifikaatori või kellaaja ja kuupäeva järgi [20] . Kui on tehtud Sqoopi job, siis pärast igat sellist laadimist salvestatakse viimane identifikaatori või kuupäeva väärtus Sqoopi metastore'i ning järgmise laadimise ajal võetakse ainult sellest suuremaid (kuupäeva puhul suurem või võrdne) väärtusi [27] . Kuna Talend käivitab „sqoop import” käskluse, siis pole Sqoopil viimast laaditud väärtust kuhugi salvestada ning iga kord laaditakse alla terve tabel.



Joonis 8: Sqoopi seadistamine Talend Open Studios

4.1.2 Andmete töötlemine

Andmete töötlus käib läbi Apache Hive'i. Talendis on selle jaoks komponent tHiveLoad, mis võtab sisendiks parameetrid, mille alusel genereerib sisestamislause.

4.1.3 Protsesside ajastamine

Protsesside ajastamine käib Talendis läbi Oozie. Erinevalt eelmisest lahendusest ei ole igale Talendis olevale sõlmele eraldi Oozie sõlme, vaid Talend kompileerib töövoost JAR faili ning laseb Ooziel selle käivitada.

4.2 Lahenduse hindamine

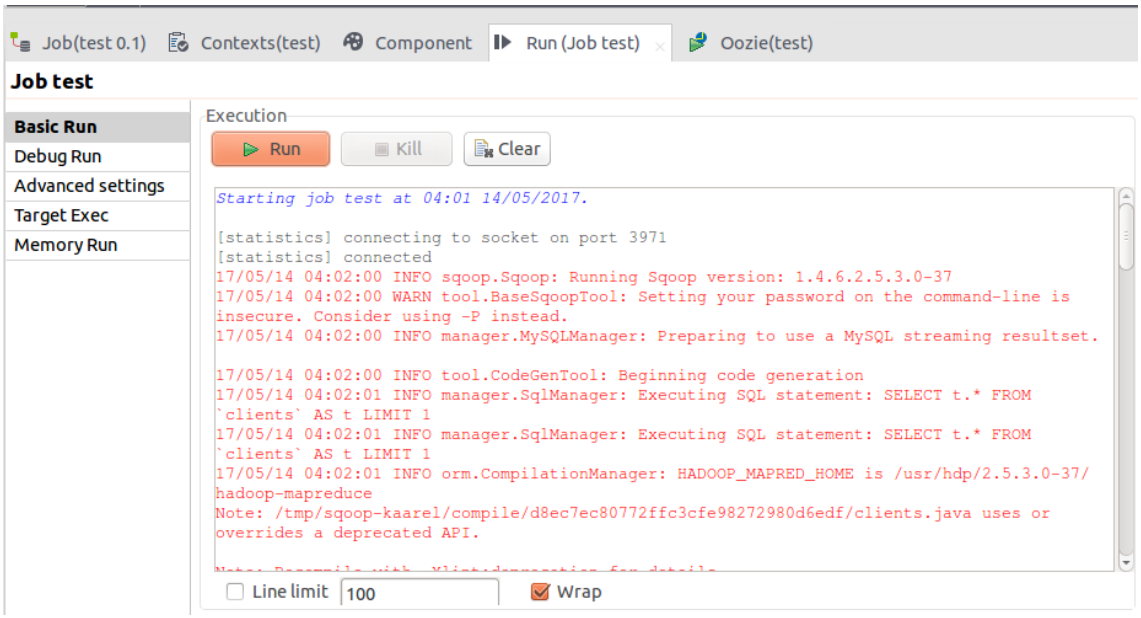
Siin peatükis hinnatakse lahendus nelja erineva kriteeriumi järgi: realisatsiooni keerukuse, vea otsimise keerukuse, jõudluse ja funktsionaalsuse.

4.2.1 Realisatsiooni keerukus

Lahenduse ülesseadmine oli üllatavalt kerge. Kõik vajalikud Hadoopi parameetrid, millega Talend Hadoopi klastri külge ühendab, leiab Talend ise üles.

4.2.2 Vea otsimise keerukus

Kuna Oozie käivitab ainult ühe sõlme, kus käivitatakse JAR fail, siis logide leidmine muutub mõnevõrra kergemaks, kuid logidest arusaamine jällegi raskemaks. Põhjus on selles, et eelmise lahenduse juures oli igal töövoo osal Oozies eraldi sõlm ning igal sõlmel on eraldi logid, kuid nagu enne mainitud, siis Talendi puhul käivitab Oozie ainult ühe JARi. See tähendab et kõik töövoo logid asuvad ühes kohas ning need pole kuidagi eraldatud, seetõttu võib logidest arusaamine raskeks muutuda.



Joonis 9: Näide, kuidas Talend logisid kuvab

Arendamise käigus on logide leidmine see-eest kerge. Töövoo käivitamisel kuvatakse kohe samas aknas ka kõik selle töövooga seotud logid, nende hulgas ka see, mis valesti läks (vt joonis 9).

4.2.3 Jõudlus

Jõudluse osas selle lahenduse juures midagi uut testida ei saanud, nii et kehtib kõik sama, mis 3.2.3 punktis.

4.2.4 Funktsionaalsus

Nagu peatükis 4.1.1 mainitud, siis kõige suurem funktsionaalsuse puudus selle lahenduse juures on Sqoopiga inkrementaalse laadimise tegemise puudus. Tegemist on väga suure puudusega, sest kui laadida andmeid relatsioonilisest andmebaasist, kus võib olla sadu miljoneid ridu, siis ei ole kindlasti hea lahendus laadida neid iga päeva uuesti.

Üks lahendus sellele probleemile oleks teha ise Talendile komponent, mis käivitaks Sqoopi job'i. Sellisel juhul tuleks teha ka komponent, mis teeks valmis Sqoopi job'i. See nõuaks aga nädalatepikkust arendustööd.

Hive'i puhul olid kõik vajalikud parameetrid olemas. Seda nii tabeli tegemise juures, kui ka andmete sisestamise puhul. Tabeli tegemise puhul sai väga detailidesse minna ning sai isegi määrata enda tehtud SerDe'sid[1].

Üks puudus võrreldes eelmise lahendusega on see, et puudub võimalus protsessi kindla päeva kohta uuesti jooksutada. Protsess jookseb küll Oozie's ning on olemas „nominalTime" väärtus, aga selle edasisaatmine JAR'ile ei tulnud välja. Ka dokumentatsioonis polnud mainitud, et Oozie'le saaks lisaparameetreid kaasa anda ning neid töövoo sees kasutada, seetõttu käivitatakse iga protsess praeguse kuupäevaga ning kui tahta protsess varasema kuupäevaga uuesti jooksutada, siis tuleb see protsessi sisse kirjutada.

Tabel 5: Talendi funktsionaalsus

Funktsionaalsus	Olemasolu
Logiandmete sissetoomine	Jah
Andmebaaside tabeli terviklik sissetoomine	Jah
Andmebaaside tabeli inkrementaalne sissetoomine	Ei
Võimalus ETL protsess kindla päeva kohta uuesti käivitada	Ei
Võimalus anda Hive'i päringule parameetreid	Jah

[1] SerDe on lühend *Serializer/Deserializer*. SerDe võimaldab Hive'l lugeda HDFS'ist ja kirjutada HDFS'i faile SerDe poolt määratud formaadis.

5 Lahendus kasutades Apache NiFit

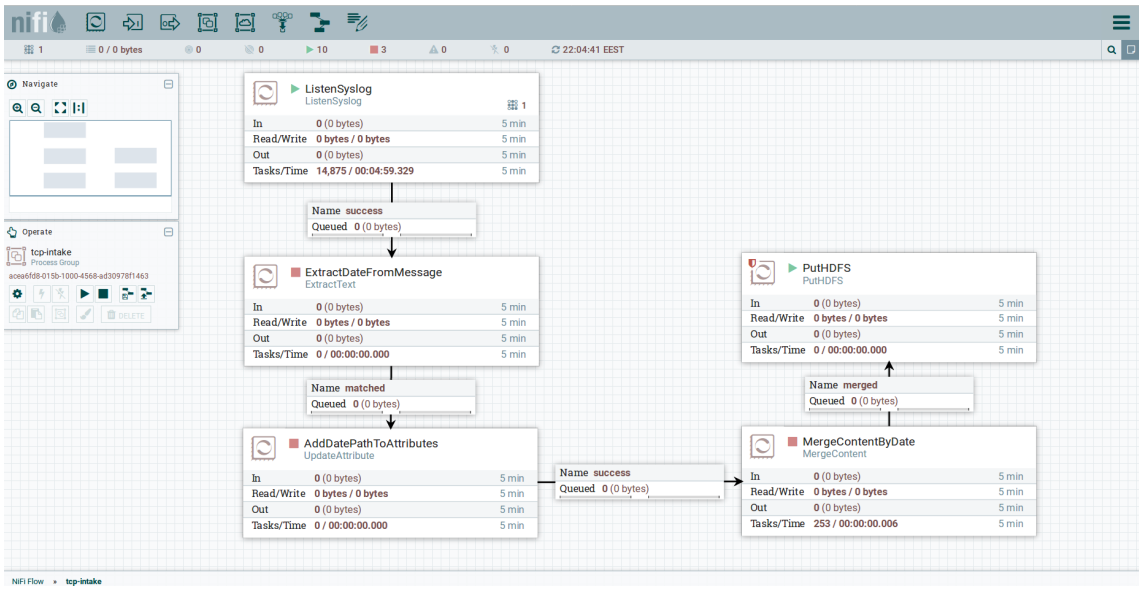
Apache Nifi erineb eelnevatest lahendustest märgatavalt. Ainukene sama komponent, mida ka eelmiste lahenduste puhul kasutati on Hive, ülejäänud komponendid on realiseeritud Nifi enda protsessoritega. Kuigi Nifi põhiline eesmärk pole realiseerida ETL protsessi, saab ta sellega hakkama.

5.1 Protsessid

Võrreldes eelmiste lahendustega on Nifi puhul iga protsess esitletud erineva andmevoo protsessiga. See tähendab, et näiteks iga Hive'i sisestamise voog käivitub eraldi ning üksteisest sõltumata.

5.1.1 Andmete sissetoomine

Andmete sissetoomise jaoks on kaks voogu, üks andmebaasi andmete jaoks ning teine logiandmete jaoks. Vaatleme kõigepealt logiandmete voogu (vt joonis 10).

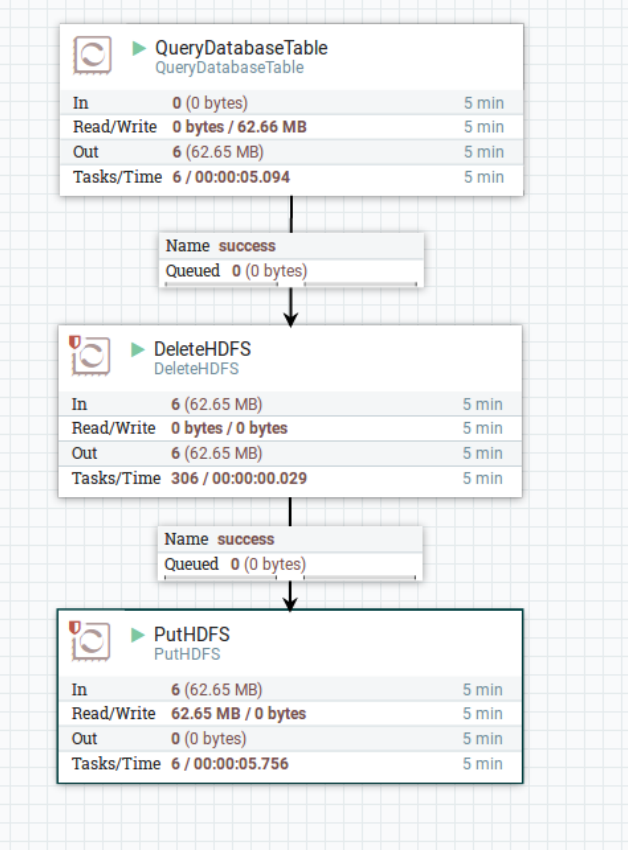


Joonis 10: Nifi voog logiandmete sissetoomiseks

Esimene protsessor on ListenSyslog, mis kuulab porti 5141 ning võtab vastu Syslogi formaadis andmeid. Samuti lisab see protsessor edasi saadetavale FlowFile'ile Syslogi atribuute nagu näiteks prioriteedi, ajatempli, saatja, sõnumi jms. Kõik sõnumid, mis on Syslog formaadis saadetakse edasi ExtractDateFromMessage protsessorisse [14] .

ExtractDateFromMessage protsessor leiab regulaaravaldistega sõnumi seest üles kuupäeva. Aasta, kuu ja päev leitakse kõik eraldi ning lisatakse FlowFile'i atribuutideks. Kui kõik 3 atribuuti on leitud, saadetakse FlowFile edasi AddDatePathToAttributes protsessorisse, mis ühendab need kolm atribuuti selliselt, et lõpuks on need formaadis „aasta/kuu/päev” ning lisab need atribuudina „concated_date”.

Järgmisena jõuab FlowFile MergeContentByDate protsessorisse, mis ühendab kokku erinevad FlowFile'd kasutaja poolt defineeritud atribuudi alusel, selles kontekstis „concated_date” alusel. Pärast ühendamist saadetakse FlowFile'id PutHDFS'i, mis kirjutab FlowFile'ide sisu HDFS'i.



Joonis 11: Nifi voog andmebaasi andmete sissetoomiseks

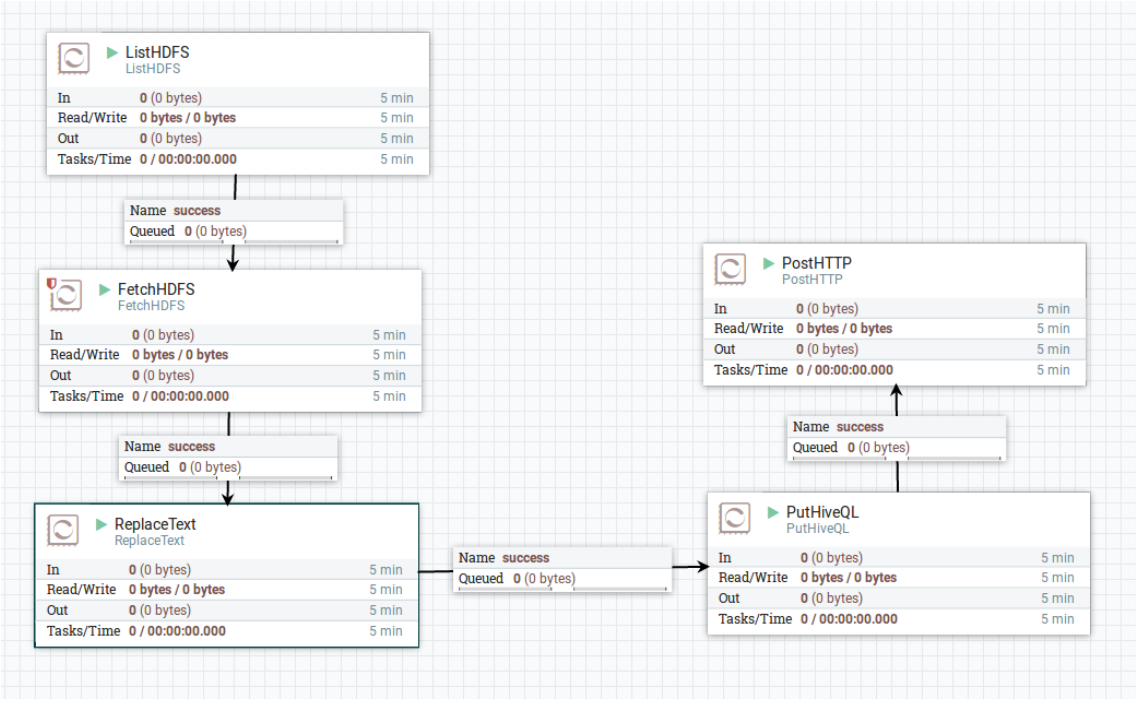
Andmebaaside jaoks tehtud voog on oma olemuselt kergem (vt joonis 11). See koosneb ainult kolmest protsessorist. Esimene protsessor QueryDatabaseTable teeb päringu välisesse andmebaasi ning salvestab saadud andmed FlowFile'i. Õnnestumise korral saadetakse andmed DeleteHDFS protsessorisse, mis FlowFile'i endaga midagi ei tee, vaid suunab selle edasi. DeleteHDFS kustutab HDFS'is faili või kataloogi, mis talle parameetrina ette antakse. DeleteHDFS saadab failid edasi PutHDFS'i, mis kirjutab FlowFile'i sisu HDFS'i.

5.1.2 Andmete töötlemine

Andmete töötlemiseks mõeldud voog on jällegi mõnevõrra keerulisem (vt joonis 12). Esiteks on kaks protsessorit ListHDFS ja FetchHDFS. Neid on vaja selleks, et kätte saada HQL (*Hive Query Language*) fail, mis on salvestatud HDFS'i. ListHDFS annab edasi FetchHDFS'ile õiges formaadis HQL faili asukoha ning FetchHDFS saab faili HDFS'ist kätte ning paneb selle FlowFile'i.

Kuna HQL failis on parameetrid ${year}, ${month} ja ${day}, siis tuleb need välja vahetada voo tööle minemise aja kuupäevaks. Seda teeb protsessor ReplaceText. ReplaceText leiab regulaaravaldisi kasutades üles HQL failist teksti „${year}-${month}-${day}” ning vahetab selle Nifi väljenduskeelt kasutades selle hetkel olevaks kuupäevaks.

Edasi liigub FlowFile PutHiveQL protsessorini, mis käivitab Hive'i DDL (*Data definition language*) või DML (*Data manipulation language*) päringu. Päringu võtab ta FlowFile'i seest, selle pärast võtsimegi HDFS'ist HQL faili, et see kergesti FlowFile'i sisse saada. Edasi liigub voog PostHTTP protsessorini. Kui ListHDFS kuvab nimekirja failidest, siis ta salvestab maha ka kõige uuema faili kuupäeva ning järgmine kord otsitakse ainult faile, mis on sellest uuemad. Seda nimetatakse Nifis oleku salvestamiseks. Kuna me tahame saada HDFS'ist faili kätte olenemata selle tekkimisajast, siis kasutame PostHTTP, et eemaldada ListHDFS'i salvestatud kõige uuema faili aeg ehk oleku.



Joonis 12: Nifi töövoog Hive'i päringute tegemiseks ja nende HDFS'i salvestamiseks

5.1.3 Protsesside ajastamine

Nifis käib protsesside ajastamine iga protsessori kaupa eraldi. On olemas kolme erinvat tüüpi ajastamist: aja järgi ajastamine, CRON väljendi[1] järgi ajastamine ja sündmuste järgi ajastamine. Aja järgi ajastamine tähendab seda, et iga kindla aja tagant käivitatakse protsessor. CRON'i järgi käivitatakse protsess vastavalt sellele, milline CRON'i väljend määratakse. Sündmuste järgi ajastamine tähendab seda, et protsessor käivitatakse iga kord, kui protsessorisse jõuab FlowFile, aga kuna see on eksperimentaalne valik, siis siin töös seda ei kasutata.

Logiandmete sissetoomise jaoks kasutatakse aja järgi ajastamist, kuid ajaperioodiks on määratud 0 sekundit. See tähendab, et protsessor käivitub nii tihti kui võimalik, kui on olemas andmed, mida töödelda ehk logiandmeid võetakse vastu kogu aeg, kui andmeid tuleb. Andmebaasi andmete sissetoomise ning andmete töötlemise esimesed protsessorid on ajastatud kasutades CRON'i ning ülejäänud on aja järgi ajastatud, kus iga käivituse vaheks on määratud 0 sekundit ehk siis need käivituvad, kui andmed eelmisest protsessorist on kätte saadud.

[1] CRON'i väljend on sõne, mis koosneb viiest või kuuest tühikuga eraldatud väljast, mis määrab ära ajad, millal käsk käivitatakse [24] .

5.2 Lahenduse hindamine

Siin peatükis hinnatakse lahendus nelja erineva kriteeriumi järgi: realisatsiooni keerukuse, vea otsimise keerukuse, jõudluse ja funktsioonalsuse.

5.2.1 Realisatsiooni keerukus

Selle lahenduse juures pidi ülesannetele natukene teise nurga alt lähenema; kui eelmiste lahenduste juures tegi üks komponent enamasti ühte asja ning selle algusest kuni lõpuni, siis Nifiga tuleb teha mitmest protsessorist koosneb voog, mis lahendab ühe ülesande. Näiteks Sqoop laadib andmed välisest andmebaasist HDFS'i ning Oozie's andes Hive'i sõlmele ette HQL faili asukoha, teeb ta failis defineeritud päringu.

Kuid pärast Nifi tööpõhimõtetest rohkem aru saades tundus voogude tegemine loogiline ning tekkis arusaam, et selline loogika suurendab võimalusi, mida Nifiga teha saab.

5.2.2 Vea otsimise keerukus

Logide vaatamine ei ole Nifis eriti mugav, sest kõik logid salvestatakse ühte faili. See tähendab, et kui midagi valesti läks, siis tuleb läbi käia kõikide protsessorite logid ning leida üles see protsessor, kus viga toimus. Potentsiaalne lahendus sellele probleemile on teha eraldi voog Nifis, mis sorteerib logid erinevatesse failidesse või kasutada Linuxi käsureal rakendust grep[1], millega ainult ühe protsessori read välja filtreerida.

Protsessori vea korral kuvab Nifi veebiliides ka veateate. Pahatihti juhtub, et veateade on pikem, kui ala, kus veateadet kuvatakse, nii et osa sõnumist pole näha. Siis tulebki minna Nifi logifaili kallale, et see veateade täies pikkuses kätte saada.

5.2.3 Jõudlus

Jõudluse osas võrreldakse võimekust logikirjeid vastu võtta. Selleks saadame *netcati* nimelise programmiga andmeid porti, mida Nifi protsessor ListenSyslog kuulab. Tulemust mõõdame käsurealt käsuga, mis kuulab porti 5141 ning loeb kuulatud sõnumite ridade arvu. Porti kuulatakse 10 sekundit. Tulemusi näeb tabelist 6

[1] Grep on käsurea vahend, mis tagastab sisendist saadud failist kõik read, mis vastavad ette antud regulaaravaldisele [25] .

Tabel 6: Nifi jõudlus

Katse number	Ridade arv 10 sekundi jooksul
1	49400
2	52274
3	53238
4	51567
5	52049

5.2.4 Funktsionaalsus

Nifis on 188 erinevat protsessorit ning selle töö puhul olid kõik vajalikud protsessorid olemas, isegi võttes arvesse, et Nifi pole loodud ETL protsesside jaoks. Kuid kuna Nifis ei käivita protsessi spetsiifiliselt ühegi päeva kohta eraldi, siis pole võimalik ka nende uuesti jooksutamine. Ehk et kui tahta varasema päeva kohta protsess uuesti jooksutada, tuleb olemasolevat voogu muuta või teha uus voog kõrvale.

Tabel 7: Nifi funktsionaalsus

Funktsionaalsus	Olemasolu
Logiandmete sissetoomine	Jah
Andmebaaside tabeli terviklik sissetoomine	Jah
Andmebaaside tabeli inkrementaalne sissetoomine	Jah
Võimalus ETL protsess kindla päeva kohta uuesti käivitada	Ei
Võimalus anda Hive'i päringule parameetreid	Ei

6 Tulemused

Tulemuste hindamiseks kasutame AHP tehnikat. Kõigepealt võrdleme kõiki nelja hindamise aluseks võetud kriteeriumi omavahel [21] . Võrdlusi teeme skaalal ühest kuni seitsmeni.

Tabel 8: Kaalude seletused

Kaal	**Kommentaar**
1	Kaks kriteeriumit on sama tähtsad
3	Üks kriteerium on teisest mõnevõrra tähtsam
5	Üks kriteerium on teisest palju tähtsam
7	Üks kriteerium on teisest väga palju tähtsam

Nüüd võrdleme kõiki nelja kriteeriumit omavahel ning anname kaalud, kui palju on üks teisest tähtsam [21] . Kaalude seletused leiab tabelis 8.

Tabel 9: Kriteeriumite võrdlused

Kriteerium		**Kumb on tähtsam**	**Kaal**
A	**B**		
Realisatsiooni keerukus	Vea otsimise keerukus	A	5
Realisatsiooni keerukus	Jõudlus	A	1
Realisatsiooni keerukus	Funktsionaalsus	B	3
Vea otsimise keerukus	Jõudlus	B	5
Vea otsimise keerukus	Funktsionaalsus	B	7
Jõudlus	Funktsionaalsus	B	3

Kaaluda määramine on üsnagi subjektiivne ülesanne, nii et siin töös lähtus autor omast kogemusest, millised kriteeriumid on tähtsamad, kui teised.

Edasi lisame kaalud maatrikisisse. Maatriksi peadiagonaalist ülespoole veerud täidame tabeli põhjal ning maatriksi alumise poole täidame ülemise poole põhjal, võttes ülemise poole väärtuse vastandväärtuse [21] . Kui on tabeli põhjal on kriteerium A tähtsam, siis lisame maatriksisse kaalu väärtuse ning kui tähtsam on kriteerium B, siis kaalu vastandväärtuse (vt tabel 10) [21] .

Tabel 10: Kriteeriumite maatriks

Kriteerium	Realisatsiooni keerukus	Vea otsimise keerukus	Jõudlus	Funktsionaalsus
Realisatsiooni keerukus	1	5	1	1/3
Vea otsimise keerukus	1/5	1	1/5	1/7
Jõudlus	1	5	1	1/3
Funktsionaalsus	3	7	3	1

Nüüd arvutame omavektori. Selleks liidame iga maatriksi veeru kokku ning jagame maatriksi väärtuse läbi selle veeru summaga (vt tabel 11) [21] .

Tabel 11: Kriteeriumi maatriksi omavektori leidmine

Kriteerium	Realisatsiooni keerukus	Vea otsimise keerukus	Jõudlus	Funktsionaalsus
Realisatsiooni keerukus	0,1923	0,2778	0,1923	0,1842
Vea otsimise keerukus	0,0385	0,0555	0,0385	0,0789
Jõudlus	0,1923	0,2778	0,1923	0,1842
Funktsionaalsus	0,5769	0,3889	0,5769	0,5526

Sellest maatrikist omavektori leidmiseks liidame kõik read kokku ning jagame kriteeriumite arvuga.

$$\frac{1}{3}\begin{vmatrix} 0{,}1923+0{,}2778+0{,}1923+0{,}1842 \\ 0{,}0385+0{,}0556+0{,}0385+0{,}0789 \\ 0{,}1923+0{,}2778+0{,}1923+0{,}1842 \\ 0{,}5769+0{,}3889+0{,}5769+0{,}5526 \end{vmatrix} = \begin{vmatrix} 0{,}2117 \\ 0{,}0529 \\ 0{,}2117 \\ 0{,}5238 \end{vmatrix}$$

Pärast paari kaupa võrdlemist tuli välja, et kõige tähtsam on lahenduse funktsionaalsus ning teist ja kolmandat kohta jagasid jõudlus ja realisatsiooni keerukus. Kõige vähem tähtsam oli vea otsimise keerukus.

Nüüd, kui omavektor kriteeriumitele leitud on, tuleb leida omavektor ka igale kriteeriumile eraldi, kus võrdleme lahendusi.

Esimene kriteerium, mida võrdleme on realisatsiooni keerukus. Mida kergem on kasutatavatest tehnoloogiatest mitteteadjal inimesel protsess üles seada, seda kõrgema väärtuse saab lahendus. Kõige keerukamale lahendusele paneme kaaluks 1 ning selleks on lahendus, mis kasutas Falconit ja Oozie't. Põhjus selleks on graafilise kasutajaliidese puudumine terve protsessi tegemiseks. Nifi saab väärtuseks 4, graafiline kasutajaliides on olemas ning töötab hästi, kuid kogu töövoo tegemine ei ole nii intuitiivne, kui olla võiks. Kõige kergem oli lahendusest realiseerida Talendis ning see saab väärtuseks 6. Graafiline liides oli olemas ja töötas nii nagu ette nähtud ning erinevalt Nifist olid olemas komponendid, mis lahendasid probleeme terviklikult.

$$R = \begin{vmatrix} 1 & 1{,}5 & 6 \\ 0{,}6667 & 1 & 4 \\ 0{,}1667 & 0{,}25 & 1 \end{vmatrix} \rightarrow \begin{vmatrix} 0{,}5454 \\ 0{,}3636 \\ 0{,}0909 \end{vmatrix}$$

Järgmisena võrdleme vea otsimise keerukust.

Jõudluse all hindame logiandmete sissetoomise võimsust. Kuna kolme lahenduse peale kasutati ainult kahte erinevat moodust, kuidas logiandmeid vastu võeti, siis Talendi ja Falconi tulemused on samad. Testides oli näha, kui palju võimsam Nifi oli andmete vastuvõtmisel, nii et Nifi saab maksimumväärtuse 7 ning Flume, mida kasutasid teised süsteemid saab väärtuse 1.

$$J = \begin{vmatrix} 1 & 0{,}1429 & 1 \\ 7 & 1 & 7 \\ 1 & 0{,}1429 & 1 \end{vmatrix} \rightarrow \begin{vmatrix} 0{,}1111 \\ 0{,}7777 \\ 0{,}1111 \end{vmatrix}$$

Viimasena võtame vaatluse alla funktsionaalsuse. Funktsionaalsuse arvutamisel sai iga lahendus alguses 10 punkti ning seejärel lahutati puuduolevate funktsionaalsuste punkte. Kui funktsionaalsus puudus, siis lahutati 2 punkti, aga kui oli võimalus teha mingi erilahend, siis liideti 1 punkt juurde. Lõppkokkuvõttes jäi seis selline, et Falcon sai maksimaalsed 10 punkti. Talendil puudusid kaks funktsiooni: andmebaaside tabeli inkrementaalne sissetoomine ja võimalus ETL protsess kindla päeva kohta uuesti käivitada. Inkrementaalse laadimise kohta on võimalik teha erilahendus, tehes Talendi uue komponendi, kuid kindla päeva kohta uuesti laadimist ei tundunud olevat võimalik teostada ka erilahendusega. Niisiis Talend saab väärtuseks 7. Nifil puudusid samuti kaks funktsiooni: võimalus ETL protsess kindla päeva kohta uuesti käivitada ja võimalus anda Hive'i päringule parameetreid. Kindla päeva kohta uuesti käivitamine pole Nifi arhitektuuri juures võimalik. Hive'ile parameetrite etteandmine on võimalik, kui teha ise Nifile protsessor. Seega saab ka Nifi 7 punkti

$$F=\left|\begin{array}{ccc|c} 1 & 1{,}4286 & 10 & 0{,}4167 \\ 0{,}7 & 1 & 7 & \rightarrow 0{,}2917 \\ 2{,}4 & 3{,}4286 & 7 & 0{,}2917 \end{array}\right|$$

Nüüd, kui kõikide kriteeriumite kohta on omavektor leitud, saame selle kokku panna üheks suureks maatriksiks ning seejärel see korrutada alguses leitud kriteeriumi maatriksiga [21] . Tulemuseks saame pingerea lahendustest.

$$L=\begin{vmatrix} 0{,}5454 & 0{,}5556 & 0{,}1111 & 0{,}2917 \\ 0{,}3636 & 0{,}1111 & 0{,}7778 & 0{,}2917 \\ 0{,}0909 & 0{,}3334 & 0{,}1111 & 0{,}4167 \end{vmatrix} * \begin{vmatrix} 0{,}2117 \\ 0{,}0529 \\ 0{,}2117 \\ 0{,}5238 \end{vmatrix} = \begin{vmatrix} 0{,}3211 \\ 0{,}4002 \\ 0{,}2786 \end{vmatrix}$$

Kõige parema tulemuse sai Nifi 0,4002'ga, järgmisena tuli Talend 0,3211 ning viimasena Falcon 0,2786. Kuid tuleks pöörata tähelepanu, et Nifi oli parim ainult jõudluse arvestus ja Talendi lahendus oli parim nii realisatsiooni keerukuses kui ka vea otsimise keerukuses, siis tasuks kaaluda Talendi kasutamist. Eriti juhul, kui sissevõetavate logiandmete hulk ei ole väga suur.

7 Kokkuvõte

Antud töö eesmärgiks oli võrrelda kolme ETL protsesside tegemise vahendit Hadoopi raamistikul. Aluseks võeti juba realiseeritud lahendus telekommunkatsioonieттevõttele ning pakuti välja kaks alternatiivset lahendust. Olemasolev süsteem on realiseeritud kasutades Apache Falconit ning alternatiivseteks lahendusteks pakuti välja Apache Nifi ning Talendi Open Studio.

Kõik kolm lahendust realiseeriti näiteülesande põhjal. Näiteülesanne koostati selliselt, et see oleks võimalikult sarnane olemasoleva süsteemile, mis loodi telekommunikatsiooni-ettevõtte jaoks. Pärast lahenduste realiseerimist hinnati kõiki kolme lahendust nelja kriteeriumi alusel: realisatsiooni keerukus, vea otsimise keerukus, jõudlus ja funktsionaalsus. Lõpuks võrreldi kõiki kolme lahendust omavahel kasutades AHP meetodit ning leiti parim lahendus.

Arvestades kõiki kriteeriumi tuli parimaks lahenduseks Apache Nifi. Nifile järgnes Talendi Open Studio ning viimasele kohale jäi praegune lahendus, mis kasutab Apache Falconit. Töös jõuti järeldusele, et Nifi osutus sobivaimaks lahenduseks eelkõige tänu oma suurepärasele logiandmete sissetoomise jõudlusele. Kui sissetulevate logiandmete maht ei ole väga suur, siis tasuks kindlasti mõelda Talendi Open Studio kasutamist, sest see oli nii realisatsiooni keerukuse kui ka vea otsimise keerukuse aspektides parem, kui Nifi.

Eesmärkide täituvuse võib lugeda positiivseks. Töös väljapakutud kaks alternatiivset lahendust osutusid mõlemad paremaks, kui olemasolev lahendus. Üks töö võimalikke edasiarendusi oleks hinnata, kui ressursimahukas oleks praeguse lahenduse pealt üleminek kummagi alternatiivse lahenduse peale.

Kasutatud kirjandus

[1] Saarm, U. Veebiroomaja rakendus vastavalt meediaagentuuri vajadustele: bakalaureusetöö. Tallinna Ülikool, Tallinn, 2015.

[2] Rouse, M. Hadoop. [WWW] http://searchcloudcomputing.techtarget.com/definition/Hadoop (20.05.2017)

[3] Four Key Pillars To A Big Data Management Solution. [WWW] http://info.talend.com/rs/talend/images/WP_EN_BD_Talend_4Pillars_BigDataManagement.pdf (20.05.2017)

[4] Talend Open Studio for Big Data Components Reference Guide. [WWW] https://help.talend.com/#/reader/hm5FaPiiOP31nUYHph0JwQ/4bqjn07LZs8TfEeoPOLfDQ (20.05.2017)

[5] Prabhakar, A. Apache Flume – Architecture of Flume NG. [WWW] http://blog.cloudera.com/blog/2011/12/apache-flume-architecture-of-flume-ng-2/ (20.05.2017)

[6] Apache Sqoop – Overview. [WWW] https://blogs.apache.org/sqoop/entry/apache_sqoop_overview (20.05.2017)

[7] Apache Hive. [WWW] https://cwiki.apache.org/confluence/display/Hive/Home (20.05.2017)

[8] Hive enforces schema during read time? [WWW] http://stackoverflow.com/a/11764519 (20.05.2017)

[9] Managed and External Tables. [WWW] https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ManagedandExternalTables (20.05.2017)

[10] Oozie Workflow Overview. [WWW] https://oozie.apache.org/docs/4.3.0/DG_Overview.html (20.05.2017)

[11] Apache Falcon. [WWW] https://hortonworks.com/apache/falcon/ (20.05.2017)

[12] Apache NiFi. [WWW] https://hortonworks.com/apache/nifi/ (20.05.2017)

[13] Why Upgrade? [WWW] https://www.talend.com/products/why-upgrade (20.05.2017)

[14] ListenSyslog [WWW] https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.2.0/org.apache.nifi.processors.standard.ListenSyslog/index.html (20.05.2017)

[15] Flume 1.7.0 User Guide. [WWW] https://flume.apache.org/FlumeUserGuide.html (20.05.2017)

[16] Oozie Coordinator Specification. [WWW] https://oozie.apache.org/docs/4.3.0/CoordinatorFunctionalSpec.html (20.05.2017)

[17] Syslog. [WWW] https://en.wikipedia.org/wiki/Syslog (20.05.2017)

[18] White, T. Hadoop: The Definitive Guide. 3rd ed. Sebastopol : O'Reilly Media / Yahoo Press, 2012

[19] Rakendusliides. [WWW] https://et.wikipedia.org/wiki/Rakendusliides (20.05.2017)

[20] Sqoop import, Syntax. [WWW] https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#_syntax (20.05.2017)

[21] Analytic Hierarchy Process (What is AHP). [WWW] http://web.cjcu.edu.tw/~lcc/Courses/TUTORIAL/AHP%20Tutorial.doc (20.05.2017)

[22] Data manipulation language. [WWW] https://en.wikipedia.org/wiki/Data_manipulation_language (21.05.2017)

[23] Data definition language. [WWW] https://en.wikipedia.org/wiki/Data_definition_language (21.05.2017)

[24] CRON expression. [WWW] https://en.wikipedia.org/wiki/Cron#CRON_expression (21.05.2017)

[25] grep. [WWW] https://en.wikipedia.org/wiki/Grep (21.05.2017)

[26] Extract, transform, load. [WWW] https://en.wikipedia.org/wiki/Extract,_transform,_load (21.05.2017)

[27] Sqoop job. [WWW] https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#_purpose_6 (21.05.2017)

[28] Apache Avro™ 1.8.1 Documentation. [WWW] https://avro.apache.org/docs/current/ (21.05.2017)

[29] JSON. [WWW] https://et.wikipedia.org/wiki/JSON (21.05.2017)

[30] Comma-separated values. [WWW] https://en.wikipedia.org/wiki/Comma-separated_values (21.05.2017)

Lisa 1 – Flume'i konfiguratsioon

```
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = syslogtcp
a1.sources.r1.port = 5140
a1.sources.r1.host = localhost
a1.sources.r1.keepFields = true
a1.sources.r1.channels = c1
a1.sources.r1.interceptors = i1

a1.sources.r1.interceptors.i1.type = regex_extractor
a1.sources.r1.interceptors.i1.regex = \\S+\\s(\\d{4})\\-(\\d{2})\\
(\\d{2})
a1.sources.r1.interceptors.i1.serializers = s1 s2 s3
a1.sources.r1.interceptors.i1.serializers.s1.name = year
a1.sources.r1.interceptors.i1.serializers.s2.name = month
a1.sources.r1.interceptors.i1.serializers.s3.name = day

a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /flume-dirs/test-channel/checkpoint
a1.channels.c1.dataDirs = /flume-dirs/test-channel/data

a1.sinks.k1.type = hdfs
a1.sinks.k1.channel = c1
a1.sinks.k1.hdfs.path = /datasource/flume-test/%{year}/%{month}/%
{day}
a1.sinks.k1.hdfs.filePrefix = events
a1.sinks.k1.hdfs.rollInterval = 360
a1.sinks.k1.hdfs.rollSize = 0
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.batchSize = 10000
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.useLocalTimeStamp = true
```

Lisa 2 – Falconi protsess dimensioonide täitmiseks

```
<process name="load-and-transfer-dimensions"
xmlns="uri:falcon:process:0.1">
    <clusters>
        <cluster name="FalconCluster">
            <validity start="2017-05-07T00:05Z" end="2099-11-
15T01:05Z"/>
        </cluster>
    </clusters>

    <parallel>1</parallel>
    <order>LIFO</order>
    <frequency>minutes(360)</frequency>
    <timezone>UTC</timezone>

    <workflow engine="oozie" path="/workflows/oozie-load-and-transfer-
dimensions.xml"/>
</process>
```

Lisa 3 – Falconi protsess faktitabeli laadimiseks

```
<process name="transfer-fact" xmlns="uri:falcon:process:0.1">
    <clusters>
        <cluster name="FalconCluster">
            <validity start="2017-04-02T00:05Z" end="2099-11-
15T01:05Z"/>
        </cluster>
    </clusters>

    <parallel>1</parallel>
    <order>LIFO</order>
    <frequency>days(1)</frequency>
    <timezone>UTC</timezone>

    <properties>
        <property name="ydayDay" value="$
{coord:formatTime(coord:dateOffset(coord:nominalTime(), -1, 'DAY'),
'dd')}" />
        <property name="ydayMonth" value="$
{coord:formatTime(coord:dateOffset(coord:nominalTime(), -1, 'DAY'),
'MM')}" />
        <property name="ydayYear" value="$
{coord:formatTime(coord:dateOffset(coord:nominalTime(), -1, 'DAY'),
'yyyy')}" />
        <property name="table_name" value="flumetest" />
        <property name="schema" value="falcon." />
    </properties>

    <workflow engine="oozie" path="/workflows/oozie-transfer-
fact.xml"/>
</process>
```