TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JONNE PIHLANEN

SUOSITTELIJAJÄRJESTELMÄN RAKENTAMINEN APACHE SPARKILLA

Diplomityö

Tarkastaja: Timo Aaltonen
Tarkastaja ja aihe hyväksytty
tiedekuntaneuvoksen kokouksessa
2. Toukokuuta 2018

ABSTRACT

JONNE PIHLANEN: Building a Recommendation Engine with Apache Spark
Tampere University of Technology
Master of Science, 47 pages
November 2018
Master's Degree Program in Signal Processing
Major: Data Engineering
Examiner: Timo Aaltonen
Keywords: Apache Spark, Recommendation, MovieLens, Scala, AWS, EMR, S3

The amount of recommendation engines around the Internet is constantly growing. Even the most common everyday tasks, such as reading the news, incorporate an existence of a recommender system. A recommendation engine is a system that tries to form an opinion about user's preferences and recommend items of use. This thesis studies the usage of Apache Spark when building a recommendation engine with Scala programming language and Amazon Web Services (AWS).

TIIVISTELMÄ

JONNE PIHLANEN:Suosittelijajärjestelmän rakentaminen Apache Sparkilla
Tampereen teknillinen yliopisto
Diplomityä, 47 sivua
lokakuu 2018
Signaalinkäsittelyn koulutusohjelma
Pääaine: Data Engineering
Tarkastaja: Timo Aaltonen
Avainsanat: Apache Spark, Recommendation, MovieLens, Scala, AWS, EMR, S3

Suosittelujärjestelmien määrä Internetissä on kasvanut jatkuvasti. Jopa kaikista arkipäiväisimmät toimet, kuten uutisien lukeminen, sisällyttävät suosittelujärjestelmän. Suosittelujärjestelmä on järjestelmä, joka yrittää muodostaa käyttäjän mieltymyksistä mielipiteeen ja suositella hyödyllisiä asioita. Tässä työssä tutustutaan Apache Sparkiin, Scala-ohjelmointikieleen sekä Amazon Web Services (AWS) palveluun ja rakennetaan suosittelujärjestelmä näiden teknologioiden avulla.

ALKUSANAT

Vihdoinkin loppumetreillä.

Tämän työn tekemiseen kului noin 900 päivää, työpaikan vaihto, epätoivoa, laiskuutta ja lopuksi vielä kaaos AWS:n laskun kanssa, mutta onneksi myös muutamia oivalluksen hetkiä.

Erityiskiitos tuesta vaimolleni Nooralle. Kiitos myös kaikille muille ketkä jaksoivat muistuttaa, hoputtaa ja välillä hymähdelläkin tämän työn tekemiselle ja tekemisestä, ilman tuota hoputusta olisin varmaankin täyttämässä jo toista jatkoaikahakemustani!

Tampere, 18.11.2018

Jonne Pihlanen

SISÄLLYS

LYHENTEET JA MERKINNÄT

ALS	Alternating Least Squares
Apache Spark	Sovelluskehys hajautettujen ohjelmien rakentamiseen
AWS	Amazon Web Services, pilvipalvelualusta
S3 Bucket	S3:n tarjoama abstraktio, eräänlainen kansio
DAG	Directed Acyclic Graph
DataFrame	Sparkin tarjoama tietorakenne
Dataset	Sparkin tarjoama tietorakenne
EMR	Elastic Map Reduce
Eksplisiittinen	Suoraan, selvästi ilmaistu
Free Tier	AWS:n ilmainen kokeilujakso
Implisiittinen	Epäsuorasti, epäselvästi ilmaistu
JAR	Koottu kirjasto, tiedostoformaatti
JVM	Virtuaalikone, joka suorittaa JAR-tiedostoja
Latentti tekijä	Piilevä tekijä, vaikea tai mahdoton havainnoida
MapReduce	Hajautetun laskennan malli, Sparkin edeltäjä
MLlib	Apache Sparkin tarjoama koneoppimis-kirjasto
MSE	Mean Square Error
RDD	Sparkin tarjoama tietorakenne
REPL	komentorivityökalu
RMSE	Root Mean Square Error
S3	Simple Storage Service
Scala	Ohjelmointikieli
Sparse	Harva
SSH	Secure Shell, järjestelmä turvalliseen tiedonsiirtoon
ZIP	Tiedostoformaatti, joka yhdistää monta tiedostoa yhdeksi

1. JOHDANTO

Suosittelujärjestelmät ovat nykyisin jatkuvasti läsnä jokapäiväisessä elämässämme. Ne auttavat päätöksenteossa verkko-ostoksissa, suoratoistopalveluissa, sosiaalisessa mediassa tai yksinkertaisesti uutisten lukemisessa. Yksinkertaisin ja luonnollisin suosittelun muoto on ihmiseltä ihmiselle suosittelu. Ihmiset voivat kuitenkin tehokkaasti suositella vain niitä asioita, jotka ovat itse henkilökohtaisesti kokeneet. Tällöin suosittelijajärjestelmistä tulee hyödyllisiä, sillä ne voivat mahdollisesti tarjota suosituksia tuhansista tai jopa miljoonista erilaisista tuotteista.

Suosittelu voidaan jakaa kahteen pääkategoriaan: tuotepohjaiseen ja käyttäjäpohjaiseen. [2] Tuotepohjaisessa suosittelussa tarkoituksena on etsiä samankaltaisia tuotteita, sillä käyttäjän ajatellaan olevan mahdollisesti kiinnostunut samankaltaisista tuotteista myös tulevaisuudessa. Käyttäjäpohjaisessa suosittelussa käyttäjän ajatellaan olevan kiinnostunut tuotteista, joita samankaltaiset käyttäjät ovat ostaneet, joten siinä on tarkoituksena etsiä samankaltaisia käyttäjiä, jotta voidaan suositella näiden ostamia tuotteita.

Tämän työn päämääränä on tutustua Apache Spark-sovelluskehykseen sekä Scala-ohjelmointikieleen ja toteuttaa suosittelujärjestelmä näiden teknologioiden avulla. Toteutus tehdään toimimaan pilvipalvelussa, joten työssä tutustutaan myös kevyesti kahteen pilvipalvelujätti Amazonin AWS:n (Amazon Web Services) tarjoamaan palveluun: EMR (Elastic Map Reduce) sekä S3 (Simple Storage Service). EMR on hallittu klusterialusta, joka yksinkertaistaa big data -sovelluskehysten, kuten Apache Sparkin, käyttämistä AWS:n palveluissa. [3] S3 on tietovarasto, joka on suunniteltu helpottamaan pilvilaskentaa ja se tarjoaa yksinkertaisen rajapinnan tietovaraston hallintaan. [4]

Apache Spark on sovelluskehys, joka mahdollistaa hajautettujen ohjelmien rakentamisen. [13] Hajautetussa ohjelmassa suoritus voidaan jakaa useiden käsittelysolmujen kesken. Jotkin suositteluongelmat voidaan mallintaa hajautettuna ohjelmana, jossa kaksi matriisia, käyttäjät ja tuotteet, prosessoidaan iteratiivisella algoritmilla,

joka mahdollistaa ohjelman suorittamisen rinnakkain. [13]

Apache Spark on rakennettu Scala-ohjelmointikielellä. [13] Scala on monikäyttöinen, moniparadigmainen ohjelmointikieli, joka tarjoaa tuen funktionaaliselle ohjelmoinnille sekä vahvan tyypityksen. Työn käytännön osuus on toteutettu Scalaa käyttäen, joten lyhyt johdanto ohjelmointikieleen tarjotaan lukijalle.

Tämä työ on rakentuu seuraavista osista. Luvuissa kaksi ja kolme esitetään työn kannalta oleellinen teoriaosuus. Luvussa neljä keskustellaan Apache Sparkista, avoimen lähdekoodin järjestelmästä, joka mahdollistaa hajautettujen ohjelmien rakentamisen. Luku viisi esittää toteutuksen suosittelijajärjestelmälle. Luvussa kuusi käydään läpi tulokset. Lopuksi luvussa seitsemän esitellään johtopäätökset.

2. TEORIA

Seuraava kappale kuvailee yleisellä tasolla matriisin tekijöihinjakoa, vuorottelevien pienimpien neliöiden (Alternating Least Squares, ALS) -algoritmia sekä Scala -ohjelmointikieltä. Työssä tarkasteltava Spark-sovelluskehys sisältää toteutuksen ALS-algoritmille, joka on matriisin tekijöihinjako-algoritmi. [21] Spark on kirjoitettu Scala-ohjelmointikielellä ja tästä syystä se on myös suositeltu kieli Spark-ohjelmien kirjoittamiseen. Tämän työn koodiesimerkit on kirjoitettu käyttäen Scala:n syntaksia, joten lukijalle tarjotaan lyhyt johdanto Scalaan.

2.1 Matriisin tekijöihinjako

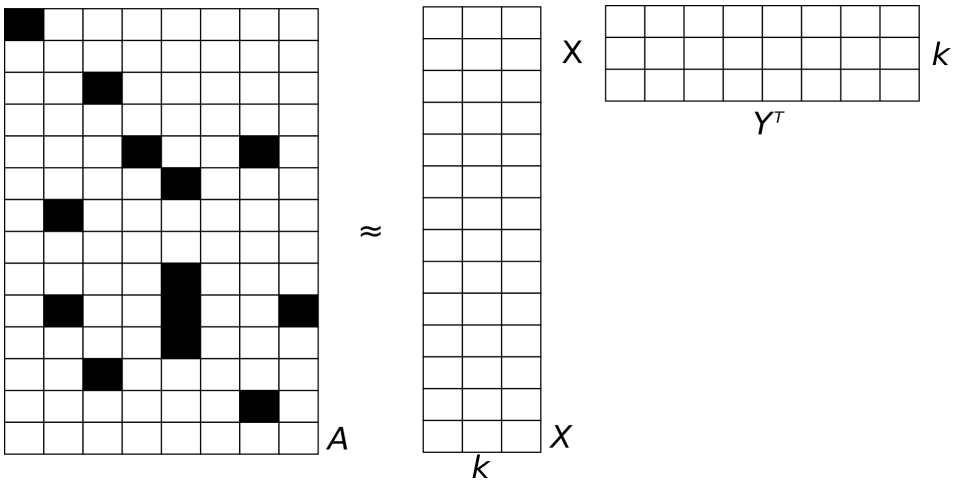
Matriisin tekijöihinjaossa matriisi hajoitetaan pienempien matriisien tuloksi. Matriisin tekijöihinjako kuuluu suureen algoritmien luokkaan nimeltä latenttien tekijöiden mallit (Latent-factor models). Latenttien tekijöiden mallit yrittävät selittää usean käyttäjän ja tuotteen välillä havaittuja vuorovaikutuksia käyttämällä suhteellisen pientä määrää piileviä, latentteja tekijöitä. Voidaan esimerkiksi yrittää selittää miksi ihminen ostaisi tietyn albumin lukemattomien mahdolisuuksien joukosta kuvailemalla käyttäjiä ja tuotteita mieltymysten perusteella, joista ei ole mahdollista saada tietoa. [21] Latenttia tekijää ei ole mahdollista tarkastella sellaisenaan. Esimerkiksi ihmisen terveys on latentti tekijä, sillä sitä ei ole mahdollista mitata kuten esimerkiksi verenpainetta.

Matriisin tekijöihinjako-algoritmit käsittelevät käyttäjä- ja tuotetietoja suurena matriisina A. Jokainen rivissä i sekä sarakkeessa j sijaitseva alkio esittää arvostelua, jonka käyttäjä on antanut tietylle tuotteelle. [21] Yleensä A on harva (sparse), jolla tarkoitetaan että useimmat A:n alkiot sisältävät arvon nolla. Tämä johtuu siitä, että kaikista mahdollisuuksista usein vain muutama käyttäjä-tuote-kombinaatio on olemassa. [21]

Matriisin tekijöihinjako mallintaa A:n kahden pienemmän matriisin X ja Y tulona,

jotka ovat varsin pieniä. Koska A:ssa on monta riviä ja saraketta, X ja Y sisältävät paljon rivejä, mutta vain muutaman (k) sarakkeen. Nämä k-saraketta vastaavat latentteja tekijöitä, joita käytetään kuvailemaan datassa sijaitsevia vuorovaikutuksia. Hajotelma (factorization) on ainoastaan arvio, sillä k on pieni. [21] Alla olevassa kuvassa 2.1 on esitetty hahmotelma matriisin tekijöihinjaosta.

Kuva 2.1 *Matriisin tekijöihinjako (muokattu lähteestä [21])*



Tavanomainen lähestymistapa matriisin tekijöihinjakoon perustuvassa yhteisösuodatuksessa on kohdella käyttäjä-tuote -matriisin alkioita käyttäjien antamina eksplisiittisinä arvosteluina. Eksplisiittistä tietoa on esimerkiksi käyttäjän antama arvio tuotteelle. Spark ALS kykenee käsittelemään sekä implisiittistä että eksplisiittistä tietoa. Implisiittistä tietoa on esimerkiksi sivujen katselukerrat tai tieto siitä, onko käyttäjä kuunnellut tiettyä artistia. [24] [21]

Usein monissa tosielämän käyttötapauksissa on käytettävissä ainoastaan implisiittistä tietoa, kuten katselukerrat, klikkaukset, ostokset, tykkäykset tai jakamiset. Spark MLlib kohtelee tietoa lukuina, jotka esittävät havaintojen vahvuutta, kuten klikkausten määrä tai kumulatiivinen aika, joka käytetään elokuvan katseluun. MLlib ei siis yritä mallintaa arviomatriisia suoraan. Ekplisiittisten arvioiden sijaan, nämä luvut liittyvät havaittujen käyttäjämieltymysten varmuuteen. Tämän tiedon perusteella malli koettaa etsiä latentteja tekijöitä, joiden avulla voidaan ennustaa käyttäjän arvio tuotteelle. [24]

Näihin algoritmeihin viitataan joskus matriisin täyttö (matrix completion) algoritmeina. Tämä johtuu siitä, että alkuperäinen matriisi A saattaa olla harva vaikka matriisitulo XY^T on tiheä. Vaikka tulosmatriisi sisältää arvon kaikille alkioille, se

on kuitenkin vain likiarvo A:sta. [21]

2.1.1 Alternating Least Squares (ALS)

Yhteisösuodatusta käytetään usein suosittelijajärjestelmissä. Nämä tekniikat pyrkivät täyttämään käyttäjä-tuote-assosiaatiomatriisin puuttuvat kohdat. Spark MLlib tukee mallipohjaista yhteisösuodatusta, jossa käyttäjiä ja tuotteita kuvaillaan pienellä määrällä latentteja tekijöitä, joita voidaan käyttää puuttuvien kohtien ennustamiseen. Spark MLlib käyttää *vuorottelevien pienimpien neliöiden* (Alternating Least Squares, ALS) -algoritmia näiden latenttien tekijöiden oppimiseen. [24]

Spark ALS yrittää arvata arvostelumatriisin A kahden alemman dimension matriisin, X ja Y, tulona. [23]

$$A = XY^T \tag{2.1}$$

Tyypillisesti näihin arvioihin viitataan tekijämatriiseina. Perinteinen lähestymistapa on iteratiivinen. Jokaisen iteraation aikana, toista tekijämatriisia pidetään vakiona ja toinen ratkaistaan käyttäen *pienimpien summien* (Mean Square Error, MSE) -algoritmia. Pienimpien summien algoritmeja käsitellään aliluvussa 2.1.2. Juuri ratkaistua tekijämatriisia pidetään vuorostaan vakiona kun ratkaistaan toista tekijämatriisia. [23] Spark ALS mahdollistaa massiivisen rinnakkaistamisen, sillä algoritmia voidaan suorittaa toisistaan erillään. Tämä on erinomainen ominaisuus suuren mittakaavan (large-scale) laskenta-algoritmille. [21]

Spark ALS on lohkotettu versio ALS tekijöihinjako-algoritmista. Ajatuksena on ryhmittää kaksi tekijäryhmää, *käyttäjät* ja *tuotteet*, lohkoihin. Ryhmittämistä seuraa kommunikaation vähentäminen lähettämällä jokaiseen tuotelohkoon vain yksi kopio jokaisesta käyttäjävektorista iteraation aikana. Vain ne käyttäjä-vektorit lähetetään, joita tarvitaan tuotelohkoissa. Vähennetty kommunikaatio saavutetaan laskemalla valmiiksi joitain tietoja suositusmatriisista, jotta voidaan päätellä jokaisen käyttäjän ulostulot ja jokaisen tuotteen sisääntulot. Ulostulolla tarkoitetaan niitä tuotelohkoja, joihin käyttäjä tulee myötävaikuttamaan. Sisääntulolla tarkoitetaan niitä ominaisuusvektoreita, jotka jokainen tuote ottaa vastaan niiltä käyttäjälohkoilta, joista ne ovat riippuvaisia. Tämä mahdollistaa sen, että voidaan lähettää vain taulukollinen ominaisuusvektoreita jokaisen käyttäjä- ja tuotelohkon välillä.

Vastaavasti tuotelohko löytää käyttäjän arviot ja päivittää tuotteita näiden viestien perusteella. [23]

Sen sijaan, että etsittäisiin alemman dimension arviot suositusmatriisille A, etsitäänkin arviot mieltymysmatriisi P:lle, jossa P:n alkiot saavat arvon 1 kun $r > 0$ ja arvon 0 kun $r <= 0$. Eksplisiittisen tuotearvion sijaan arvostelut kuvaavat käyttäjän mieltymyksen (r, rating) vahvuuden luottamusarvoa. [23]

$$A_i Y (Y^T Y)^{-1} = X_i \tag{2.2}$$

ALS operoi kiinnittämällä yhden tuntemattomista u_i ja v_j ja vaihtelemalla tätä kiinnittämistä. Kun toinen on kiinnitetty, toinen voidaan laskea ratkaisemalla pienimpien neliöiden ongelma. Tämä lähestymistapa on hyödyllinen, koska se muuttaa aiemman, ei-konveksin, ongelman neliömäiseksi, jolloin se voidaan ratkaista optimaalisesti. [1] Ei-konveksilla tarkoitetaan sellaista ongelmaa, jolla saattaa olla olemassa useita paikallisia ratkaisuja ja saattaa kestää kauan tunnistaa, onko ongelmalla ratkaisua lainkaan, tai että löydetty ratkaisu on myös globaali ratkaisu [29]. Alla on [1] mukainen yleinen kuvaus ALS-algoritmista. Esimerkissä mainittu RMSE-funktio esitellään tarkemmin aliluvussa 2.1.2.

Ohjelma 2.1 *ALS-algoritmin kuvaus [1]*

1. Alusta matriisi V.
2. Kiinnitä V, ratkaise U minimoimalla RMSE–funktio.
3. Kiinnitä U, ratkaise V minimoimalla RMSE–funktio.
4. Toista askeleita 2 ja 3 konvergenssiin asti.

Matriisi V alustetaan asettamalla ensimmäiseksi riviksi arvioitavan kohteen keskimääräinen arvio ja pieni satunnaisluku jäljelläoleviin alkioihin. Konvergenssilla tarkoitetaan jonkin ilmiön lähestymistä ajan kuluessa jotain tiettyä arvoa, tässä tapauksessa sitä, että RMSE ei enää pienene tarpeeksi.

2.1.2 RMSE

RMSE (Root Mean Square Error) on kenties suosituin ennustettujen arvosteluiden tarkkuuden evaluointiin käytetty metriikka. Sitä käytetään yleisesti regressioalgoritmien avulla luotujen mallien evaluointiin. Regressioalgoritmien yhteydessä virheellä tarkoitetaan havainnon todellisen sekä ennustetun numeroarvon välistä eroa. RMSE:n tuntemiseksi tulee tuntea ensin MSE (Mean Square Error). Kuten nimi viittaa, MSE on virheiden neliöiden keskiarvo ja se voidaan laskea neliöimällä jokaisen havainnon virhe ja laskemalla virheiden neliöiden keskiarvo. RMSE voidaan puolestaan laskea ottamalla neliöjuuri MSE:stä. Sekä RMSE että MSE edustavat opetusvirhettä ja ne ilmoittavat kuinka hyvin malli sovittuu opetusdataan. Niiden avulla saadaan selville havaintojen sekä ennustettujen arvojen välinen poikkeavuus. Alhaisemman MSE:n tai RMSE:n omaavan mallin sanotaan sovittuvan paremmin opetusdataan kuin korkeammat virhearvot omaavan mallin. [13]

Suosittelujärjestelmä luo ennustettuja arvosteluita $\hat{r}_{ui}$ testiaineistolle τ käyttäjä-tuote pareja (u, i) joille todelliset arviot r tunnetaan. [13] Ennustettujen ja todellisten arvioiden välinen RMSE saadaan laskettua seuraavasti:

$$RMSE = \sqrt{\frac{1}{|\tau|} \sum_{(u,i)\in\tau} (\hat{r}_{ui} - r_{ui})^2} \tag{2.3}$$

2.2 Amazon Web Services (AWS)

AWS (Amazon Web Services) on Amazonin tarjoama kokoelma pilvilaskentaan (cloud computing) tarkoitettuja tai sitä avustavia palveluita. Tässä työssä käytetään pääasiassa hyödyksi kahta AWS:n palvelua, EMR:ää (Elastic Map Reduce) sekä S3:a (Simple Storage Service). EMR:n avulla on mahdollista käyttää Big Data - sovelluskehyksiä, kuten Apache Sparkia. S3 on skaalautuva tietovarasto internetin tarpeisiin, jonka tarkoitus on helpottaa ohjelmistokehittäjien elämää. [5] [3] [4]

2.2.1 Elastic Map Reduce (EMR)

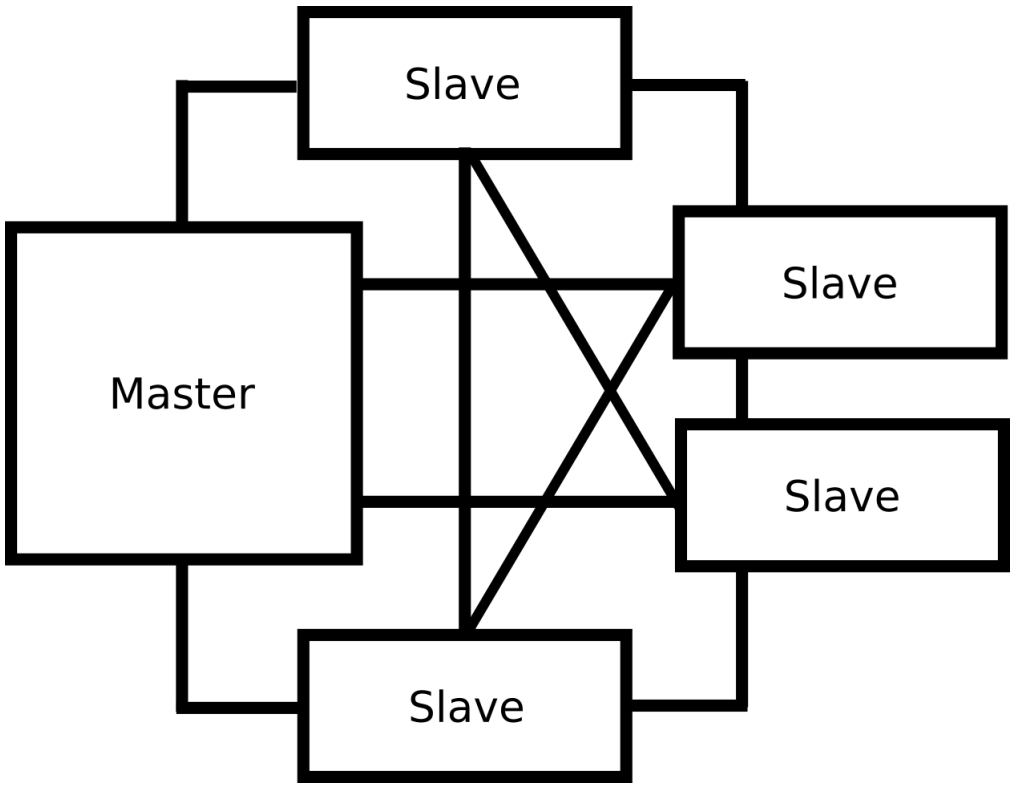
Amazon EMR tarjoaa hallitun klusterialustan (managed cluster platform), joka mahdollistaa suurten datamäärien prosessoinnin. EMR:ssä on mahdollista ajaa Apache Spark:ia ja se on kyvykäs liikuttelemaan suuria datamääriä AWS:n tietovarastoista, kuten S3:sta, ulos ja sisään. EMR käyttää hyväkseen dynaamisesti skaalautuvia

Amazon EC2 (Elastic Compute Cloud) instansseja. Niiden tarkoituksena on tehdä prosessoinnista helppoa, nopeaa ja kustannustehokasta. [3]

Klusteri on Amazon EMR:n keskeisin komponentti. Klusteri on kokoelma Amazon Elastic Compute Cloud (Amazon EC2) instansseja. Jokaista instanssia klusterissa kutsutaan solmuksi. Jokaisella solmulla on rooli eli tyyppi klusterin sisällä. Amazon EMR asentaa erilaisia sovelluskomponentteja jokaiseen solmuun, antaen jokaiselle solmulle roolin hajautetussa sovelluksessa kuten Apache Spark. [3]

Amazon EMR:ssä on seuraavanlaisia solmuja: Master, Core ja Task. Master on solmu, joka tarkkailee tehtävien ja klusterin tilaa ajamalla sovelluskomponentteja, jotka ovat vastuussa datan ja tehtävien jakamisesta Slave-solmuille. Core on Slave-solmu, jossa sijaitsee sovelluskomponteja, jotka ajavat tehtäviä ja tallentavat dataa HDFS:ään klusterissa. Task on Slave-solmu, jossa sijaitsee sovelluskomponentteja, jotka ajavat vain tehtäviä, Task-solmujen määrittäminen on vapaaehtoista. [3]

Kuva 2.2 *EMR-klusteri (muokattu lähteestä [3])*



Yllä oleva kuva 2.2 esittää EMR-klusterin, jossa on master-solmu sekä neljä slave-solmua.

2.2.2 Simple Storage Service (S3)

Amazon S3 (Simple Storage Service) on tietovarasto, joka tarjoaa yksinkertaisen rajapinnan, jonka avulla voidaan tallettaa tai noutaa minkä verran dataa tahansa, milloin vain ja mistä tahansa webissä. Se on suunniteltu tekemään web-mittakaavan (web-scale) laskennasta yksinkertaisempaa kehittäjille. Se antaa kehittäjille pääsyn samaan hyvin skaalautuvaan, luotettavaan, nopeaan ja edulliseen tietovarasto-infrastruktuuriin, jota Amazon käyttää itse globaalin verkkosivustokatraansa ajamiseen. [4]

Tässä työssä S3:a käytetään varastoimaan opetusdataa. Tarvittaessa Apache Sparkin laskennan välivaiheita voidaan tallentaa muistiin, jolloin niiden tulokset voitaisiin lukea suoraan ilman että niitä tarvitsee laskea uudelleen. Myös koko opetettu malli voitaisiin tallentaa S3:een ja esimerkiksi ladata vaikka omalle koneelle, jolloin AWS:ää tulisi myös käytettyä optimaalisesti, sillä EC2-instanssien varaaminen aiheuttaa maksuja koko ajan. Simple Storage Service:n tarjoama abstraktio on nimeltään *bucket*, joka on käsitteellisesti kansio, johon tietoa varastoidaan.

2.3 Scala

Scala on moniparadigmainen ohjelmointikieli, joka tukee sekä olio- että funktionaalista ohjelmointia. Funktionaalista ohjelmointia varten Scalasta löytyy tuki funktionaalisen ohjelmoinnin käsitteille, kuten muuttumattomat tietorakenteet ja funktiot ensimmäisen luokan kansalaisina. Olio-ohjelmointia varten Scalasta löytyy tuki käsitteille kuten luokat, oliot ja piirre (trait). Scala tukee myös kapselointia, perintää, moniperintää ja muita tärkeitä olio-ohjelmoinnin konsepteja. Scala on staattisesti tyypitetty kieli ja sillä kirjoitetut ohjelmat käännetään Scala-kääntäjää käyttäen. Scala on JVM-perustainen (Java Virtual Machine, Java-virtuaalikone) kieli, joten Scala kääntäjä kääntää sovelluksen Java-tavukoodiksi, jota voidaan ajaa missä tahansa Java-virtuaalikoneessa. Tavukooditasolla Scala-ohjelmaa ei voida erottaa Java-sovelluksesta. Scalan ollessa JVM-perustainen, Scala on täysin yhteensopiva Javan kanssa ja näin ollen Java-kirjastoja voidaan käyttää suoraan Scala-koodissa. Tästä syystä Scala-sovellukset hyötyvät suuresta Java-koodin määrästä. Vaikka Scala tukee sekä olio- että funktionaalista ohjelmointia, funktionaalista ohjelmointia suositaan. [13]

2.3.1 Perustyypit

Scalan perustyypit numeroiden esittämiseen ovat Byte, Short, Int, Long, Float ja Double. Lisäksi Scalassa on perustyypit Char, String ja Boolean. Char on 16 bittinen etumerkitön Unicode-merkki. String on jono Char:eja. Boolean esittää totuusarvoa tosi (true) tai epätosi (false). [13]

Javasta poiketen Scalassa ei ole ollenkaan primitiivisiä tyyppejä vaan jokainen tyyppi on toteutettu luokkana. Käännöksen aikana kääntäjä tarvittaessa automaattisesti muuntaa Scala-tyypit Javan primitiivisiksi tyypeiksi. [13]

2.3.2 Muuttujat

Scalassa on kahdentyyppisiä muuttujia: muuttuvia ja vakioita. Muuttuva muuttuja määritellään avainsanan *var* avulla. Muuttuvaa muuttujaa ei voida asettaa uudelleen luomisen jälkeen. Var:ien käyttöä ei suositella, mutta joskus niiden käyttämisellä saadaan aikaan yksinkertaisempaa ohjelmakoodia ja tästä syystä Scala tukee myös muuttuvia muuttujia. Vakiota, *val*, ei sen sijaan voida antaa uudelleen luomisen jälkeen. [13]

Syntaksi *val*:in ja *var*:in luomiseksi on esitetty alla olevassa ohjelmassa 2.2

Ohjelma 2.2 *Muuttujien luominen ja uudelleen asettaminen*

```
1  var x = 10
2  x = 20
3  val y = 10
```

Mikäli vakiota yritetään uudelleenmäärittää myöhemmin ohjelmassa, kääntäjä antaa virheen. Huomionarvoista ylläolevassa syntaksissa on se, että Scala kääntäjä ei pakota määrittelemään muuttujan tyyppiä sillon kuin kääntäjä pystyy päättelemään (type inference) sen.

Ohjelma 2.3 *Muuttujan luominen tyyppimäärittelyn avulla*

```
1  var x: Int = 10
2  val y: Int = 10
```

Yllä olevassa ohjelmassa 2.3 on määritetty muuttuja sekä vakio tyyppien kanssa.

2.3.3 Funktiot

Funktio on lohko suoritettavaa koodia, joka palauttaa arvon. Se on konseptuaalisesti samankaltainen kuin matematiikassa: funktio ottaa sisääntulon ja palauttaa ulostulon. [13]

Scalan funktiot ovat ensimmäisen luokan kansalaisia, jolla tarkoitetaan, että funktiota voidaan:

- käyttää kuten muuttujaa
- antaa syötteenä toiselle funktiolle
- määritellä nimettömänä funktioliteraalina
- asettaa muuttujaan
- määritellä toisen funktion sisällä
- palauttaa toisen funktion ulostulona

Scalassa funktio määritellään avainsanalla *def*. Funktion määrittely aloitetaan funktion nimellä, jota seuraa sulkeissa olevat, pilkulla erotetut, parametrit tyyppimäärittelyineen. Parametrien jälkeen funktiomäärittelyyn tulee kaksoispiste, funktion ulostulon tyyppi, yhtäsuuruusmerkki sekä funktion runko joko aaltosulkeissa tai ilman. [13]

Ohjelma 2.4 *add-Funktio*

```
1  def add(first: Int, second: Int): Int = {
2    val sum = first + second
3    sum
4  }
```

Ylläolevassa ohjelmassa 2.4 funktion nimi on *add* ja se ottaa kaksi *Int* tyyppistä sisääntuloa. Funktio palauttaa *Int*-tyyppisen arvon, jonka se muodostaa lisäämällä

annetut sisääntulot yhteen ja palauttamalla tuloksen. Scalassa kaikki lausekkeet ovat arvon palauttavia lausekkeita, joten funktion rungon viimeisen lausekkeen arvosta tulee funktion paluuarvo [13].

Scala mahdollistaa myös lyhyemmän version samasta funktiosta:

Ohjelma 2.5 *add-Funktio*

```
1  def add(first: Int, second: Int) = first + second
```

Ohjelma 2.5 tekee täsmälleen saman asian kuin ohjelma 2.4, mutta se on vain kirjoitettu lyhyemmin. Paluuarvon tyyppi on jätetty antamatta, sillä kääntäjä pystyy päättelemään sen koodista. Paluuarvon tyyppi suositellaan kuitenkin annettavan aina. Aaltosulkeet on myöskin jätetty pois, sillä ne ovat pakolliset vain kun funktion runko sisältää useamman kuin yhden lausekkeen. [13]

3. SUOSITTELIJAJÄRJESTELMÄT

Suosittelulla tarkoitetaan tehtävää, jossa tuotteita suositellaan käyttäjille. Yksinkertaisin suosittelu tapahtuu ihmiseltä toiselle, ilman tietokoneita. Ihmiset voivat kuitenkin tehokkaasti suositella vain niitä asioita, jotka ovat itse henkilökohtaisesti kokeneet. Tällöin suosittelijajärjestelmistä tulee hyödyllisiä, sillä ne voivat mahdollisesti tarjota suosituksia sadoista tai jopa tuhansista erilaisista tuotteista. Suosittelijajärjestelmät ovat joukko tekniikoita ja ohjelmistoja, jotka tarjovat suosituksia mahdollisesti hyödyllisistä tuotteista. Tuotteella tarkoitetaan tässä yhteydessä yleistä asiaa, jota järjestelmä suosittelee henkilölle. Suosittelujärjestelmät rakennetaan yleensä suosittelemaan vain tietyn tyyppisiä tuotteita, kuten esimerkiksi kirjoja tai elokuvia. [20]

Suosittelijajärjestelmien tarkoitus on auttaa asiakkaita päätöksenteossa, tuotteiden määrän ollessa valtava. Tavallisesti suositukset ovat räätälöityjä, millä tarkoitetaan että suositukset eroavat käyttäjien tai käyttäjäryhmien välillä. Suositukset voivat olla myös räätälöimättömiä ja niiden tuottaminen onkin usein yksinkertaisempaa. Lista, joka sisältää 10 suosituinta tuotetta, on esimerkki räätälöimättömästä suosittelusta. Järjestäminen tehdään ennustamalla kaikista sopivimmat tuotteet käyttäjän mieltymysten tai vaatimusten perusteella. Tämän mahdollistamiseksi suosittelijajärjestelmän on kerättävä käyttäjältä tämän mieltymykset. Mieltymykset voivat olla suoraan käyttäjältä kysyttyjä tai käyttäjän antamia tuotearvioita tai ne voidaan tulkita käyttäjän toiminnasta kuten klikkauksista, sivun katselukerroista tai ajasta jonka käyttäjä on viipynyt tietyllä tuotesivulla. Suosittelijajärjestelmä voisi esimerkiksi tulkita tuotesivulle päätymisen todisteeksi mieltymyksestä sivun tuotteista. [20]

Suosittelijajärjestelmien kehitys alkoi melko yksinkertaisesta havainnosta: ihmiset tapaavat luottaa toisten ihmisten suosituksiin tehdessään rutiininomaisia päätöksiä. On esimerkiksi yleistä luottaa vertaispalautteeseen valittaessa kirjaa luettavaksi tai luottaa elokuvakriitikoiden kirjoittamiin arvioihin. Ensimmäinen suosittelijajärjes-

telmä yritti matkia tätä käytöstä etsimällä verkkoyhteisöstä suosituksia aktiiviselle käyttäjälle. Suositukset haettiin käyttämällä algoritmeja. Tämä lähestymistapa on tyypiltään yhteisösuodattamista. Yhteisösuodattamisessa ideana on että, jos käyttäjät pitivät samankaltaisista tuotteista aikaisemmin, he luultavasti pitävät samoja tuotteita ostaneiden henkilöiden suosituksia merkityksellisinä. [20]

Verkkokauppojen kehityksen myötä syntyi tarve suosittelulle vaihtoehtojen rajoittamiseksi. Käyttäjät kokivat aina vain vaikeammaksi löytää oikeat tuotteet sivustojen suurista valikoimista. Tiedon määrän räjähdysmäinen kasvaminen internetissä on ajanut käyttäjät tekemään huonoja päätöksiä. Vaihtoehdot ovat hyväksi, mutta vaihtoehtojen lisääntyminen on alkanut hyödyn tuottamisen sijaan heikentää kuluttajien hyvinvointia. [20]

Viimeaikoina suosittelijajärjestelmät ovat osoittautuneet tehokkaaksi lääkkeeksi tiedon *ylimääräongelmaa* vastaan. Suosittelijajärjestelmät käsittelevät tätä ilmiötä tarjoamalla uusia, aiemmin tuntemattomia, tuotteita jotka ovat todennäköisesti merkityksellisiä käyttäjälle tämän nykyisessä tehtävässä. Kun käyttäjä pyytää suosituksia, suosittelujärjestelmä tuottaa suosituksia käyttämällä tietoa ja tuntemusta käyttäjistä, saatavilla olevista tuotteista ja aiemmista *tapahtumista* (transactions). Tutkittuaan tarjotut suositukset, käyttäjä voi hyväksyä tai hylätä ne tarjoten epäsuoraa ja täsmällistä palautetta suosittelijalle. Tätä uutta tietoa voidaan myöhemmin käyttää hyödyksi tuotettaessa uusia suosituksia seuraaviin käyttäjän ja järjestelmän vuorovaikutuksiin. [20]

Verrattuna klassisten tietojärjestelmien, kuten tietokantojen ja hakukoneiden, tutkimukseen, suosittelijajärjestelmien tutkimus on verrattain tuoretta. Suositteljiajärjestelmistä tuli itsenäisiä tutkimusalueita 90-luvun puolivälissä. Viimeaikoina mielenkiinto suosittelujärjestelmiä kohtaan on kasvanut merkittävästi. Esimerkiksi verkkosivustoissa, kuten Amazon.com, YouTube, Netflix sekä IMDB, suosittelujärjestelmillä on iso rooli. Suosittelujärjestelmien tutkimiseen ja kehittämiseen omistettuja konferensseja on myös olemassa, kuten RecSys ja AI Communications (2008). [20]

Suosittelujärjestelmällä voidaan ajatella olevan kaksi päätarkoitusta: palveluntarjoajan avustaminen ja arvon tuottaminen palvelun käyttäjälle. Suosittelujärjestelmän on siis tasapainoiteltava sekä palveluntarjoajan että käyttäjän tarpeiden välillä. [20] Palveluntarjoaja voi esimerkiksi ottaa suosittelujärjestelmän avuksi parantamaan tai monipuolistamaan myyntiä, lisäämään käyttäjien tyytyväisyyttä, lisäämään käyttäjien uskollisuutta tai ymmärtämään paremmin mitä käyttäjä haluaa [20]. Käyttäjä

puolestaan saattaa haluta suosituksena tuotesarjan, apua selaamiseen tai mahdollistaa muihin vaikuttamisen. [20]

GroupLens [2], BookLens [8] ja MovieLens [12] olivat uranuurtajia suosittelujärjestelmien tutkimisessa ja kehittämisessä. GroupLens on tutkimuslaboratorio *computer science and engineering* -laitoksella Minnesotan Yliopistossa, joka on erikoistunut muun muassa suosittelujärjestelmiin ja verkkoyhteisöihin [2]. BookLens on on GroupLensin rakentama kirjojen suosittelujärjestelmä [8]. MovieLens on GroupLensin ylläpitämä elokuvien suosittelujärjestelmä [12]. Uranuurtavan tutkimuksen lisäksi nämä sivustot julkaisivat kaikkien saataville aineistoja, joka ei ollut yleistä tuohon aikaan. [2]

3.1 Suositustekniikat

Suosittelujärjestelmällä täytyy olla ymmärrys tuotteista, jotta se pystyy tekemään suosituksia. Tämän mahdollistamiseksi järjestelmän täytyy pystyä ennustamaan tuotteen käytännöllisyys tai ainakin verrata tuotteiden hyödyllisyyttä ja tämän perusteella päättää suositeltavat tuotteet. Ennustamista voidaan luonnostella yksinkertaisella personoimattomalla suosittelualgoritmilla, joka suosittelee vain suosituimpia elokuvia. Tätä lähestymistapaa voidaan perustella sillä, että tarkemman tiedon puuttuessa käyttäjän mieltymyksistä, elokuva, josta muutkin ovat pitäneet on todennäköisesti myös keskivertokäyttäjän mieleen, ainakin enemmän kuin satunaisesti valikoitu elokuva. Suosittujen elokuvien voidaan siis katsoa olevan kohtuullisen osuvia suosituksia keskivertokäyttäjälle. [20]

Seuraavassa tuotetta kuvataan symbolilla i (item) ja käyttäjää kuvataan symbolilla u (user). Tuotteen i hyödyllisyyttä käyttäjälle u voidaan mallintaa reaaliarvoisella funktiolla $R(u, i)$, kuten yleensä tehdään *yhteisösuodatuksessa* ottamalla huomioon käyttäjien antamat arviot tuotteista. Yhteisösuodatuksessa suosittelijan perustehtävä on ennustaa R:n arvoa käyttäjä-tuote pareille ja laskea arvio todelliselle funktiolle R. Laskiessaan tätä ennustetta käyttäjälle u ja tuotejoukolle, järjestelmä suosittelee tuotteita, joilla on suurin ennustettu hyödyllisyys. Ennustettujen tuotteiden määrä on usein paljon pienempi kuin tuotteiden koko määrä, joten voidaan sanoa, että suosittelijajärjestelmä suodattaa käyttäjälle suositeltavat tuotteet. [20]

Suosittelijajärjestelmät eroavat toisistaan kohdistetun toimialan, käytetyn tiedon ja erityisesti siinä kuinka suositukset tehdään, jolla tarkoitetaan suosittelualgoritmia

[20]. Tässä työssä keskitytään vain yhteen suositteluteknikoiden luokkkaan, yhteisösuodatukseen, sillä tätä menetelmää käytetään Apache Sparkin MLlib kirjastossa.

Yhteisösuodatusta käyttävät suosittelijajärjestelmät perustuvat käyttäjien yhteistyöhön. Niiden tavoitteena on tunnistaa malleja käyttäjän mielenkiinnoista voidakseen tehdä suunnattuja suosituksia [1]. Tämän lähestymistavan alkuperäisessä toteutuksessa suositellaan aktiiviselle käyttäjälle niitä tuotteita, joita muut samankaltaiset käyttäjät ovat pitäneet aiemmin [20]. Käyttäjä arvostelee tuotteita. Seuraavaksi algoritmi etsii suosituksia perustuen käyttäjiin, jotka ovat ostaneet samanlaisia tuotteita tai perustuen tuotteisiin, jotka ovat eniten samanlaisia käyttäjän ostoshistoriaan verrattuna. Yhteisösuodatus voidaan jakaa kahteen kategoriaan, jotka ovat *tuotepohjainen- ja käyttäjäpohjainen yhteisösuodatus*. Yhteisösuodatus on eniten käytetty ja toteutettu tekniikka suositusjärjestelmissä [11] [20] [9].

Yhteisösuodatus analysoi käyttäjien välisiä suhteita ja tuotteiden välisiä riippuvuuksia tunnistaakseen uusia käyttäjä-tuote -assosiaatioita [14]. Päätelmä siitä, että käyttäjät voisivat pitää samasta laulusta, koska molemmat kuuntelevat muita samankaltaisia lauluja on esimerkki yhteisösuodatuksesta [21].

Koska yhteisösuodatuksessa suosittelu perustuu pelkästään käyttäjän arvosteluihin tuotteesta, yhteisösuodatus kärsii ongelmista jotka tunnetaan nimillä *uusi käyttäjäongelma* ja *uusi tuoteongelma* [11]. Ellei käyttäjä ole arvostellut yhtään tuotetta, algoritmi ei kykene tuottamaan myöskään yhtään suositusta. Muita yhteisösuodatuksen haasteita ovat *kylmä aloitus* sekä *niukkuus* (sparsity). Kylmällä aloituksella tarkoitetaan sitä, että tarkkojen suositusten tuottamiseen tarvitaan tyypillisesti suuri määrä dataa. Niukkuudella tarkoitetaan sitä, että tuotteiden määrä ylittää usein käyttäjien määrän. Tästä johtuen suhteiden määrä on todella niukka, sillä useat käyttäjät ovat arvostelleet tai ostaneet vain murto-osan tuotteiden koko määrästä. [1]

3.1.1 Muistiperustainen yhteisösuodatus

Muistiperustaisissa menetelmissä käyttäjä-tuote -suosituksia käytetään suoraan uusien tuotteiden ennustamiseksi. Tämä voidaan toteuttaa kahdella tavalla, käyttäjäpohjaisena suositteluna tai tuotepohjaisena suositteluna.

Seuraavat kappaleet kuvaavat käyttäjäpohjaista yhteisösuodatusta ja tuotepohjais-

ta yhteisösuodatusta. Ohjelmat 3.1, 3.2 ja 3.3 on luonnosteltu tekstipohjaisten selitysten perusteella käyttäen Scala-syntaksia.

Tuotepohjainen yhteisösuodatus

Tuotepohjaisessa yhteisösuodatuksessa (Item-based collaborative filtering, IBCF) algoritmi aloittaa etsimällä samankaltaisia tuotteita käyttäjän ostoshistoriasta [11]. Seuraavaksi mallinnetaan käyttäjän mieltymykset tuotteelle perustuen saman käyttäjän tekemiin arvosteluihin [20]. Alla oleva ohjelma 3.1 esittelee tuotepohjaisen yhteisösuodatuksen idean jokaiselle uudelle käyttäjälle.

Ohjelma 3.1 *Tuotepohjaisen yhteisösuodatuksen algoritmi [11]*

```
1  // Jokaiselle kahdelle tuotteelle, mittaa kuinka samankaltaisia ne ovat.
2
3  case class Item(id: Int, feature1: String, feature2: String)
4  val items: Seq[Item] = Seq(Item(), Item(), Item())
5  case class Similarity(item1: Item, item2: Item, similarity: Double)
6
7  val similarItems: Seq[Similarity] = items.map { i1 =>
8    items.map { i2 => Similarity(i1, i2, getSimilarity(i1, i2)) }
9  }
10
11 // Tunnista k−samankaltaisinta tuotetta jokaiselle tuotteelle.
12
13 case class Similarities(item: Item, kMostSimilarItems: Seq[Similarity])
14 val similarities = findKMostSimilarItems(similarItems)
15
16 // Jokaiselle käyttäjälle, tunnista tuotteet,
17 // jotka ovat eniten samankaltaisia käyttäjän ostoshistorian kanssa.
18
19 val similars = users.map { user =>
20   user.purchases.map { purchase => findSimilarItem(purchase, similarities) }
21 }
```

Amerikan suurimman verkkokaupan, Amazon.com:in, on aiemmin tiedetty käyttäneen tuote-tuotteeseen-yhteisösuodatusta. Tässä toteutuksessa algoritmi rakentaa samankaltaisten tuotteiden taulun etsimällä tuotteita, joita käyttäjät tapaavat ostaa yhdessä. Seuraavaksi algoritmi etsii käyttäjän ostoshistoriaa ja arvosteluita vastaavat tuotteet, yhdistää nämä tuotteet ja palauttaa suosituimmat tai eniten korreloivat tuotteet. [16]

Käyttäjäpohjainen yhteisösuodatus

Tuotepohjaisessa yhteisösuodatuksessa (User-based collaborative filtering, UBCF) algoritmi aloittaa etsimällä samankaltaisimmat käyttäjät. Seuraava askel on arvostella samankaltaisten käyttäjien ostamat tuotteet. Lopuksi valitaan parhaan arvosanan saaneet tuotteet. Samankaltaisuus saadaan laskettua vertaamalla käyttäjien ostoshistorioiden samankaltaisuutta. [20]

Askeleet jokaiselle uudelle käyttäjälle käyttäjäpohjaisessa yhteisösuodatuksessa on esitetty ohjelmissa 3.2 ja 3.3.

Ohjelma 3.2 *Käyttäjäpohjaisen yhteisösuodatuksen algoritmi osa 1/2 [11]*

```
1  // Mittaa jokaisen käyttäjän samankaltaisuus uuteen käyttäjään.
2
3  case class Similarity(userId1: Int, userId2: Int, score: Double)
4
5  val newUser: User = User("Adam", 31, purchases)
6  val similarities = users.map { user =>
7    Similarity(newUser.id, user.id, cosineSimilarity(user, newUser)
8  }
9
10 // Tunnista samankaltaisimmat käyttäjät.
11 // Vaihtoehtoja on kaksi: Voidaan valita joko parhaat k käyttäjää
12 // (k-nearest neighbors) tai voidaan valita käyttäjät,
13 // joiden samankaltaisuus ylittää tietyn kynnysarvon.
14
15 val mostSimilarUsers = similarities.filter(_.score > 0.8)
```

Ohjelma 3.3 *Käyttäjäpohjaisen yhteisösuodatuksen algoritmi osa 2/2 [11]*

```
1  // Arvostele samankaltaisimpien käyttäjien ostamat tuotteet.
2  // Arvostelu saadaan joko keskiarvona kaikista tai painotettuna keskiarvona,
3  // käyttäen samankaltaisuuksia painoina.
4
5  val ratedItems = mostSimilarUsers.map { user =>
6    user.purchases.map { purchase =>
7      val purchases = mostSimilarUsers.map { usr =>
8        usr.purchases. filter (_.id == purchase.id)
9      }
10     purchases.sum() / purchases.size
11   }
12 }
13
14 // Valitse parhaiten arvostellut tuotteet.
15
16 val topRatedItems = ratedItems.take(10)
```

3.1.2 Mallipohjainen yhteisösuodatus

Muistipohjaisen yhteisösuodatuksen käyttäessä tallennettuja suosituksia suoraan ennustamisen apuna, mallipohjaisissa lähestymistavoissa näitä arvosteluita käytetään ennustavan mallin oppimiseen. Perusajatus on mallintaa käyttäjä-tuote vuorovaikutuksia tekijöillä jotka edustavat käyttäjien ja tuotteiden piileviä ominaisuuksia (latent factors) järjestelmässä. Piileviä ominaisuuksia ovat esimerkiksi käyttäjän mieltymykset ja tuotteiden kategoriat. Tämä malli opetetaan käyttämällä saatavilla olevaa dataa ja myöhemmin käytetään ennustamaan käyttäjien arvioita uusille tuotteille. [20] ALS-algoritmi on esimerkki mallipohjaisesta yhteisösuodatusalgoritmista.

4. APACHE SPARK

Apache Spark on sovelluskehys hajautettujen ohjelmien kirjoittamiseen. [21]. Spark-ohjelmia voidaan kirjoittaa Java-, Scala-, Python- sekä R-ohjelmointikielillä. Jokainen Spark-sovellus koostuu driver-ohjelmasta sekä yhdestä tai useammasta executor-ohjelmasta. Driver on ohjelma, joka ajaa käyttäjän pääohjelmaa ja hajauttaa laskennan klusteriin. Executor on yksi kone klusterissa. [21]

Spark voidaan esitellä kuvailemalla sen edeltäjää, MapReduce:a, ja sen tarjoamia etuja. MapReduce tarjosi yksinkertaisen mallin ohjelmien kirjoittamiseen ja pystyi suorittamaan kirjoitettua ohjelmaa rinnakkain sadoilla tietokoneilla. MapReduce skaalautuu lähes lineaarisesti datan koon kasvaessa. Suoritusaikaa hallitaan lisäämällä lisää tietokoneita suorittamaan tehtävää. [21]

Apache Spark säilyttää MapReduce:n lineaarisen skaalautuvuuden ja vikasietokyvyn mutta laajentaa sitä kolmella merkittävällä tavalla. Ensiksi, MapReducessa map- ja reduce-tehtävien väliset tulokset täytyy kirjoittaa levylle kun taas Spark kykenee välittämään tulokset suoraan liukuhihnan (pipeline) seuraavalle vaiheelle. Toiseksi, Apache Sparkin voidaan ajatella kohtelevan kehittäjiä paremmin tarjoamalla monipuolisen joukon transformataatioita, joiden avulla voidaan muutamalla koodirivillä ilmaista monimutkaisia liukuhihnoja (pipelines, ohjelmistoja). Kolmanneksi, Spark esittelee muistissa tapahtuvan prosessoinnin tarjoamalla abstraktion nimeltä *Resilient Distributed Dataset (RDD)*. RDD tarjoaa kehittäjälle mahdollisuuden materialisoida minkä tahansa askeleen liukuhihnassa ja tallentaa sen muistiin. Tämä tarkoittaa sitä, että tulevien askelien ei tarvitse laskea aiempia tuloksia uudelleen ja tällöin on mahdollista jatkaa juuri käyttäjän haluamasta askeleesta. Aiemmin tämänkaltaista ominaisuutta ei ole ollut saatavilla hajautetun laskennan järjestelmissä. [21]

Vaikka Spark-ohjelmia voidaankin kirjottaa usealla ohjelmointikielellä, Scalan käyttämisellä saavutetaan kuitenkin muutamia etuja, joita muut kielet eivät tarjoa. Tehokkuus saattaa parantua, sillä datan siirtäminen eri kerrosten välillä tai muunnos-

ten suorittaminen datalle voi johtaa heikompaan tehokkuuteen. Spark on kirjoitettu Scala-ohjelmointikielellä, joten viimeisimmät ja parhaimmat (latest and greatest) ominaisuudet ovat aina käytössä, eikä niiden käännöstä tarvitse odotella. Spark ohjelmoinnin filosofia on helpompi ymmärtää kun Sparkia käytetään kielellä, jolla se on rakennettu. Suurin hyöty, jonka Scalan käyttäminen tarjoaa, on kuitenkin kehittäjäkokemus joka tulee saman ohjelmointikielen käyttämisestä kaikkeen. Datan tuonti, manipulointi ja koodin lähettäminen klustereihin hoituvat samalla kielellä. [21]

Spark-jakelun mukana toimitetaan luku-evaluointi-tulostus-silmukka (Read Eval Print Loop, REPL). REPL on komentorivityökalu, joka mahdollistaa uusien asioiden nopean testailun konsolissa, eikä sovelluksista tarvitse rakentaa itsenäisiä (self-contained) alusta asti. Kun REPLissä kehitetyn sovelluksen tai sovelluksen osan voidaan katsoa olevan tarpeeksi valmis, on järkevää tehdä siitä koottu kirjasto (JAR). Näin varmistutaan ettei ohjelmakoodia tai tuloksia pääse katoamaan, vaikkakin REPL tarjoaa samantapaisen muistin komentohistoriasta kuin perinteinen komentorivikin.

JAR eli Java ARchive on suosittuun ZIP-tiedostoformaattiin perustuva alustariippumaton tiedostoformaatti, jota käytetään kokoamaan monta tiedostoa yhdeksi tiedostoksi. [18] JVM (Java Virtual Machine, Java-virtuaalikone) on abstrakti tietokone. Kuten oikea tietokone, se omaa käskykannan ja muokkaa useita muistialueita ajon aikana. JVM ei tiedä mitään ohjelmointikielistä, kuten Scala tai Java, vaan se operoi ainoastaan *class*-tiedostoilla, jotka ovat binääritiedostoja. Class-tiedosto sisältää JVM-käskyt sekä symbolitaulun. [19]

4.1 Resilient Distributed Dataset API (RDD API)

Resilient Distributed Dataset (RDD) on Sparkin tarjoama pääabstraktio. RDD on muuttumaton, partitioitu elementtikokoelma, joka voidaan hajauttaa klusterin useiden koneiden välillä. [27]

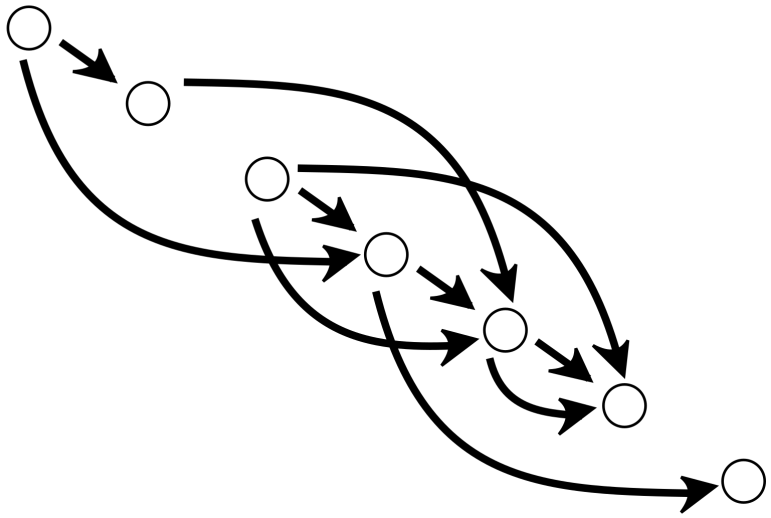
RDD:t tukevat kahdenlaisia operaatioita: transformaatioita ja toimia (actions). Transformaatioilla luodaan uusia datasettejä olemassaolevista dataseteistä. Toimet palauttavat arvon driver-ohjelmaan laskennan valmistuttua. *map*-operaatio on esimerkki transformaatiosta ja *reduce*-operaatio on esimerkki toimesta. Molemmat operaatiot käyvät annetun datasetin läpi ja kohdistavat annetun funktion jokaiselle elementille, mutta *map* palauttaa uuden RDD:n ja *reduce* palauttaa tuloksensa

driver-ohjelmalle. [25]

Kaikki transformaatiot Sparkissa ovat laiskasti evaluoituvia, jolla tarkoitetaan sitä, että lausekkeen evaluointia viivytetään siihen asti kun sen arvoa tarvitaan. Kun uusi RDD luodaan, mitään laskentaa ei oikeasti vielä tapahdu, vaan Spark tietää missä data sijaitsee tai miten data saadaan laskettua, kun tulee aika tehdä sille jotain. Tämä suunnittelu mahdollistaa Sparkin tehokkaamman toiminnan. Esimerkiksi *map* ja *reduce* -operaatioista koostuva liukuhihna suoriutuu tehokkaammin, sillä *map*-transformaation luoma datasetti voidaan jättää palauttamatta driver-ohjelmaan, koska *reduce*-toimen vastaus palautuu. [28]

RDD voidaan luoda kahdella tavalla, rinnakkaistamalla (parallelize) tai viittaamalla ulkoiseen aineistoon. Rinnakkaistamisessa olemassaoleva Scala-kokoelma saadaan muunnettua RDD:ksi. Ulkoiseen ainestoon viittaamisella tarkoitetaan viittaamista aineistoon ulkoisessa varastointijärjestelmässä kuten *HDFS*:sä tai *HBase*:ssa. [25]

Kuva 4.1 *Directed Acyclic Graph (muokattu lähteestä [10])*



RDD:t voidaan tallentaa muistiin, jolloin ohjelmistokehittäjä voi uudelleenkäyttää niitä tehokkaasti rinnakkaisissa operaatioissa. RDD:t voivat palautua solmuvirheistä automaattisesti käyttäen Directed Acyclic Graph (DAG) -moottoria. DAG tukee asyklistä datavirtaa, jolla tarkoitetaan sitä, että jokainen graafin kaari kulkee topologisessa järjestyksessä aiemmasta myöhempään. Kuvassa 4.1 on havainnollistettu DAG:in asyklistä datavirtaa. Jokaista Spark-työtä kohti luodaan DAG klusterissa suoritettavan tehtävän tasoista. Verrattuna MapReduceen, joka luo DAGin kahdesta ennaltamäärätystä tilasta (Map ja Reduce), Sparkin luomat DAGit voivat sisältää

minkä tahansa määrän tasoja. Tästä syystä jotkin työt voivat valmistua nopeammin kuin ne valmistuisivat MapReducessa. Yksinkertaisimmat työt voivat valmistua vain yhden tason jälkeen ja monimutkaisemmat tehtävät valmistuvat yhden monitasoisen ajon jälkeen, ilman että niitä täytyy pilkkoa useampiin töihin. [30]

4.2 Dataset API

Dataset (DS) on vahvasti tyypitetty kokoelma domain spesifisiä objekteja, jotka voidaan muuntaa rinnakkain käyttäen funktionaalisia tai relaatio-operaatioita. DS on kehitetty korvaamaan RDD Sparkissa. Dataset:ille olemassa olevat operaatiot on jaettu *transformaatioihin* (transformations) ja *toimiin* (actions). Transformaatiot ovat operaatioita, jotka luovat uusia Dataset-objekteja, kuten *map*, $filter$, *select* ja *aggregate*. Toimet ovat operaatioita jotka suorittavat laskentaa ja palauttavat tuloksia. Toimia ovat esimerkiksi *count*, *show* tai datan kirjoittaminen tiedostojärjestelmään. [26]

Kuten RDD:t, Dataset-instanssit ovat laiskasti evaluoituvia, jolla tarkoitetaan sitä, että laskenta aloitetaan vasta kun toimintoa kutsutaan tai instanssin arvoa tarvitaan. Dataset on pohjimmiltaan looginen suunnitelma, jolla kuvataan datan tuottamiseen tarvittava laskenta. Toimea kutsuttaessa, Sparkin kyselyoptimoija (query optimizer) optimoi loogisen suunnitelman ja generoi fyysisen suunnitelman. Fyysinen suunnitelma takaa rinnakkaisesti ja hajautetusti tapahtuvan tehokkaan suorituksen. Loogista suunnitelmaa, kuten myös optimoitua fyysistä suunnitelmaa, voidaan tutkia käyttämällä DS:n *explain* funktiota. [26]

Domain-spesifisten olioiden tehokkaaseen tukemiseen tarvitaan enkooderia. Enkooderilla tarkoitetaan ohjelmaa, joka muuntaa tietoa jonkin algoritmin mukaisesti ja tässä tapauksessa sitä käytetään yhdistämään domain-spesifinen tyyppi T Sparkin sisäiseen tyyppijärjestelmään. Esimerkiksi luokan $Person$ tapauksessa, joka sisältää kentät nimi (merkkijono) ja ikä (kokonaisluku), enkooderia voidaan käyttää käskemään Sparkia luomaan koodia ajon aikana joka sarjallistaa $Person$ olion binäärirakenteeksi. Generoidulla binäärirakenteella on usein pienempi muistijalanjälki ja se on myös optimoitu tehokkaaseen dataprosessointiin. Datan binääriesitys voidaan tarkistaa käyttämällä DS:n tarjoamaa *schema* funktiota. [26]

Dataset voidaan luoda tyypillisesti kahdella eri tavalla. Yleisin tapa on käyttää *SparkSession*:in tarjoamaa *read*-funktiota ja osoittaa Spark joihinkin tiedostoihin

tiedostojärjestelmässä, kuten ohjelmassa 4.1 esitettyyn *json*-tiedostoon:

Ohjelma 4.1 *Esimerkki JSON-tiedosto*

```
[{
  "name": "Matt",
  "salary": 5400
}, {
  "name": "George",
  "salary": 6000
}]
```

Ohjelmassa 4.2 on esitetty Datasetin luominen käyttäen *spark.read* -funktiota.

Ohjelma 4.2 *Uuden Dataset-olion luominen käyttäen read-funktiota*

```
1  case class Person(id: BigInt, firstName: String, lastName: String)
2  val people = spark.read.json("./people.json").as[Person]
```

Case-luokat ovat tavallisia Scala-luokkia, jotka ovat:

- Oletustarvoisesti muuttumattomia (immutable)
- Hajoitettavia (decomposable) hahmonsovituksen (pattern matching) avulla
- Vertailtavissa viitteiden sijasta rakenteellisen samankaltaisuuden mukaan
- Yksinkertaisempia luoda (instantiate) ja käyttää

Mikäli tyyppimuunnos (casting) jätettäisiin tekemättä, päädyttäisiin luomaan DataFrame-olio, jonka sisäinen mallin (schema) Spark pyrkisi arvaamaan. DataFrame-rajapintaa käsitellään seuraavassa aliluvussa. Tyyppimuunnos tehdään käyttämällä *as*-avainsanaa.

Ohjelma 4.3 *SparkSession-kontekstin luominen*

```
1  val spark = SparkSession
2    .builder
3    .appName("MovieLensALS")
4    .getOrCreate()
```

SparkSession on Spark-ohjelmoinnin lähtökohta, kun halutaan käyttää Dataset- ja DataFrame-rajapintoja. Ohjelmassa 4.3 luodaan SparkSession ketjuttamalla rakentajan kutsuja. [26]

Dataset-oliot ovat samankaltaisia kuin RDD:t, sillä nekin tarjoavat vahvan tyypityksen ja mahdollisuuden käyttää voimakkaita lambda-funktioita [28]. Lambda-funktiolla tarkoitetaan yleisesti anonyymiä funktiota, jota ei olla sidottu muuttujaan. Perinteisen sarjallistamisen, kuten Java-sarjallistamisen, sijaan käytetään erikoistunutta enkooderia olioiden sarjallistamiseen. Sarjallistamisella tarkoitetaan olion muuntamista tavuiksi, jolloin olion muistijalanjälki pienenee. Yleisesti sarjallistamista tarvitaan datan prosessointiin tai verkon yli lähettämiseen. Molempia, sekä enkoodereita että sarjallistamista käytetään olioiden muuntamiseen tavuiksi, mutta enkooderit luodaan dynaamisesti koodissa. Enkooderit käyttävät sellaista muotoa, että Spark kykenee suorittamaan monenlaisia operaatioita, kuten suodattamista, järjestämistä ja hajautusta (hashing), ilman että tavuja tarvitsee purkaa takaisin olioksi. [25]

Seuraavassa ohjelmassa 4.4 luodaan uusi Dataset lukemalla *json*-tiedosto tiedostojärjestelmästä. Seuraavaksi luodaan uusi Dataset muunnoksen kautta. Olion kloonaamiseksi käytetään case class :n *copy*-metodia, koska case class -instanssit ovat muuttumattomia ja näin ollen copy-metodi on ainoa tapa kloonata eli luoda uusi muuttumaton (immutable) instanssi siitä. Lopuksi looginen- ja fyysinen suunnitelma tulostetaan konsoliin kutsumalla *explain*-funktiota uudelle Dataset-oliolle.

Ohjelma 4.4 *Dataset-esimerkki*

```
1  val people = spark.read.json("./people.json").as[Person]
2
3  val peopleWithDoubleSalary = people.map { person =>
4    person.copy(salary = person.salary * 2)
5  }
6
7  peopleWithDoubleSalary.explain(true)
```

4.3 DataFrame API

DataFrame on nimettyihin sarakkeisiin järjestetty Dataset. Se on käsitteellisesti yhtenevä relaatiotietokannan taulun tai R/Python kielten tietokehyksen (data frame) kanssa. [28] Alla oleva kuva 4.2 esittää hahmotelman DataFrame:sta.

Kuva 4.2 *DataFrame*

Name: String	Age: Int	Weight: Double
Matt	20	80
George	25	75

DataFrame voidaan rakentaa useammalla tavalla, kuten esimerkiksi jäsennellyistä tiedostoista, hakemalla ja viittaamalla ulkoisista tietokannoista tai olemassaolevista RDD-olioista. Se mahdollistaa datan prosessoinnin kilobittien suuruusluokasta aina petabitteihin asti ja klusterin yksittäisen solmun klustereista suuriin klustereihin. DataFrame-rajapinta on saatavilla Scala-, Java-, Python- ja R-ohjelmointikielille. Kooditasolla Scala-toteutuksessa DataFrame on riveistä rakentuva Dataset. [25] [31]

Alla olevassa ohjelmassa 4.5 luodaan uusi DataFrame tietorakenne kohdistamalla

spark.read -funktio JSON-tiedostoon. Tämän seurauksena Spark pyrkii päättelemään uuden DataFrame:n sisäisen mallin.

Ohjelma 4.5 *DataFrame:n luominen käyttäen read-funktiota*

```
1 val people = spark.read.json("./people.json")
```

DataFrame-API tarjoaa domain-spesifisen kielen (DSL, Domain-Specific Language) jäsennellyn datan, kuten JSON:in, manipulointiin:

Ohjelma 4.6 *Sarakkeen valitseminen ja tulostaminen DataFrame DSL:n avulla*

```
1 people.select("name").show()
```

Yllä olevassa ohjelmassa 4.6 valitaan luodusta DataFrame:sta vain *name* -sarake ja tulostetaan se konsoliin. [28]

4.4 Mllib

MLlib on Sparkin koneoppimiskirjasto. Projektin tavoitteena on tehdä käytännönläheisestä koneoppimisesta skaalautuvaa ja helppoa. MLlib tarjoaa muun muassa seuraavanlaisia palveluita:

- **Koneoppimisalgoritmit** oppimisalgoritmeja kuten luokittelua, regressiota, klusterointia ja yhteisösuodatusta
- **Featurization** piirreirroituis (feature extraction), transformaatiot, dimensioiden vähentäminen (dimensionality reduction) -ja valitseminen
- **Liukuhihnat** työkalut koneoppimisliukuhihnojen rakentamiseen, arviointiin ja muokkaamiseen
- **Persistointi** algoritmien, mallien ja liukuhihnojen tallentaminen ja lataaminen
- **Apuohjelmat** esimerkiksi lineaari algebra, statistiikka ja datan käsittely

Tässä työssä tutustutaan MLlib-kirjaston tarjoamaan ALS-kirjastoon, jolla suositukset tarjoava koneoppimismalli opetetaan. [7]

5. TOTEUTUS

Tässä luvussa esitetään työn toteutuksen oleelliset osat. Opetusdata, sen lataaminen ja siistiminen Sparkia varten. EMR-klusterin pystyttäminen. Projektin rakenne. Mallin opettaminen. Ennustusten kerääminen mallin avulla.

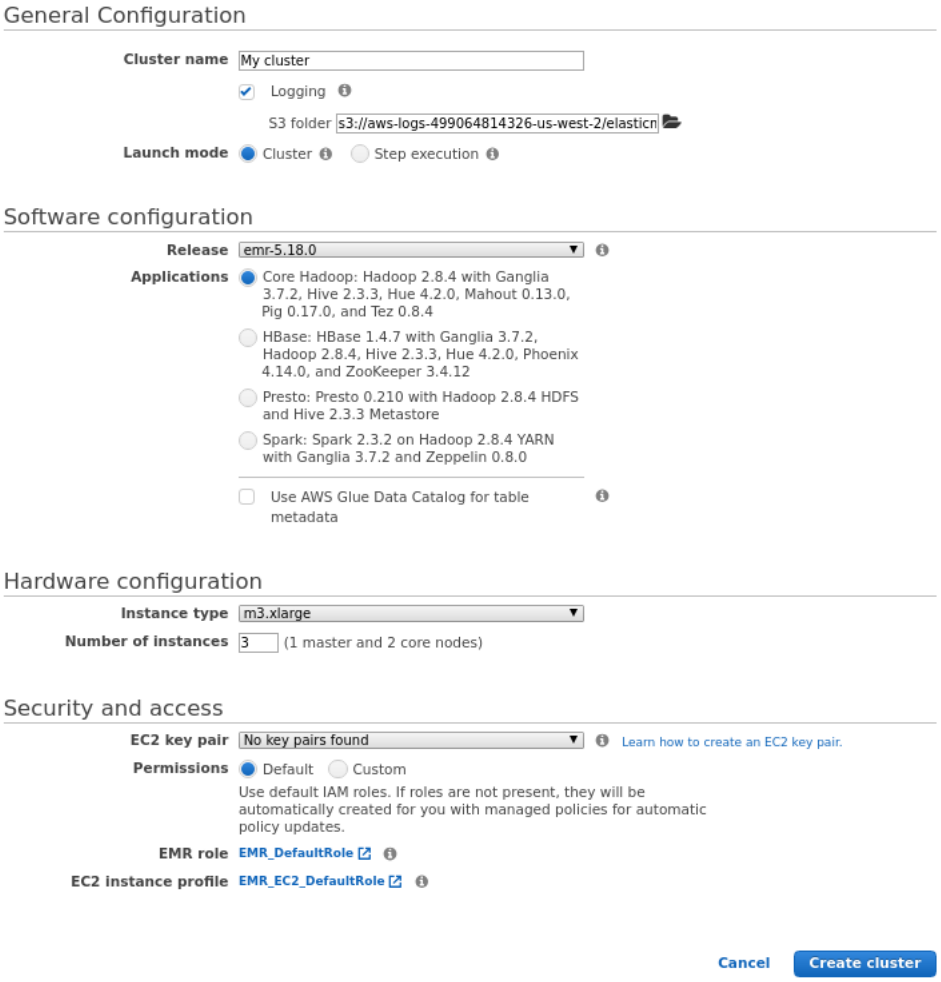
5.1 EMR-klusterin konfigurointi

Tässä osiossa esitetään EMR-klusterin konfigurointi sekä työssä käytetyllä tavalla että vaihtoehtoisesti komentorivin kautta tapahtuvalla keinolla. EMR-klusteri on mahdollista pystyttää myös suoraan Spark-sovelluksesta. Klusterin konfigurointi piti tehdä käytännössä kolmannen osapuolen [15] videon perusteella, sillä AWS:n omat ohjeistukset eivät olleet riittävät, tai ainakin ne koettiin vaikeaselkoisiksi.

5.1.1 WEB-käyttöliittymän avulla

EMR-klusteri voidaan pystyttää kätevästi myös graafisen käyttöliittymän kautta AWS:n konsolissa. Ensin klusterille annetaan nimi. Seuraavaksi voidaan valita, halutaanko tallentaa lokit S3-buckettiin. Spark-sovellus tuotaa yllättävän paljon lokitusta ja tämän valinnan kanssa kannattaakin olla tarkkana, erityisesti jos haluaa operoida Amazon Free Tier:in puitteissa, sillä jokainen kirjoitus syö osansa tästä rajallisesta määrästä. Tämän jälkeen valitaan *Launch mode*, joka tarkoittaa käytännössä sitä, että halutaanko liukuhihna ajaa vain kerran vai useammin. Seuraavaksi valitaan haluttu EMR-versio sekä tarvittavat ohjelmistokomponentit. Tämän työn tarpeisiin tulee valita luonnollisesti Spark. Seuraavaksi valitaan tarvittavien virtuaalikoneiden suorituskyky ja lukumäärä. Viimeinen kohta käsittelee salausta ja pääsynhallintaa. Käyttäjän tulee luoda EC2-avainpari, jotta on mahdollista muodostaa SSH-yhteys klusterin päävirtuaalikoneelle.

Kuva 5.1 *EMR-klusterin luominen*



Yllä olevassa kuvassa 5.1 on esitetty klusterin konfiguroinnin www-käyttöliittymä, johon päästään käsiksi AWS:n konsolin kautta.

5.1.2 Komentorivin avulla

Ohjelma 5.1 *Klusterin luominen komentoriviltä [6]*

```
$ aws emr create-cluster \
  --name "cluster" \
  --release-label emr-5.17.0 \
  --applications Name=Spark \
  --ec2-attributesKeyName=myKey \
  --instance-type m4.large \
  --instance-count 3 \
  --use-default-roles
```

Yllä olevassa ohjelmassa 5.1 pystytetään EMR klusteri AWS:n komentorivityökalun avulla. Rivinvaihtomerkit "\" on lisätty selkeyden takia, joten ne eivät ole merkityksellisiä komennon kannalta. [6] Valinnat omat samat komentoriviversiossa, kuin käyttöliittymänkin kautta.

Ohjelma 5.2 *Sovelluksen ajaminen AWS EMR:ssä*

```
$ sbt package
$ aws s3 cp movielens.jar s3://movielens/movielens.jar
$ aws s3 cp s3://movielens/movielens.jar .
$ spark-submit ./movielens.jar
```

Yllä olevassa ohjelmassa 5.2 esitetään askeleet, joilla sovellus saadaan paketoitua, lähetettyä EC2-virtuaalikoneelle ja ajettua EMR-klusterissa. Ensin ohjelma paketoidaan *jar*-tiedostoksi. Seuraavaksi *jar*-tiedosto lähetetään S3-palveluun, jossa sijaitsee *movielens* niminen bucket. Tämän jälkeen otetaan SSH-yhteys EC2-virtuaalikoneelle ja ladataan *jar*-tiedosto samaisesta bucketista. Lopuksi käytetään *spark-submit* -työkalua sovelluksen ajamiseen EMR-klusterissa.

5.2 Opetusdata

GroupLens Research on kerännyt ja laittanut saataville aineistoja MovieLens-sivustolta. Aineistot on kerätty useiden aikajaksojen aikana, riippuen aineiston koosta. Tässä

työssä käytettävä MovieLens 20M -aineisto sisältää 20 000 000 (kaksikymmentä miljoonaa) arviota, jotka ovat antaneet 138 000 käyttäjää 27 000 elokuvalle. MovieLens 20M -aineisto koostuu *movies.csv* and *ratings.csv* tiedostoista. [12] Taulukoissa 5.1 ja 5.2 on esitetty esimerkit näistä tiedostoista.

Taulukko 5.1 *Näyte movies.csv tiedostosta*

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller

Taulukko 5.2 *Näyte ratings.csv tiedostosta*

userId	movieId	rating	timestamp
1	31	2.5	1260759144
1	1029	3.0	1260759179
1	1061	3.0	1260759182
1	1129	2.0	1260759185
1	1172	4.0	1260759205
1	1263	2.0	1260759151
1	1287	2.0	1260759187
1	1293	2.0	1260759148
1	1339	3.5	1260759125

Toteutuksessa käytettiin RDD-pohjaista rajapintaa, sillä Dataset-pohjainen rajapinta ei ollut vielä täysin toiminnallinen yhteisösuodatuksen ongelmille. Aineiston lataaminen oli mahdollista toteuttaa Dataset-rajapintaa hyödyntäen, mutta varsinainen suositus täytyi tehdä RDD-rajapintaa käyttäen. Dataset-rajapinta tarjoaa useita parannuksia, kuten esimerkiksi yksinkertaisemman tiedon lataamisen. Mikäli työ tehtäisiin alusta, olisi Dataset-rajapinta oikea vaihtoehto toteutukselle.

5.3 Projektin rakenne

Ensimmäinen askel itsenäisen Spark-sovelluksen rakentamisessa on tehdä oikeanlainen kansiorakenne ja luoda tiedosto, jossa kuvaillaan sovelluksen riippuvuudet. Itsenäisellä Spark-sovelluksella tarkoitetaan käyttövalmista *JAR*-tiedostoa (Java ARchive), joka voidaan jakaa Spark-klusterille ja se sisältää sekä koodin että kaikki riippuvuudet. Tätä varten lähdekoodi tulee saada paketoitua ja tämä saavutetaan SBT-työkalulla. SBT (Scala Build Tool) on käännöstyökalu Scala, Java ja C++-kielille, jonka avulla lähdekoodi saadaan sekä käännettyä että paketoitua JAR:iksi [22].
Sovelluksia voidaan ottaa käyttöön klusterissa *spark-submit* työkalun avulla, joka mahdollistaa kaikkien Sparkin tukemien klusterinhallintatyökalujen käyttämisen yhtenäisen käyttöliittymän kautta. Tämä ominaisuus osoittautui erittäin hyödylliseksi kun sovellusta ajettiin EMR-klusterissa, sillä *spark-submit* -työkalu ottaa syötteenä vain käännetyn JAR-tiedoston ja alkaa ajamaan sovellusta.

Ohjelma 5.3 *Sovelluksen käyttöönotto klusterissa*

```
$ spark-submit movielens-recommendations_2.11-1.0.jar
```

Yllä olevassa ohjelmassa 5.3 on esimerkki ohjelman käyttöönotosta Spark-klusterissa.

5.4 Opetusdatan lataaminen Spark sovellukseen

Alla olevassa ohjelmassa 5.4 on esimerkki opetusdatan lataamisesta S3:sta.

Ohjelma 5.4 *Aineiston lataaminen*

```
1  val ratings = sc.textFile("s3n://bucket/ratings.csv")
2    .mapPartitionsWithIndex((i, it) => if (i == 0) it.drop(1) else it)
3    .map { line =>
4      val fields = line.split(",")
5      val userId = fields(0).toInt
6      val movieId = fields(1).toInt
7      val rating = fields(2).toDouble
8
9      Rating(userId, movieId, rating)
10 }
```

Esimerkissä luodaan RDD *ratings* lataamalla csv-tiedosto. Tiedostosta suodatetaan ensin pois otsikkorivit ja tämän jälkeen tiedosto käydään läpi rivi kerrallaan ja pätkitään pilkulla erotetut arvot taulukkoon käyttäen Scalan String-luokan sisäänrakennettua *split* funktiota. Tämän jälkeen taulukossa olevista arvoista muodostetaan *Rating*-olioita. Rating on Sparkin tarjoama apuluokka, jota ei enään käytetä uudemman *ml*-kirjaston kanssa, vaan ALS-malli ottaa syötteenään vain tupleja, joissa on oikeat arvot oikeilla paikoillaan. Huomionarvoista on se, kuinka tiedostoihin voidaan viitata suoraan S3:n tiedoston nimellä ja Spark osaa hakea tiedostot suoraan S3-bucketista. Opetusdataa ei juuri tarvinnut siistiä, sillä opetukseen käytettiin valmista, hyvin jäsenneltyä datasettiä.

5.5 Mallin opettaminen

Alla olevassa ohjelmassa 5.5 on esitetty esimerkki ALS-mallin opettamisesta.

Ohjelma 5.5 *Mallin opettaminen*

```
1  for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
2    val model = ALS.train(training, rank, numIter, lambda)
3    val validationRmse = computeRmse(model, validation, numValidation)
4
5    if (validationRmse < bestValidationRmse) {
6      bestModel = Some(model)
7      bestValidationRmse = validationRmse
8      bestRank = rank
9      bestLambda = lambda
10     bestNumIter = numIter
11   }
12 }
```

Yllä olevassa esimerkissä suoritetaan varsinainen mallin opetus. Opetus tapahtuu niin, että opetetaan muutama versio mallista ja valitaan malleista paras käyttäen RMSE-metriikkaa. Koodin tasolla opetus tapahtuu käyttäen MLlib / ALS kirjaston funktiota *train*, joka ottaa sisääntulonaan *ratings*, *rank*, *iterations* sekä *lambda* parametrit:

- *ratings* on RDD Rating -olioita, jotka sisältävät käyttäjän tunnisteen, elokuvan tunnisteen ja suosituksen

- *rank* on piilevien ominaisuuksien sisällytettävä määrä
- *iterations* on ALS algoritmin iteraatioiden määrä
- *lambda* on regularisaatio-parametri, jolla yritetään ehkäistä mallin ylioppimista

Tutkimuksessa [17] on tutkittu parhaita parametreja ALS-algoritmille ja päädytty lambda-arvoon 0.1 sekä iteraatioiden määrään 20. Parhautta on tutkittu RMSE-metriikan kautta ja kyseisillä parametreilla RMSE saadaan pienimmilleen eli mallin voidaan sanoa sovittuvan parhaiten opetusdataan. Tutkimuksessa oltiin päädytty arvoon 0.819942, kun taas paras itse opetettu malli päätyi RMSE-arvoon 0.807167. Omassa opetuksessa eroavaisuuksina olivat tämän hetken lähin vastaava datasetti, joka ei ollut aivan niin suuri kuin tutkimuksessa käytetty, myös opetusaineistojen suhde oli hieman eri, sillä oman toteutuksen RMSE-validointi tarvitsi myös oman osansa datasta. Validointiin olisi tietysti voitu käyttää samaa opetusdataa, kuten oletettavasti tutkimuksessakin oli tehty tai sitten RMSE oli arvioitu eri menetelmää käyttäen. Tutkimuksessa paras arvo saatiin käyttäen datasettiä 80-20 suhteessa ja omassa opetuksessa käytössä oli datasetti, joka oli jaettu 60-20-20 suhteessa.

5.6 Ennustaminen

Alla olevassa ohjelmassa 5.6 on esitetty esimerkki suositusten ennustamisesta.

Ohjelma 5.6 *Suositusten ennustaminen*

```
1  val myRatedMovieIds = Set(personalRatings.map(_.product))
2  val candidates = movies.keys. filter (!myRatedMovieIds.contains(_)).toSeq
3  val candidatesRDD = sc.parallelize(candidates)
4
5  val recommendations = bestModel
6     .get
7     .predict(candidatesRDD)
8     . collect ()
9     .sortBy(- _.rating)
10    .take(10)
```

Ohjelmassa 5.6 haetaan henkilökohtaiset suositukset käyttämällä mallin *predict*-metodia, joka ottaa parametrinaan mahdollisten elokuvien joukon. Mahdollisilla elokuvilla tarkoitetaan elokuvia, joita käyttäjä ei ole vielä nähnyt, eli ne eivät sisälly *personalRatings*-muuttujan sisältämiin elokuviin. Muuttuja *bestModel* sisältää aiemmin talteen otetun, parhaan RMSE-arvon omaava malli. Tarkemmin sanottuna se on mallin sisältävä *Option* muuttuja, jonka sisällä olevaan malliin päästään käsiksi *get*-funktion avulla.

5.7 Opetusvirheen evaluointi

Alla olevassa ohjelmassa 5.7 esitetään apufunktio *computeRmse*, jolla evaluoidaan opetetun mallin virhettä.

Ohjelma 5.7 *computeRmse-funktio*

```
1  def computeRmse(
2    model: MatrixFactorizationModel,
3    data: RDD[Rating],
4    n: Long
5  ): Double = {
6    val predictions: RDD[Rating] =
7      model.predict(data.map(x => (x.user, x.product)))
8    val predictionsAndRatings = predictions
9      .map(x => ((x.user, x.product), x.rating))
10     .join(data.map(x => ((x.user, x.product), x.rating)))
11     .values
12
13   math.sqrt(
14     predictionsAndRatings
15       .map(x => (x._1 - x._2) * (x._1 - x._2))
16       .reduce(_ + _) / n
17     )
18 }
```

Ohjelmassa 5.7 esitetään apufunktio *computeRmse*, jonka avulla evaluoidaan opetetun mallin virhettä. Apufunktio on ehkä hieman vähäinen nimitys, sillä kuten kappaleessa 2 todettiin, käytetään RMSE:tä kriteerinä sille, että kuinka hyvin malli

sovittuu opetusdataan. Lisäksi vertailevassa tutkimuksessa [17] RMSE-arvoa käytettiin kriteerinä, jolla todettiin paras malli vaihtoehdoista.

6. TULOKSET

Tässä kappaleessa käsitellään työn tuloksia.

6.1 Sisääntulot

Tässä aliluvussa esitetään suosittelujärjestelmälle sisääntulona annetut elokuvat.

Taulukko 6.1 *Itse arvioidut elokuvat*

Tunniste	Nimi	Arvostelu
112897	The Expendables 3 (2014)	4.0
116887	Exodus: Gods and Kings (2014)	4.0
117529	Jurassic World (2015)	4.0
118696	The Hobbit: The Battle of the Five Armies (2014)	4.5
128520	The Wedding Ringer (2015)	4.5
122882	Mad Max: Fury Road (2015)	4.0
122886	Star Wars: Episode VII - The Force Awakens (2015)	4.5
131013	Get Hard (2015)	4.0
132796	San Andreas (2015)	3.0
136305	Sharknado 3: Oh Hell No! (2015)	1.0
136598	Vacation (2015)	4.0
137595	Magic Mike XXL (2015)	1.0
138208	The Walk (2015)	2.0
140523	The Visit (2015)	3.5
146656	Creed (2015)	4.0
148626	The Big Short (2015)	4.5
149532	Marco Polo: One Hundred Eyes (2015)	4.5
150548	Sherlock: The Abominable Bride (2016)	4.5
156609	Neighbors 2: Sorority Rising (2016)	3.5
159093	Now You See Me 2 (2016)	4.0
160271	Central Intelligence (2016)	4.0

Ylläolevassa taulukossa 6.1 on esitetty suosittelujärjestelmälle sisääntuloina annetut, aiemmin nähdyt elokuvat. Sisääntulon rakenne on seuraava: sarakkeessa yksi sijaitsee elokuvan tunniste, sarakkeeseen kaksi on sijoitettu elokuvan nimi ja sarakkeessa kolme sijaitsee elokuvalle annettu arvio asteikolla 0-5. Taulukossa nähtävät arvot ovat vain pieni osa kaikesta opetukseen käytetystä aineistosta.

6.2 Suositukset

Tässä osassa käsitellään suosittelujärjestelmän tarjoamat suositukset, eli työn varsinaiset tulokset.

Taulukko 6.2 *Toteutetun järjestelmän suosittelemat elokuvat*

Numero	Nimi	Tyylilajit
1	The War at Home (1979)	Drama
2	Immagine in Cornice (2007)	Documentary, Music
3	Octopus (2000)	Adventure, Horror
4	My Brother Tom (2001)	Drama
5	Return to the 36th Chamber (1980)	Action, Comedy
6	Bob Funk (2009)	Comedy
7	Hamoun (1990)	Drama
8	Notebook (2006)	Drama, Musical, Romance
9	My Weakness Is Strong (2009)	Documentary, Comedy
10	Deathstalker II (1987)	Adventure, Comedy, Fantasy

Ylläolevassa taulukossa 6.2 on suosittelujärjestelmältä saaduista suosituksista 10 ensimmäistä. Mukaan on lisätty myös elokuvien tyylilajit, jotta suosituksien paikkansapitävyyden arviointi helpottuisi.

Tulokset saadaan kysymällä suosittelijajärjestelmältä suosituksia tietyn käyttäjän tunnisteelle (Tässä tapauksessa työn kirjoittajan). Järjestelmän tulosjoukosta suodatetaan pois ne elokuvat, jotka on nähty, sillä näille suosituksen pitäisi olla erittäin suuri. Tuloksissa ei ole havaittavissa varsinaisia puutoksia. Tuloksien paikaansapitävyyttä voidaan arvioida esimerkiksi tyylilajien perusteella. Elokuvan hyvyys on hyvinkin henkilökohtainen kokemus, eikä siihen oteta kantaa tässä työssä. Tuloksista voidaan havaita että ne täyttävät ainakin yhden aiemmin mainituista suosittelujärjestelmän tehtävistä: yllätyksellisyyden.

7. YHTEENVETO

Tässä kappaleessa esitetään yhteenveto.

7.1 Johtopäätökset

Suosittelujärjestelmän rakentamiseen on olemassa monia mahdollisia toteutusvaihtoehtoja. Apache Spark vaikutti sekä mielenkiintoiselta opiskelukohteelta että hyödylliseltä tulevaisuuden kannalta. Lisäksi Sparkin mukana luonnollisesti tuleva Scala-ohjelmointi vaikutti teknologian valintaan. AWS-palveluiden tuntemus on myös varsin hyödyllinen taito nykyajan ohjelmistokehityksessä ja tämä valinta oli sitä kautta helposti perustelu.

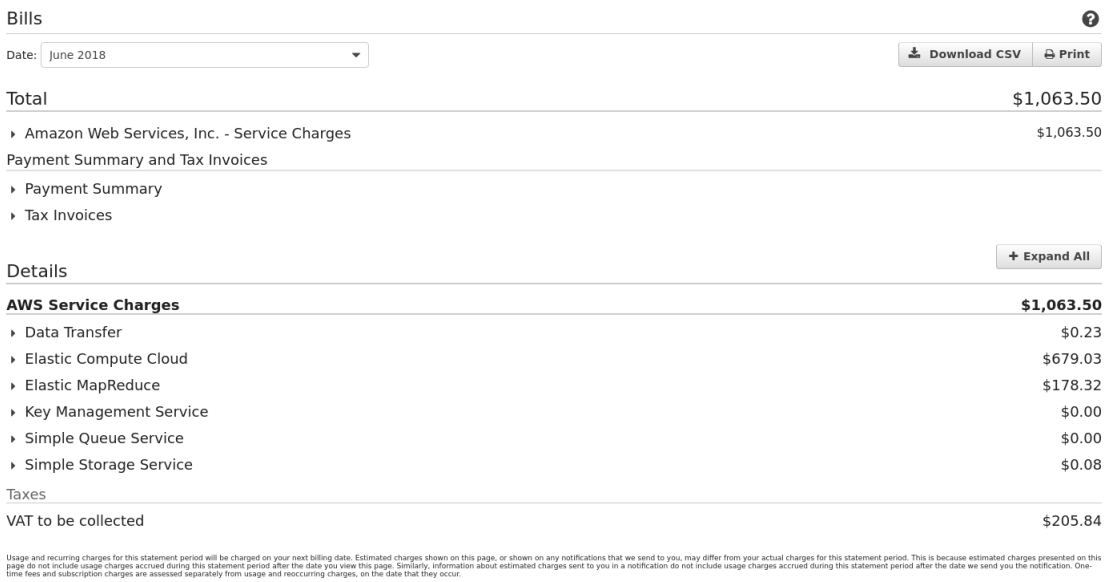
Aluksi EMR-liukuhihnan pystyttämistä yritettiin vain Amazonin käyttöliittymän konsolin kautta, mutta ilman lisäohjeistusta kaikkien valintojen tutkiminen ja ymmärtäminen vaikutti turhan työläältä. Etsinnän jälkeen sopivat ohjeistukset löytyivät opetusvideon muodossa ja EMR saatiin konfiguroitua. Aiemman EMR opiskelun ansiosta videolla mainitut asiat olivat helposti omaksuttavissa, kuten esimerkiksi salaisuuksien (credentials) lisääminen ja EMR:n virtuaalikoneisiin yhdistäminen. Lisäksi aiemmat tiedot Amazon S3:sta sekä yleiset komentorivitaidot olivat hyödyksi. Jälkeenpäin mietittynä sopivan ohjeistuksen etsimiseen käytetyn ajan olisi voinut tietysti myös käyttää EMR:n opiskeluun Amazonin omien ohjeistusten mukaan.

Työssä käytetty EMR-liukuhihna pystytettiin löytyneen video-ohjeen [15] mukaisesti. Pystytettävän palvelun nimi on On-Demand Pipeline, jonka voisi ajatella viittaavan siihen että maksujakin kerrytettäisiin vain palvelua käytettäessä, mutta tosiasiassa varatut EC2-instanssit pysyvät ajossa kellon ympäri. Jatkuvasta maksujen syntymisestä ei varsinaisesti ilmoitettu missään vaiheessa, vaan asiaan havahtui vasta laskun saavuttua. Tämän olisi voinut välttää laittamalla luvussa 5 mainittuun valintaruutuun valinnan, joka olisi sulkenut liukuhihnan ohjelman suorituksen jälkeen. Varsinainen koodin testailu ja integrointi EMR:n kanssa oli kuitenkin inkre-

mentaalista, joten sulkeutunut liukuhihna olisi ollut melkoinen hidaste.

Kolmen EC2 (m3.xlarge) -instanssin kuukauden mittaisesta ajamisesta olisi tullut maksettavaa reilut 1000 dollaria. Laskutus toimii niin, että virtuaalikoneiden ajaminen maksaa 0.266 dollaria/tunti ja tämän lisäksi siihen lisätään 0.07 dollaria/-tunti EMR-lisämaksua. Onneksi asia saatiin selvitettyä asiakaspalvelun kanssa ja AWS-asiakaspalvelun insinöörit tarkistivat lokeista että olin tosiasiassa käyttänyt EMR-liukuhihnaa vain muutaman kerran, kuten kerroinkin. AWS:n asiakaspalvelu ansaitsee kiitosta siitä, kuinka mutkattomasti asia hoitui lopulta: asiakaspalvelijat mitätöivät edellisen kuukauden (kesäkuu 2018) laskun kokonaan ja lisäsivät kuluvalle kuulle (heinäkuu 2018) krediittejä sen verran, että lasku tulisi kuitattua. Jälkeenpäin opiskeltuani lisää siitä, että miten olisin voinut välttää laskun, selvisi että on olemassa erilliset AWS:n palvelut laskujen monitorointiin ja tulevan laskun ennustamiseen. Amazon Cost Explorer mahdollistaa laskujen tutkimisen, Amazon Budget puolestaan valvoo tulevan laskun määrää ja ilmoittaa mikäli ollaan menossa asetetuista rajoista yli. Alla olevassa kuvassa 7.1 on AWS-lasku työhön liittyvien koodien testaamisesta ja suorittamisesta kesäkuulta 2018.

Kuva 7.1 *AWS-lasku*



Parempi tai ainakin edullisempi keino työn toteuttamiseen olisi ollut pystyttää liukuhihna, ajaa tarvittavat eräajot ja tuloksien kirjaamisen jälkeen pysäyttää liukuhihna, jolloin varatut resurssit olisi vapautettu. Huonoa tässä on se, että samaa liukuhihnaa ei ilmeisesti saa uudestaan käyntiin, jolloin ainakin virtuaalikoneella olevat asiat

ovat lopullisesti menetettyjä kun se sulkeutuu. Terminoitu, eli pysäytetty liukuhihna jää kummittelemaan palveluun joksikin aikaa. Aiemmin konfiguroitu liukuhihna on kuitenkin mahdollista kopioida uuden liukuhihnan pohjaksi, joten kerran konfiguroidusta liukuhihnasta on tässä mielessä hyötyä seuraavallakin kerralla.

Parempaa asiakaskokemusta olisi tiedottaa selkästi, että tämä palvelu tosiaan varaa X määrän resursseja jatkuvasti, kunnes se terminoidaan. Muita vaihtoehtoja EMR-liukuhihnan asiakasystävällisempään toimintaan olisivat esimerkiksi:

- EMR-liukuhihna voisi mahdollisesti mennä "nukkumaan", kuten joissakin muissa pilvipalveluissa. Tähän ominaisuuteen tarvittaisiin jonkinlaista logiikkaa tunnistamaan, ettei koneella ole ajossa mitään tähdellistä.
- EMR-liukuhihna voisi mahdollisesti terminoitua kokonaan X ajan käyttämättömyyden takia, tämä ei kuitenkaan olisi käytännöllistä, sillä tulokset katoaisivat virtuaalikoneelta.

Oletusarvoisesti EMR kirjoittaa lokit S3-buckettiin ja tämä saattaa helposti johtaa Free Tier -tilauksen pyyntöjen rajan ylittämiseen. AWS lähetti pyyntöjen hupenemisesta onneksi viestin ja näin ollen päästiin ratkomaan kyseistä konfiguraatiovirhettä. EMR-klusteria ei ole mahdollista muokata jälkeenpäin, joten se täytyy poistaa (terminate), jotta lokien kirjoittaminen saatiin poistettua käytöstä. Amazon Free Tier sisältää S3:n osalta 2000 luku/kirjoitus -kutsua kuukaudessa ja koska kaikki Spark-sovelluksen tuottamat lokit kirjoitetaan niin tämä raja tuli vastaan nopeasti.

Saadut tulokset eivät ole palveluiden, kuten Netflix, tasolla, mutta ei sitä varmaan kannattanut odottaakaan. Mielenkiintoista oli se, kuinka "huonoilta" saadut suositukset vaikuttivat. Yllättäviä ja uusia kylläkin, mutta mikään elokuvista ei kuulosta soveliaalta tai mielekkäältä. Tässä tosin voikin piillä juuri hyvän suosittelun raja, sillä luultavasti ihmisen muodostama mielipide vaikkapa pelkän nimen perusteella saattaa johtaa elokuvan hylkäämiseen. Ihminen ei välttämättä ole täysin objektiivinen valitsemaan sitä, onko jokin suositeltu elokuva katsomisen arvoinen. Elokuvan julkaisuvuosi, ohjaajan tunnettuus, näyttelijät ja jopa kansikuva herättävät mielipiteitä, jotka saattavat johtaa elokuvan hylkäämiseen tai ainakin siirtämiseen sille kuuluisalle "katson joskus" -listalle.

Suuremman aineiston käyttäminen sekä isomman arvostelumäärän tarjoaminen järjestelmälle voisi ajatella parantavan tuloksia. Voisi olettaa että tarjotut 20 elokuvan

arvostelua eivät vielä riittäneet siihen, että järjestelmä muodostaisi kovin kokonaisvaltaista kuvaa arvosteluiden antajan elokuvamausta.

7.2 Tulevaa työtä

Uudempaa ML-kirjastoa olisi mielenkiintoista tutkia, sillä Dataset-rajapintaa voidaan nyt käyttää yhteisösuodatuksen ongelmien ratkomiseen. Toteutusta yritettiin alunperin myös Dataset-rajapintaa hyväksikäyttäen, mutta kaikki ominaisuudet eivät olleet tuolloin vielä käytössä. Työtä aloittaessa Spark oli versiossa 1.5/1.6 ja työtä lopettaessa versiossa 2.3. MLlib-kirjasto on vaihtunut ML nimiseen kirjastoon ja tässä yhteydessä myös ohjelmointimalli on vaihtunut RDD-perusteisesta rajapinnasta Dataset-perusteiseen rajapintaan.

Yhteisösuodatusta voidaan tietenkin käyttää muuhunkin tarkoitukseen kuin elokuvien suositteluun, kuten esimerkiksi kirjojen. Olisikin mielenkiintoista tutkia myös jotain muuta ongelmaa ja soveltaa siihen ALS-algoritmia.

Verratun tutkimuksen [17] mukaista mallia olisi mahdollista tutkia lisää, jos selviäisi keino, jolla RMSE:tä arvioitiin. Pitäisi siis esimerkiksi selvittää, oliko kyseisen tutkimuksen 60-40 suhteessa jaetun aineiston testidatasta osa käytetty validointiin kuten omassa toteutuksessa tehtiin, jossa aineisto jaettiin 60-20-20 suhteessa. Mikäli AWS-resursseja olisi rajattomasti käytössä, niin olisi myös melko suoraviivaista tutkia useampaakin kombinaatiota mallin parametreille.

Työssä mainituista teknologioista esimerkiksi *spark-submit* -työkalua voitaisiin tutkia lisää. Olisi mielenkiintoista ymmärtää, kuinka klusterinhallinta toimii ja miten Spark-tehtäviä (Spark Job) jaetaan eri solmuille. Nykyisessä toteutuksessa on oletettu, että *spark-submit* etsii tarvittavat parametrit automaattisesti EMR-klusterista, sillä niitä ei annettu eksplisiittisesti. Tämä etsintä voitaisiin toteuttaa esimerkiksi ympäristömuuttujien avulla. Kuten aliluvussa 2.2.1 mainittiin, EMR-klusteri on Hadoop-klusteri ja *spark-submit* työkalun avainominaisuutena pidetään klusterinhallinnan helpottumista. Sparkin hajautetun ohjelmointimallin kannalta olisi myös mielenkiintoista tutkia sulkeumia (closure) ja erityisesti sitä, kuinka ne toimivat oikeassa klusterissa. Sulkeuma on funktionaalisissa ohjelmointikielissä käytetty ominaisuus, mikä tarkoittaa funktion kykyä viitata leksikaalisen näkyvyysalueensa parametreihin.

Eräs mahdollinen lähestymistapa mallin hyödyntämiseen olisi toteuttaa vain mallin

kouluttaminen AWS:ssä ja tämän jälkeen ladata koulutettu malli johonkin paikalliseen järjestelmään. Kuten aiemmin todettiin, Spark-sovellukseen on mahdollista ladata ennakkoon koulutettu malli. Tulosten kysyminen valmiiksi koulutetulta mallilta onnistuu varmasti resursseiltaan pienemmällä laitteella kuin varsinainen mallin kouluttaminen. Mallia voitaisiin ehkä käyttää hyväksi myös jopa matkapuhelimissa tai jopa joissain tehokkaimmissa ja käyttöjärjestelmältään sekä kirjastoiltaan soveliaissa sulautetuissa laitteissa.

Olisi myös mielenkiintoista tutkia kuinka suurella omien arvosteluiden määrällä suositukset alkaisivat olla parempia. Lähestymistapa tässä voisi olla esimerkiksi sellainen, että jätettäisiin joitakin omia suosikkielokuvia tarkoituksella arvostelematta ja tarkkailtaisiin missä kohdassa järjestelmä rupeaisi ehdottamaan juuri näitä elokuvia.

KIRJALLISUUTTA

[1] C. Aberger, Recommender: An Analysis of Collaborative Filtering Techniques, Personal and Ubiquitous Computing Journal, 2014.

[2] C. C. Aggarwal, Recommender Systems, Springer International Publishing, 2016.

[3] Amazon, Amazon EMR (Elastic Map Reduce). Saatavissa (viitattu 24.6.2018): https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html

[4] Amazon, Amazon S3 (Simple Storage Service), 2018. Saatavissa (viitattu 24.6.2018): https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html

[5] Amazon, AWS (Amazon Web Services), 2018. Saatavissa (viitattu 24.6.2018): https://aws.amazon.com/

[6] Amazon, Create a Cluster With Spark, 2018. Saatavissa (viitattu 24.6.2018): https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-launch.html

[7] Apache Software Foundation, Machine Learning Library (MLlib) Guide. Saatavissa (viitattu 2.10.2018): https://spark.apache.org/docs/latest/ml-guide.html

[8] BookLens. Saatavissa (viitattu 23.8.2017): https://booklens.umn.edu/

[9] R. Burke, Hybrid Recommender Systems: Survey and Experiments, User Modeling and User-Adapted Interaction, vsk. 12, nro 4, 2002, ISSN 1573-1391, s. 331–370.

[10] D. Eppstein, Directed Acyclic Graph. Saatavissa (viitattu 13.5.2017): https://en.wikipedia.org/wiki/Directed_acyclic_graph#/media/File:Topological_Ordering.svg

[11] S. K. Gorakala, M. Usuelli, Building a Recommendation Engine with R, first p., Packt Publishing, 2015.

[12] GroupLens Research, MovieLens. Saatavissa (viitattu 23.8.2017): `https://movielens.org/info/about`

[13] M. Guller, Big Data Analytics with Spark, first p., Apress, 2015.

[14] Y. Koren, R. Bell, C. Volinsky, Matrix Factorization Techniques For Recommender Systems, Computer Society, 2009.

[15] Level Up, Run Spark Application(Java) on Amazon EMR (Elastic MapReduce) cluster, 2017. Saatavissa (viitattu 19.6.2018): `https://www.youtube.com/watch?v=hSWkKk36TS8`

[16] G. Linden, B. Smith, J. York, Amazon.com Recommendations, IEEE INTERNET COMPUTING, 2003, ISSN 1089-7801, s. 76–79.

[17] G. Miryala, R. Gomes, K. R. Dayananda, COMPARATIVE ANALYSIS OF MOVIE RECOMMENDATION SYSTEM USING COLLABORATIVE FILTERING IN SPARK ENGINE. Saatavissa (viitattu 3.4.2018): `http://www.jgrcs.info/index.php/jgrcs/article/view/1015/644`

[18] Oracle, JAR File Overview. Saatavissa (viitattu 25.9.2018): `https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html`

[19] Oracle, The Java Virtual Machine Specification. Saatavissa (viitattu 25.9.2018): `https://docs.oracle.com/javase/specs/jvms/se10/html/index.html`

[20] F. Ricci, L. Rokach, B. Shapira, P. B. Kanto, Recommender Systems Handbook, first p., Springer, 2011.

[21] S. Ryza, U. Laserson, S. Owen, J. Wills, Advanced Analytics with Spark, O'Reilly Media, Inc., 2015.

[22] S. SBT, The Interactive Build Tool, 2015. Saatavissa (viitattu 25.9.2018): `https://www.scala-sbt.org/1.x/docs/index.html`

[23] Spark, ALS, 2014. Saatavissa (viitattu 13.3.2017): `http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS`

[24] Spark, Collaborative Filtering - RDD-based API, 2014. Saatavissa (viitattu 13.3.2017): `http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html`

[25] Spark, Spark Programming Guide, 2014. Saatavissa (viitattu 13.3.2017): http://spark.apache.org/docs/latest/programming-guide.html

[26] Spark, Dataset, 2016. Saatavissa (viitattu 13.3.2017): https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/sql/Dataset.html

[27] Spark, RDD, 2016. Saatavissa (viitattu 13.3.2017): https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.rdd.RDD

[28] Spark, Spark SQL Programming Guide, 2016. Saatavissa (viitattu 13.3.2017): http://spark.apache.org/docs/latest/sql-programming-guide.html

[29] P. A. Stavrou, What is the difference between convex and non-convex optimization problems?, 2013. Saatavissa (viitattu 10.5.2017): https://www.researchgate.net/post/What_is_the_difference_between_convex_and_non-convex_optimization_problems

[30] M. Technologies, Apache Spark. Saatavissa (viitattu 13.3.2017): https://mapr.com/products/product-overview/apache-spark/

[31] TutorialsPoint, Spark SQL - DataFrames, 2018. Saatavissa (viitattu 29.10.2018): https://www.tutorialspoint.com/spark_sql/spark_sql_dataframes.htm