TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JONNE PETTERI PIHLANEN

SUOSITTELIJAJÄRJESTELMÄN RAKENTAMINEN APACHE SPARKILLA

Diplomityö

Examiner: ????
Examiner and topic approved by the
Faculty Council of the Faculty of
xxxx
on 1st September 2014

ABSTRACT

JONNE PETTERI PIHLANEN: Building a Recommendation Engine with Apache Spark
Tampere University of Technology
Diplomityö, xx pages
September 2016
Master's Degree Program in Signal Processing
Major: Data Engineering
Examiner: ????
Keywords:

The amount of recommendation engines around the Internet is constantly growing.

This paper studies the usage of Apache Spark when building a recommendation engine.

TIIVISTELMÄ

JONNE PETTERI PIHLANEN: Building a Recommendation Engine with Apache Spark
Tampereen teknillinen yliopisto
Diplomityö, xx sivua
syyskuu 2016
Signaalinkäsittelyn koulutusohjelma
Pääaine: Data Engineering
Tarkastajat: ????
Avainsanat:

PREFACE

Thanks to my wife, Noora, for pushing me forward with the thesis when my own interest was completely gone. Without you this would never have been ready.

Tampere,

Jonne Pihlanen

SISÄLLYS

LIST OF ABBREVIATIONS AND SYMBOLS

Spark Fast and general engine for large-scale data processing

Information retrieval (IR) Activity of obtaining relevant information resources from a collection of information resources.

1. JOHDANTO

Suosittelujärjestelmiä on menestyksellisesti käytetty auttamaan asiakkaita päätöksenteossa. Itse asiassa ne ovat jatkuvasti läsnä jokapäiväisessä elämässämme. Mikäli asiakas tekee ostoksia, katsoo elokuvaa Netflixistä, selailee Facebookia tai yksinkertaisesti lukee vain uutisia. Periaatteessa kaikki elämämme osa-alueet sisältävät jonkinlaista suosittelua. Ihmiset voivat kuitenkin tehokkaasti suositella vain niitä asioita, jotka ovat itse henkiläkohtaisesti kokeneet. Tälläin suosittelijajärjestelmistä tulee hyädyllisiä, sillä ne voivat mahdollisesti tarjota suosituksia tuhansista erilaisista tuotteista.

Suosittelu voidaan jakaa kahteen pääkategoriaan: tuotepohjaiseen ja käyttäjäpohjaiseen suositteluun. Tuotepohjaisesa suosittelussa tarkoituksena on etsiä samankaltaisia tuotteita, sillä käyttäjä saattaa haluta mieluummin samankaltaisia tuotteita myäs tulevaisuudessa. Käyttäjäpohjaisessa suosittelussa käyttäjän ajatellaan olevan kiinnostunut tuotteista, joita samankaltaiset käyttäjät ovat ostaneet. Käyttäjäpohjainen suosittelu yrittää siis etsiä samankaltaisia käyttäjiä, jotta voidaan suositella näiden käyttäjien ostamia tuotteita.

Apache Spark on sovelluskehys, joka mahdollistaa hajautettujen ohjelmien rakentamisen. Hajautettu ohjelma tarkoittaa, että ohjelman suoritus on jaettu useiden käsittelysolmujen kesken. Suositteluongelma voidaan mallintaa hajautettuna sovelluksena, jossa kaksi matriisia, käyttäjät ja tuotteet, prosessoidaan iteratiivisella algoritmilla, joka mahdollistaa ohjelman suorittamisen rinnakkain.

Apache Spark on rakennettu Scala ohjelmointikielellä. Scala on monikäyttäinen, moniparadigmainen ohjelmointikieli, joka tarjoaa tuen funktionaaliselle ohjelmoinnille sekä vahvan tyypityksen. Tyässä käytetään Scalaa, joten lyhyt johdanto ohjelmointikieleen tarjotaan.

Tämä työ on rakentuu seuraavista osista. Luku kaksi kuvailee suosittelujärjestelmiä. Luvussa kolme keskutellaan Apache Sparkista, avoimen lähdekoodin järjestel-

mästä, joka mahdollistaa hajautettujen ohjelmien rakentamisen. Luku neljä esittää toteutuksen suosittelijajärjestelmälle. Luvussa viisi käydään läpi tulokset. Lopuksi luvussa kuusi esitellään johtopäätäkset.

2. SUOSITTELIJAJÄRJESTELMÄT

Suosittelulla tarkoitetaan tehtävää, jossa tuotteita suositellaan käyttäjille. Kaikista yksinkertaisin suosittelun keino on vertaisten kesken, täysin 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 tuhansista erilaisista tuotteista. Suosittelijajärjestelmät ovat joukko tekniikoita ja ohjelmistoja jotka tarjovat suosituksia mahdollisesti hyödyllisistä tuotteista. Tuote tarkoittaa yleistä asiaa jota järjestelmä suosittelee henkilölle. Suosittelujärjestelmät on yleensä tarkoitettu suosittelemaan tietyn tyyppisiä tuotteita kuten kirjoja tai elokuvia. [10]

Suosittelijajärjestelmien tarkoitus on auttaa asiakkaita päätöksenteossa kun tuotteiden määrä on 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. (VIITE?) Räätälöimätöntä suosittelua on esimerkiksi yksinkertainen top 10 lista. Järjestäminen tehdään ennustamalla kaikista sopivimmat tuotteet käyttäjän mieltymysten tai vaatimusten perusteella. Tämän suorittamiseksi suosittelijajärjestelmän on kerättävä käyttäjältä tämän mieltymykset. Mieltymykset voivat olla nimenomaisesti ilmaistuja tuotearvioita tai ne voidaan tulkita käyttäjän toiminnasta kuten klikkauksista tai sivun katselukerroista. Suosittelijajärjestelmä voisi esimerkiksi tulkita tuotesivulle navigoinnin todisteeksi mieltymyksestä sivun tuotteista. [10]

Suosittelijajärjestelmien kehitys alkoi melko yksinkertaisesta havainnosta: ihmiset tapaavat luottaa toisten suosituksiin tehdessään rutiininomaisia päätöksiä. On esimerkiksi yleistä luottaa vertaispalautteeseen valitessaan kirjaa luettavaksi tai luottaa elokuvakriitikoiden kirjoittamiin arvioihin. Ensimmäinen suosittelijajärjestelmä yritti matkia tätä käytöstä käyttämällä algoritmeja suosituksien löytämiseen yhteisöstä aktiiviselle käyttäjälle, joka etsi suosituksia. Tämä lähestymistapa on olennaisesti yhteistyösuodattamista ja idea sen takana on että jos käyttäjät pitivät saman-

kaltaisista tuotteista aikaisemmin, he luultavasti pitävät samoja tuotteita ostaneiden henkilöiden suosituksia merkityksellisinä. [10]

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. Hyödyn tuottamisen sijaan vaihtoehtojen määrä oli alkanut heikentää kuluttajien hyvinvointia. Vaihtoehdot ovat hyväksi, mutta vaihtoehtojen lisääntyminen ei ole aina parempi. [10]

Viimeaikoina suosittelijajärjestelmät ovat osoittautuneet tehokkaaksi lääkkeeksi käsillä olevaa 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 suosittelijan tietokannasta. 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. [10]

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. Esimerkkinä suuren profiilin verkkosivustot kuten Amazon.com, YouTube, Netflix sekä IMDB, joissa suosittelujärjestelmillä on iso rooli. Oma lukunsa ovat myös vain suosittelujärjestelmien tutkimiseen ja kehittämiseen tarkoitetut konferenssit kuten RecSys ja AI Communications (2008). [10]

Suosittelujärjestelmällä voidaan ajatella olevan kaksi päätarkoitusta. Ensimmäinen on avustaa palveluntarjoajaa. Toinen on tuottaa arvoa palvelun käyttäjälle. Suosittelujärjestelmän on siis tasapainoteltava sekä palveluntarjoajan että käyttäjän tarpeiden välillä. [10] Palveluntarjoaja voi esimerkiksi ottaa suosittelujärjestelmän avuksi parantamaan tai monipuolistamaan myyntiä, lisäämään käyttäjien tyytyväisyyttä, lisäämään käyttäjien uskollisuutta tai ymmärtämään paremmin mitä käyttäjä haluaa [10]. Lisäksi käyttäjillä saattaa olla seuraavanlaisia odotuksia suosittelujärjestelmältä. Käyttäjä saattaa haluta suosituksena tuotesarjan, apua selaamiseen

tai mahdollistaa muihin vaikuttamisen. Vaikuttaminen saattaa olla pahantahtoista. [10]

GroupLens, BookLens ja MovieLens olivat uranuurtajia suosittelujärjestelmissä. GroupLens on tutkimuslaboratorio tietojenkäsittelyopin ja tekniikan laitoksella Minnesotan Yliopistossa, Twin Cities:issä, joka on erikoistunut suosittelujärjestelmiin, verkkoyhteisöihin, mobiili ja kaikkialla läsnä oleviin teknologioihin, digitaalisiin kirjastoihin ja paikallisen maantieteen tietojärjestelmiin. [2] BookLens on on GroupLensin rakentama kirjojen suosittelujärjestelmä [3]. MovieLens on GroupLensin ylläpitämä elokuvien suosittelujärjestelmä [9]. Uranuurtavan tutkimuksen lisäksi nämä sivustot julkaisivat aineistoja, joka ei ollut yleistä. [2]

2.1 Suositustekniikat

Suosittelujärjestelmällä täytyy olla jonkunlainen ymmärrys tuotteista, jotta se pystyy suosittelemaan jotain. Tämän mahdollistamiseksi, järjestelmän täytyy pystyä ennustamaan tuotteen käytännöllisyys tai ainakin verrata tuotteiden hyödyllisyyttä ja tämän perusteella päättää suositeltavat tuotteet. Ennustamista voidaan luonnostella yksinkertaisella ei-personoidulla 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. [10]

Tuotteen i hyödyllisyyttä käyttäjälle u voidaan mallintaa reaaliarvoisella funktiolla $R(u, i)$, kuten yleensä tehdään yhteisösuodatuksessa ottamalla huomioon käyttäjien antamat arviot tuotteista. Yhteisösuodatus suosittelijan perustehtävä on ennustaa R:n arvoa käyttäjä-tuote pareille ja laskea arvio todelliselle funktiolle R. Laskiessaan tätä ennustetta käyttäjälle u 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. [10]

Suosittelijajärjestelmät eroavat toisistaan kohdistetun toimialan, käytetyn tiedon ja erityisesti siinä kuinka suositukset tehdään, jolla tarkoitetaan suosittelualgoritmia [10]. Tässä työssä keskitytään vain yhteen suosittelutekniikoiden luokkkaan, yhtei-

sölliseen suodatukseen, sillä tätä menetelmää käytetään Apache Sparkin MLlib kirjastossa.

Yhteisöllistä suodatusta käyttävät suosittelijajärjestelmät perustuvat käyttäjien yhteistyöhön. Niiden tavoitteena on tunnistaa kuvioita käyttäjän mielenkiinnoista voidakseen tehdä suunnattuja suosituksia [1]. Tämän lähestymistavan alkuperäisessä toteutuksessa suositellaan aktiiviselle käyttäjälle niitä tuotteita joita muut samankaltaiset mieltymyksen omaavat käyttäjät ovat pitäneet aiemmin [10]. 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. Yhteisösuodatusta on eniten käytetty ja toteutettu tekniikka suositusjärjestelmissä [6] [10] [4].

Yhteisöllinen suodatus analysoi käyttäjien välisiä suhteita ja tuotteiden välisiä riippuvuuksia tunnistaakseen uusia käyttäjä-tuote assosiaatioita [7]. Päätelmä siitä, että käyttäjät voisivat pitää samasta laulusta koska molemmat kuuntelevat muita samankaltaisia lauluja on esimerkki yhteisöllisestä suodatuksesta [11].

Koska yhteisöllisessä suodatuksessa suosittelu perustuu pelkästään käyttäjän arvosteluihin tuotteesta, yhteisöllinen suodatus kärsii ongelmista jotka tunnetaan nimillä uusi käyttäjäongelma ja uusi tuoteongelma [6]. Ellei käyttäjä ole arvostellut yhtään tuotetta, algoritmi ei kykene tuottamaan myöskään yhtään suositusta. Muita yhteisöllisen suodatuksen 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]

2.1.1 Muistiperustainen yhteisöllinen suodatus

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

Seuraavat kappaleet kuvaavat käyttäjäpohjaista yhteisösuodatusta ja tuotepohjaista

yhteisösuodatusta.

Tuotepohjainen yhteisösuodatus

Tuotepohjainen yhteisösuodatus (Item-based collaborative filtering, IBCF) aloittaa etsimällä samankaltaisia tuotteita käyttäjän ostoshistoriasta [6]. Seuraavaksi mallinnetaan käyttäjän mieltymykset tuotteelle perustuen saman käyttäjän tekemiin arvosteluihin [10]. Alla oleva koodinpätkä esittelee ICBF:n idean jokaiselle uudelle käyttäjälle.

Program 2.1 *Tuotepohjainen yhteisösuodatus algoritmi [6]*

```
1. For each two items, measure how similar they are in terms of having received
    similar ratings by similar users

1. Jokaiselle kahdelle tuotteelle, mittaa kuinka samankaltaisia ne ovat sen
    suhteen, kuinka samankaltaisia arvioita ne ovat saaneet samankaltaisilta
    käyttäjiltä.

val similarItems = items.foreach { item1 =>
        items.foreach { item2 =>
                val similarity = cosineSimilarity(item1, item2);
        }
}

2. For each item, identify the k−most similar items

2. Tunnistaa k samankaltaisinta tuotetta jokaiselle tuotteelle

val itemsSorted = sort(similarItems)

3. For each user, identify the items that are most similar to the user's
    purchases

3. Jokaiselle käyttäjälle, tunnista tuotteet jotka ovat eniten samankaltaisia
    käyttäjän ostoshistorian kanssa.
```

```
users.foreach { user =>
        user.purchases.foreach { purchase =>
                val mostSimilar = findSimilarItem(purchase)
        }
}
```

Amazon.com:in, Amerikan suurin internetkauppa, on aiemmin tiedetty käyttävät 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. [8]

Käyttäjäpohjainen yhteisösuodatus

Tuotepohjainen yhteisösuodatus (User-based collaborative filtering, UBCF) 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. [10]

Askeleet jokaiselle uudelle käyttäjälle käyttäjäpohjaisessa yhteisösuodatuksessa ovat:

Program 2.2 *Käyttäjäpohjainen yhteisösuodatus algoritmi [6]*

```
1. Mittaa jokaisen käyttäjän samankaltaisuus uuteen käyttäjään. Kuten IBCF:ssä
    , suosittuja samankaltaisuusarvioita ovat korrelaatio sekä kosinimitta.

case class Similarity(userId1: Int, userId2: Int, score: Int)

val newUser: User = User("Adam", 31, purchases)
val similarities = users.map { user =>
        Similarity(newUser.id, user.id, cosineSimilarity(user, newUser)
}
```

```
2. Tunnista samankaltaisimmat käyttäjät. Vaihtoehtoja on kaksi: Voidaan valita
    joko parhaat k käyttäjää (k−nearest neighbors) tai voidaan valita käyttäjät,
    joiden samankaltaisuus ylittää tietyn kynnysarvon.

val mostSimilarUsers = similarities. filter (_.score > 0.8)

3. Arvostele samankaltaisimpien käyttäjien ostamat tuotteet. Arvostelu saadaan
    joko keskiarvona kaikista tai painotettuna keskiarvona, käyttäen
    samankaltaisuuksia painoina.

val ratedItems = mostSimilarUsers.map { user =>
        user.purchases.map { purchase =>
                val purchases = mostSimilarUsers.map { usr =>
                        usr.purchases. filter (_.id === purchase.id)
                }
        purchases.sum() / purchases.size
        }
}

4. Valitse parhaiten arvostellut tuotteet.

val topRatedItems = ratedItems.take(10)
```

2.1.2 Mallipohjainen yhteisösuodatus

Muistipohjaiseen 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. [10]

Vaihtelevat pienimmät neliöt (Alternating Least Squares, ALS) algoritmi on esimerkki mallipohjaisesta yhteisösuodatusalgoritmista ja se esitetään seuraavassa lu-

vussa.

3. APACHE SPARK

Apache Spark on avoimen lähdekoodin sovelluskehys, joka yhdistää hajautettujen ohjelmien kirjoittamiseen tarkoitetun järjestelmän sekä elegantin mallin ohjelmien kirjoittamiseen. [11] Spark tarjoaa korkean tason rajapinnat Java, Scala, Python sekä R ohjelmointikielille.

Korkealla tasolla, jokainen Spark sovelus koostuu ajaja (driver) ohjelmasta sekä yhdestä tai useammasta täytäntöönpanijasta (executor). Ajaja on ohjelma, joka ajaa käyttäjän pääohjelmaa ja suorittaa erilaisia rinnakkaisia operaatioita klusterissa. Täytäntöönpanija on yksi kone klusterissa.

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.

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 putkiston seuraavalle vaiheelle. Toiseksi, Apache Spark kohtelee kehittäjiä paremmin tarjoamalla rikkaan joukon muunnoksia (transformations) joiden avulla voidaan muutamalla koodirivillä ilmaista monimutkaisia putkistoja. (ESIMERKKI?) 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. [11]

Spark ohjelmia voidaan kirjottaa Java, Scala, Python tai R ohjelmointikielellä. Scalan käyttämisellä saavutetaan kuitenkin muutamia etuja, joita muut kielet eivät tarjoa. Tehokkuus paranee, sillä tehtävät kuten datan siirtäminen eri kerrosten välillä tai muunnosten suorittaminen datalle saattaa johtaa heikompaan tehokkuuteen. Spark on kirjoitettu Scala-ohjelmointikielellä, joten viimeisimmät ja parhaimmat ominaisuudet ovat aina käytössä. 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. [11]

Spark-jakelun mukana toimitetaan luku-evaluointi-tulostus-silmukka, komentorivityökalu, (Read eval print loop, REPL), joka mahdollistaa uusien asioiden nopean testailun konsolissa, eikä sovelluksista tarvitse rakentaa itsenäisiä (self-contained) alusta asti. Usein kun REPLissä kehitetty sovelluksen tai sovelluksen osan voidaan katsoa olevan tarpeeksi valmis, on järkevää tehdä siitä koottu kirjasto (JAR). Näin varmistutaan etteivät koodi tai tulokset pääse katoamaan, vaikkakin REPL tarjoaa samantapaisen muistin komentohistoriasta kuin perinteinen komentorivikin.

SELITYS JAR:ISTA JA EHKÄ JVM:STÄ?

3.1 Resilient Distributed Dataset (RDD)

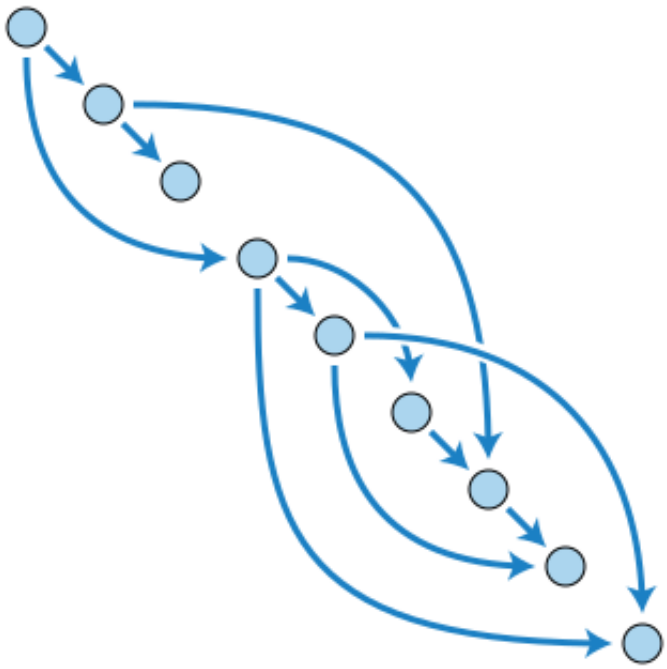
Resilient Distributed Dataset (RDD) on Sparkin tarjoama pääabstraktio. RDD on muuttumaton, osioitu elementtikokoelma joka voidaan hajauttaa klusterin useiden koneiden välillä. [16]

Tärkeä yksityiskohta ymmärtää RDD:stä on että ne ovat laiskasti evaluoituvia. Laiska evaluaatio (lazy evaluation) on evaluointi taktiikka, jossa lausekkeen evaluointia viivytetään siihen asti kun sen arvoa tarvitaan. Kun uusi RDD luodaan, mitään ei oikeasti vielä tapahdu. Spark tietää missä data sijaitsee tai miten data saadaan laskettua kun tulee aika tehdä sille jotain.

RDD voidaan luoda kahdella tavalla. Olemassaoleva Scala kokoelma voidaan rinnakkaistaa (parallelize). Toinen keino on viitata ulkoiseen aineistoon ulkoisessa varastointijärjestelmässä kuten HDFS:sä, HBase:ssa tai missä tahansa Hadoopin tuntemassa tiedostojärjestelmässä. [14]

RDD:t voidaan tallentaa muistiin, jolloin ohjelmistokehittäjä voi uudelleenkäyttää niitä tehokkaasti rinnakkaisissa operaatioissa. RDD:t voivat palautua solmuvirheistä automaattisesti käyttäen Directed Acyclic Graph (DAD) moottoria. DAG tukee syklistä 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. [18]

Kuva 3.1 *Directed Acyclic Graph [5]*



3.2 Dataset API

Dataset (DS) on RDD:n korvaaja Sparkissa. DS on vahvasti tyypitetty kokoelma aluespesifisiä objekteja jotka voidaan muuntaa rinnakkain käyttäen funktionaalisia tai relaatio-operaatioita. Dataset:ille olemassa olevat operaatiot on jaettu muunnoksiin ja toimiin. Muunnokset ovat operaatioita, jotka luovat uusia Datasettejä, kuten map, filter, select, aggregate. Toimet ovat operaatioita jotka laukaisevat laskentaa ja palauttavat tuloksia. Toimia ovat esimerkiksi count, show tai datan kirjoittaminen tiedostojärjestelmään. [15]

Dataset-instanssit ovat laiskoja luonteeltaan, jolla tarkoitetaan sitä, että laskenta aloitetaan vasta kun toimintoa kutsutaan. 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 optimoitu fyysistä suunnitelmaa, voidaan tutkia käyttämällä DS:n *explain* funktiota. [15]

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. Enkooderia voidaan käyttää esimerkiksi luokan *Person*, joka sisältää kentät nimi (merkkijono) ja ikä (kokonaisluku), kertomaan Sparkille generoi koodia ajon aikana joka serialisoi *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. [15]

Two ways typically exist to create a Dataset. The most common way is to make use of the read function provided by SparkSession and point Spark to some files on the storage system. Such as the following *json* file.

Dataset voidaan luoda tyypillisesti kahdella eri tavalla. Yleisin tapa on käyttää *SparkSession*:in tarjoamaa *read* funktiota ja osoittaa Spark joihinkin tiedostoihin tiedostojärjestelmässä, kuten seuraavaan *json* tiedostoon.

Program 3.1 *Esimerkki JSON tiedosto*

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

Program 3.2 *Uuden Dataset olion luominen käyttäen read funktiota*

```
val people = spark.read.json("./people.json").as[Person]
```

where *Person* would be a Scala case class, for example:

jossa *Person* olisi Scala case class, esimerkiksi:

Program 3.3 *case class Person*

```
case class Person(id: BigInt, firstName: String, lastName:
    String)
```

Case classes are normal Scala classes that are:

- Immutable by default
- Decomposable through pattern matching
- Compared by structural equality instead of by reference
- Succinct to instantiate and operate on

If we would omit the casting with keyword *as* then we would end up creating a DataFrame and the schema of the created object would be guessed by Spark.

Program 3.4 *Creating a SparkSession*

```
val spark = SparkSession
  .builder
  .appName("MovieLensALS")
  .config("spark.executor.memory", "2g")
  .getOrCreate()
```

SparkSession is the entry point to programming Spark with the Dataset and DataFrame API. In the above snippet we create a *SparkSession* by chaining calls to the builder method, which creates a SparkSession. Builder object for constructing a SparkSession. Datasets can also be created through transformations available on existing Datasets:

Program 3.5 *Creating a new Dataset through a transformation*

```
val names = people.map(_.name)
```

[15]

Datasets are similar to RDDs as they also provide strong typing and the ability to use powerful lambda functions [17]. These are accompanied with the benefits of Spark SQL's optimized execution engine [17]. However, instead of using standard serialization like Java serialization they use a specialized Encoder to serialize the objects. Serialization denotes a task in which an object is turned into bytes thus reducing the memory footprint of the object. In general, serialization is needed for processing or transmitting over the network. While both encoders and standard serialization are responsible for turning an object into bytes, encoders are generated dynamically by code. They are using a format that allows Spark to perform many operations such as filtering, sorting and hashing without the need of deserializing the bytes back into an object. [14]

In the following listing, we create a new Dataset by reading a *json* file from the file system. Next we create another Dataset through a transformation. We make use of the *copy* method of Scala case class to clone an object, since we had declared the *people* Dataset to be immutable. Finally, we display the physical plan by running the *explain* function on the newly created Dataset.

Program 3.6 *Displaying the logical and physical plan of a Dataset*

```
val people = spark.read.json("./people.json").as[Person]
val peopleWithDoubleSalary = people.map { person =>
        person.copy(salary = person.salary * 2)
}
peopleWithDoubleSalary.explain
```

```
== Physical Plan ==
*SerializeFromObject [staticinvoke(class org.apache.spark.
    unsafe.types.UTF8String, StringType, fromString,
    assertnotnull(input[0, $line62.$read$$iw$$iw$Person, true
    ], top level Product input object).name, true) AS name#
```

```
    212, staticinvoke(class org.apache.spark.sql.types.
    Decimal$, DecimalType(38,0), apply, assertnotnull(input
    [0, $line62.$read$$iw$$iw$Person, true], top level
    Product input object).salary, true) AS salary#213]
+- *MapElements <function1>, obj#211: $line62.
    $read$$iw$$iw$Person
+- *DeserializeToObject newInstance(class $line62.
    $read$$iw$$iw$Person), obj#210: $line62.
    $read$$iw$$iw$Person
+- *FileScan json [name#200,salary#201L] Batched: false,
    Format: JSON, Location: InMemoryFileIndex[file:/home/
    joonne/Documents/GitHub/thesis-code/people.json],
    PartitionFilters: [], PushedFilters: [], ReadSchema:
    struct<name:string,salary:bigint>
```

3.3 DataFrame API

A DataFrame is essentially a Dataset that is organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R or Python, but it has richer optimizations under the hood. DataFrames can be constructed from a range of sources such as structured data files, tables in Hive, external databases, or existing RDDs. The DataFrame API is available in Scala, Java, Python, and R. In the Scala API a DataFrame is represented by a Dataset of Rows, it essentially is simply a type alias of Dataset[Row]. [17]

Program 3.7 *Creating a new DataFrame by using read function*

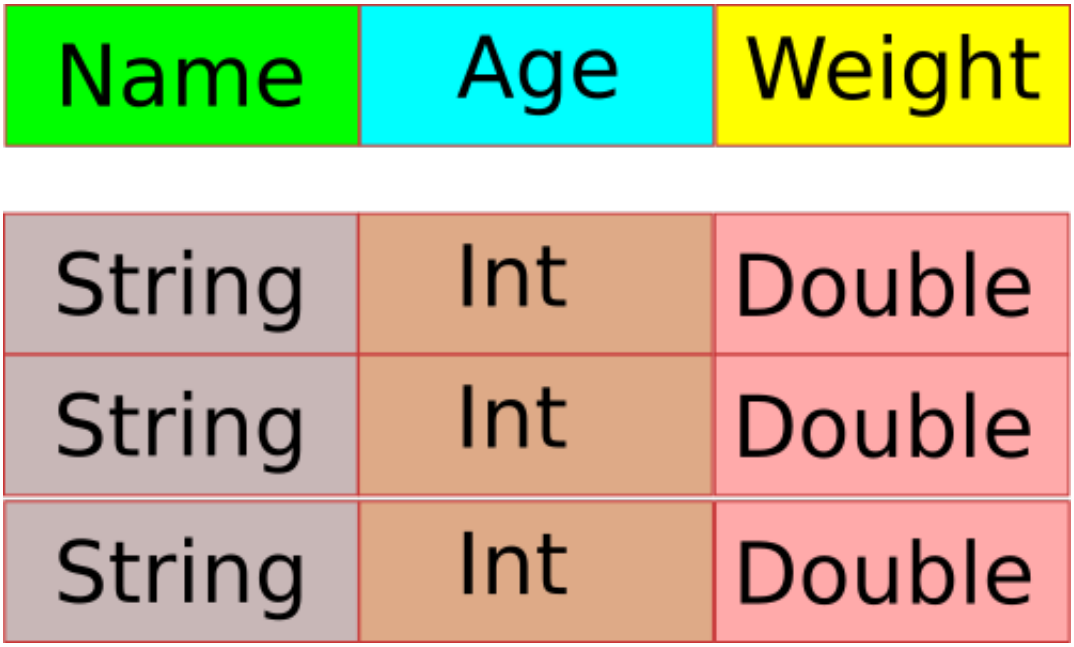
```
val people = spark.read.json("./people.json")
```

When creating a DataFrame, the schema of the

3.4 Matrix Factorization

Matrix factorization denotes a task in which a matrix is decomposed into a product of matrices. There are many different matrix decompositions. The following chapter

Kuva 3.2 *DataFrame*



will describe matrix factorization in general and the Alternating Least Squares algorithm which is the matrix factorization algorithm that is implemented in Spark. It is based on same idea as Netflix prize winner, matrix factorization models.

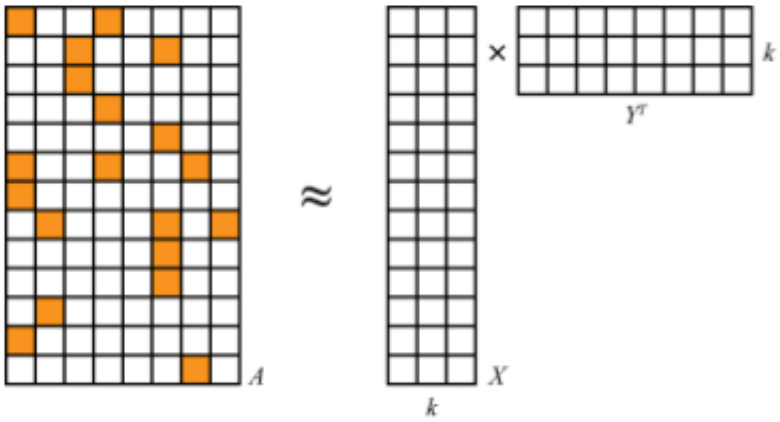
Matrix factorization belongs to a vast class of algorithms called latent-factor models. Latent-factor models try to explain observed interactions between a large number of users and products through a relatively small number of unobserved, underlying reasons. For example, they can try to explain why people would buy a particular album out of endless possibilities by describing users and albums in terms of tastes which are not directly available as data. [11] A latent factor is not available for direct observation. For example health of a human being is a latent factor. Health can not be observed as a variable such as blood pressure.

Matrix factorization algorithms treat the user and product data as if it was a large matrix A. Each entry in row i and column j represents a rating the user has given to a specific product. [11]

Usually A is sparse, which denotes that most of the entries of A are 0. This is due to the fact that usually only a few of all the possible user-product combinations exist.

Matrix factorization models factor A as the matrix product of two smaller matrices,

Kuva 3.3 *Matrix factorization [11]*



X and Y, which are quite tiny. Since A has many rows and columns, both of them have many rows, but both have just a few columns (k). The k columns match to the latent factors that are being used to explain the interactions of the data. The factorization can only be approximate because k is small. [11]

The standard approach to matrix factorization based collaborative filtering treats the entries in the user-product matrix as explicit preferences given by the user to the product, for example users giving ratings to movies. Implicit data denotes for example page views or a value representing if a user has listened to a artist. Explicit data means actual ratings that a user has given to a product. Spark ALS can handle both implicit and explicit data. [13] [11]

Usually many real-world use cases have access only to implicit feedback data such as views, clicks, purchases, likes or shares. However, instead of trying to model the matrix of ratings directly, the approach in Spark MLlib treats the data as numbers representing the strength of the observations such as the number of clicks, or the cumulative duration someone spent viewing a movie. Instead of explicit ratings, these numbers are related to the level of confidence in observed user preferences. Based on this data, the model tries to find latent factors that can be used to predict the expected preference of a user for an item. [13]

Sometimes these algorithms are referred to as matrix completion algorithms. This is because the original matrix A may be sparse while the product XY^T is dense. Hence, the product is only an approximation of A. [11]

3.4.1 Alternating Least Squares (ALS)

Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. Spark MLlib currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. Spark MLlib uses the Alternating Least Squares (ALS) algorithm to learn these latent factors. [13]

Spark ALS attempts to estimate the ratings matrix A as the product of two lower-rank matrices, X and Y. [12]

$$A = XY^T \tag{3.1}$$

Typically these approximations are referred to as factor matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix. [12] Spark ALS enables massive parallelization since it can be done separately, it can be done in parallel which is an excellent feature for a large-scale computation algorithm. [11]

Spark ALS is a blocked implementation of the ALS factorization algorithm. Idea is to group the two sets of factors, referred to as *users* and *products*, into blocks. Grouping is followed by reducing communication by only sending one copy of each user vector to each product block on each iteration. Only those user feature vectors are sent that are needed by the the product blocks. Reduced communication is achieved by precomputing some information about the ratings matrix to determine the out-links of each user and in-links of each product. Out-link denotes those blocks of products that the user will contribute to. In-link refers to the feature vectors that each product receives from each user block they depend on. This allows to send only an array of feature vectors between each user block and product block. Consequently the product block will find the users' ratings and update the products based on these messages. [12]

Essentially, instead of finding the low-rank approximations to the rating matrix A, it finds the approximations for a preference matrix P where the elements of P are 1 when $r > 0$ and 0 when $r <= 0$. The ratings then act as confidence values related

to strength of indicated user preferences rather than explicit ratings given to items. [12]

$$A_i Y (Y^T Y)^{-1} = X_i \tag{3.2}$$

Alternating Least Squares operates by rotating between fixing one of the unknowns u_i or v_j. While the other is fixed the other can be computed by solving the least-squares problem. This approach is useful because it turns the previous non-convex problem into a quadratic that can be solved optimally [1]. A general description of the algorithm for ALS for collaborative filtering taken from [1] is as follows:

Program 3.8 *Alternating Least Squares algorithm [1]*

1. Initialize matrix V by assigning the average rating **for** that movie as the first row, and small random numbers **for** the remaining entries.

2. Fix V, solve U by minimizing the RMSE function.

3. Fix U, solve V by minimizing the RMSE function.

4. Repeat Steps 2 and 3 until convergence.

Minimizing the Root Mean Square Error RMSE function denotes a task in which line is plotted. EXPLAIN RMSE.

BIBLIOGRAPHY

[1] C. Aberger, "Recommender: An analysis of collaborative filtering techniques," 2014. [Online]. Available: http://cs229.stanford.edu/proj2014/Christopher%20Aberger,%20Recommender.pdf

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

[3] BookLens. [Online]. Available: https://booklens.umn.edu/

[4] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370. [Online]. Available: http://dx.doi.org/10.1023/A:1021240730564

[5] D. Eppstein. Directed acyclic graph. [Online]. Available: https://en.wikipedia.org/wiki/Directed_acyclic_graph#/media/File:Topological_Ordering.svg

[6] S. K. Gorakala and M. Usuelli, *Building a Recommendation Engine with R*, 1st ed. Packt Publishing, 2015.

[7] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," 2009. [Online]. Available: https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

[8] G. Linden, B. Smith, and J. York, "Amazon.com recommendations," *IEEE INTERNET COMPUTING*, pp. 76–79, 2003. [Online]. Available: http://www.cin.ufpe.br/~idal/rs/Amazon-Recommendations.pdf

[9] MovieLens. [Online]. Available: https://movielens.org/info/about

[10] F. Ricci, L. Rokach, B. Shapira, and P. B. Kanto, *Recommender Systems Handbook*, 1st ed. Springer, 2011.

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

[12] Spark. (2014) ALS. [Online]. Available: http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS

[13] ——. (2014) Collaborative filtering - rdd-based api. [Online]. Available: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

[14] ——. (2014) Spark programming guide. [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html

[15] ——. (2016) Dataset. [Online]. Available: https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/sql/Dataset.html

[16] ——. (2016) Rdd. [Online]. Available: https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.rdd.RDD

[17] ——. (2016) Spark sql programming guide. [Online]. Available: http://spark.apache.org/docs/latest/sql-programming-guide.html

[18] M. Technologies. Apache spark. [Online]. Available: https://mapr.com/products/product-overview/apache-spark/