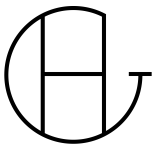


HoGent

Faculteit Bedrijf en Organisatie

Serverless cloudplatformen voor Big Data: een vergelijkende studie

Thomas Aelbrecht

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Chantal Teerlinck
Co-promotor:
Jan Huyzentruyt

Instelling: IntoData

Academiejaar: 2017-2018

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Serverless cloudplatformen voor Big Data: een vergelijkende studie

Thomas Aelbrecht

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Chantal Teerlinck
Co-promotor:
Jan Huyzentruyt

Instelling: IntoData

Academiejaar: 2017-2018

Tweede examenperiode

Woord vooraf

Deze bachelorproef werd geschreven in het kader van het voltooien van de opleiding Toegepaste Informatica afstudeerrichting Mobile Apps. Dit onderwerp interesseert mij vanwege de uitdaging om aan de slag te kunnen gaan op de grote cloudplatformen van Google en Amazon met daarbij een gigantische hoeveelheid aan data. Ik ben al langer gefascineerd door alles wat met Big Data te maken heeft en dit onderwerp was de uitstekende kans om hiermee aan de slag te gaan.

Deze scriptie zou niet tot stand gekomen zijn zonder de hulp van verscheidenen mensen. In wat volgt uit ik dan ook een oprechte dankuwel voor al wie geholpen heeft tijdens de uitvoering van dit onderzoek.

In de eerste plaats wil ik mijn co-promotor, Jan Huyzentruyt, bedanken voor de hulp tijdens de opzet van deze bachelorproef en voor alle nuttige feedback onderweg. Hij stond altijd klaar om mij te bellen, zelfs 's avonds. Ook al waren er dingen waar we beiden niets van kenden, toch ging hij steeds tot het uiterste om mij te helpen met het uitwerken van dit onderzoek. Daarnaast wil ik ook een enorme bedankt uitbrengen voor IntoData voor de financiële steun bij dit toch niet zo goedkope onderzoek.

Ook wil ik mijn promotor, Chantal Teerlinck, bedanken voor de feedback op de inhoud van mijn bachelorproef en voor het beantwoorden van mijn vele vragen over het proces en de invulling van deze bachelorproef.

Ook wil ik mijn ouders bedanken voor de financiële en mentale steun om deze opleiding tot een goed einde te brengen.

Ik wens u veel leesplezier toe!

Samenvatting

Dit onderzoek kan dienen als basis voor de keuze tussen verschillende cloud cluster-computing platformen. Dit omdat een datawarehouse praktisch onmisbaar is voor een hedendaags bedrijf. Dit onderzoek focust zich vooral op het opslaan, analyseren en verwerken van de verzamelde data. Tijdens dit onderzoek wordt zowel een Amazon AWS Spark cluster als Google BigQuery onderzocht en met elkaar vergeleken. Daarnaast wordt er ook een blik geworpen op een eventueel alternatief voor Amazon EMR: Google Cloud Dataproc. Bij dit onderzoek worden een aantal queries uitgevoerd op een dataset over boetes in New York. Daarnaast wordt elke cloudoplossing geanalyseerd o.b.v. enkele requirements opgesteld door IntoData. Bij deze analyse wordt o.a. de performantie, leercurve en kost van de gebruikte tools en technologieën vergeleken. In dit document vindt u een inleiding tot het onderwerp met o.a. de huidige stand van zaken. Daarnaast vindt u per cloudoplossing de resultaten van de queries en de scores op de requirements. Uit dit onderzoek komt naar voor dat Google BigQuery de absolute winnaar is op het gebied van verwerking van BigData in de cloud, op de voet gevolgd door Google Cloud Dataproc en uiteindelijk Amazon EMR. Daarnaast blijkt ook dat elk van deze oplossingen zijn specifieke use cases heeft. Toekomstig onderzoek kan zijn naar de verschillend in performantie tussen Google Cloud Dataproc en Amazon EMR, naar gelijkwaardige alternatieven voor Google BigQuery of naar de verschillende integratiemogelijkheden bij deze drie oplossingen.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

> "Without big data analytics, companies are blind and deaf, wandering out onto the web like deer on a freeway." – Geoffrey Moore, management consultant en theoreticus

De quote van Geoffrey Moore, management consultant en theoreticus, illustreert hoe belangrijk data is voor bedrijven. Data staat namelijk vaak centraal in de bedrijfsvoering. De hoeveelheid data wordt ook elke dag alsmaar groter waardoor deze niet geschikt is voor traditionele SQL databanken. Bedrijven moeten steeds elders op zoek naar een goede manier om hun gigantische hoeveelheden aan data te verwerken. Daarbij willen bedrijven hun klanten steeds een stapje voor zijn maar daarvoor moet men niet alleen een duidelijk beeld hebben van de noden en trends die binnen deze markt doorbreken. Men moet eveneens, in een vroeg stadium, inzicht hebben in de nieuwste technologieën en tools. Vanuit dit idee kwam IntoData met de vraag voor dit onderzoek. Men zou graag inzicht krijgen in de nieuwste cloudplatformen voor de verwerking van Big Data.

Wanneer we naar het huidige landschap van Big Data cloudplatformen kijken, vallen meteen drie grote families op: Azure, Google en Amazon. Deze bedrijven bieden alledrie een heel ecosysteem aan tools aan die het bedrijven makkelijker moet maken om met hun continue stroom aan data relevante informatie en inzichten te verwerven. Ook zorgen deze platformen ervoor dat de bedrijven zich niet veel zorgen hoeven te maken over de infrastructuur nodig voor opslag en verwerking van de data, de platformen zorgen hier zelf voor. Hieronder wordt per familie een overzicht gegeven van de mogelijkheden qua cloudplatformen op het gebied van Big Data verwerking met een korte omschrijving.

Amazon

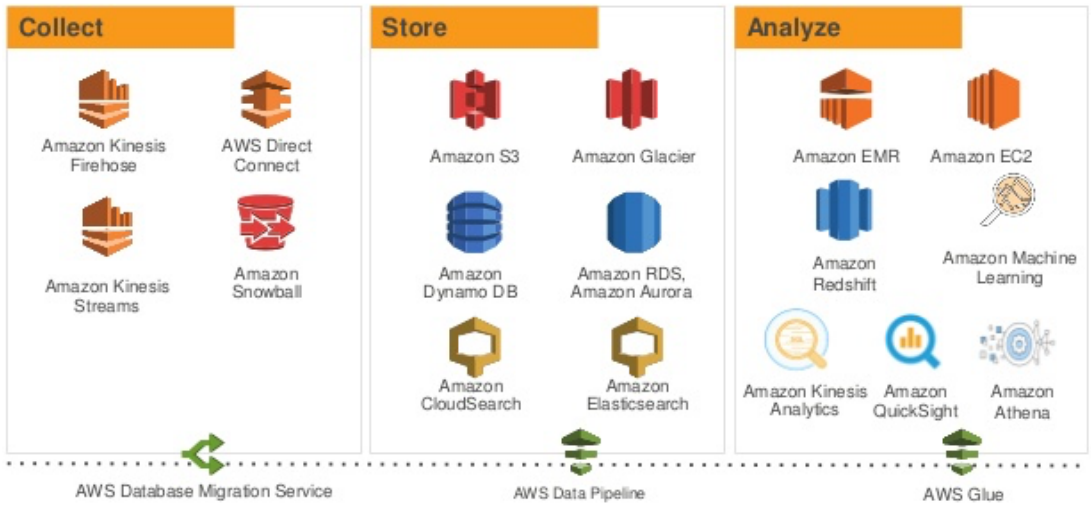
Meer informatie over de platformen uit onderstaande lijst is te vinden op de overzichtspagina van Amazon Analytics. Onderstaande tools en hun onderlinge relaties worden overzichtelijk weergegeven op Figuur 1.1.

1. Collect (verzamelen)
    - AWS Direct Connect: makkelijk een netwerkconnectie opzetten naar AWS
    - Amazon Kinesis: verzamel, verwerk en analyseer data streams
    - Amazon Snowball: migreer en transporteer petabytes aan data in en uit AWS
2. Store (persisteren)
    - Amazon CloudSearch: eenvoudig een zoekfunctie voor een website of applicatie opzetten, onderhouden en schalen
    - Amazon Data Pipeline: helpt met het verwerken en verplaatsen van data tussen verschillende AWS services
    - Amazon DynamoDB: snelle en flexibele NoSQL database
    - Amazon Elasticsearch Service: deploy, gebruik en schaal eenvoudig Elasticsearch
    - Amazon Glacier: opslag voor het archiveren van data op lange termijn
    - Amazon RDS: volledig beheerd relationele databankservice met keuze uit zes populaire databanken (Aurora, PostgreSQL, MySQL, MariaDB, Oracle en Microsoft SQL Server)
    - Amazon S3: dataopslag voor eender welk soort bestandstype en vanop eender welke locatie beschikbaar
3. Analyze (analyseren)
    - Amazon Athena: data uit Amazon S3 analyseren gebruik makend van standaard SQL
    - Amazon EMR: biedt een volledig beheerd Hadoop framework waarin grote hoeveelheden data snel en kostenefficiënt uitgevoerd kunnen worden. Amazon EMR ondersteunt o.a. Apache Spark, HBase, Presto en Flink.
    - Amazon Glue: data voorbereiden en laden in verschillende soorten dataopslaglocaties
    - Amazon Machine Learning: machine learning voor elke developer en data scientist
    - Amazon QuickSight: cloudgebaseerde business analytics (Business Intelligence)
    - Amazon Redshift: data warehouse dat een eenvoudige integratie biedt met je reeds bestaande business intelligence tools

Google

Meer informatie over de platformen uit onderstaande lijst is te vinden op de overzichtspagina van Google Cloud Big Data. Onderstaande tools worden overzichtelijk weergegeven op Figuur 1.2.

- BigQuery: volledig beheerbaar en grootschalig data warehouse



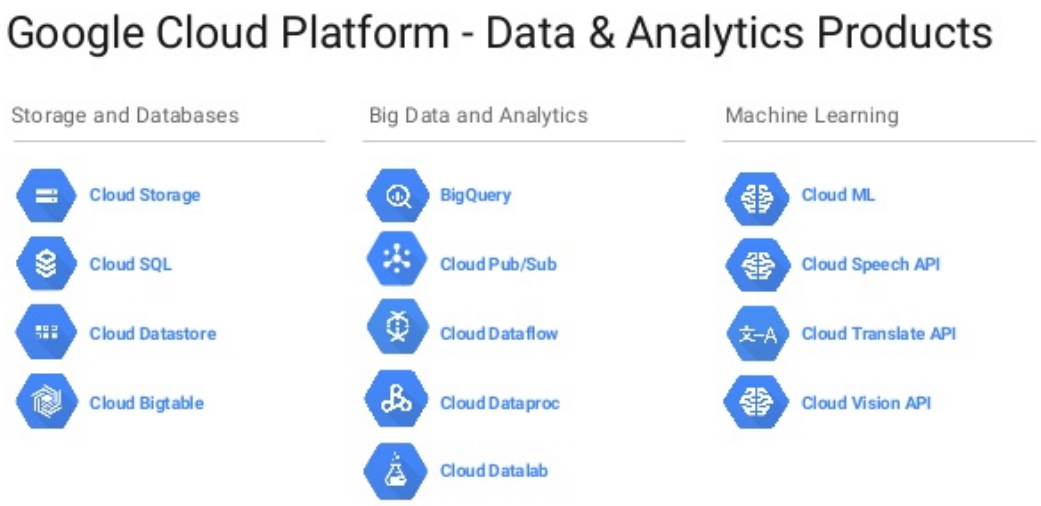
Figuur 1.1: Amazon overzicht (Amazon, 2017c)

- Cloud Dataflow: real-time verwerken van data in batch of streams
- Cloud Dataproc: Spark en Hadoop services
- Cloud Datalab: onderzoeken, analyseren en visualiseren van grote datatsets
- Cloud Dataprep (bèta): onderzoeken, opruimen en voorbereiden van data voor analyses
- Cloud Pub/Sub: event streams van overal en eender welke grootte beheren
- Genomics: verwerk grote hoeveelheden genoomdata
- Google Data Studio (bèta): verwerking en visualisatie van data voor business beslissingen

Microsoft Azure

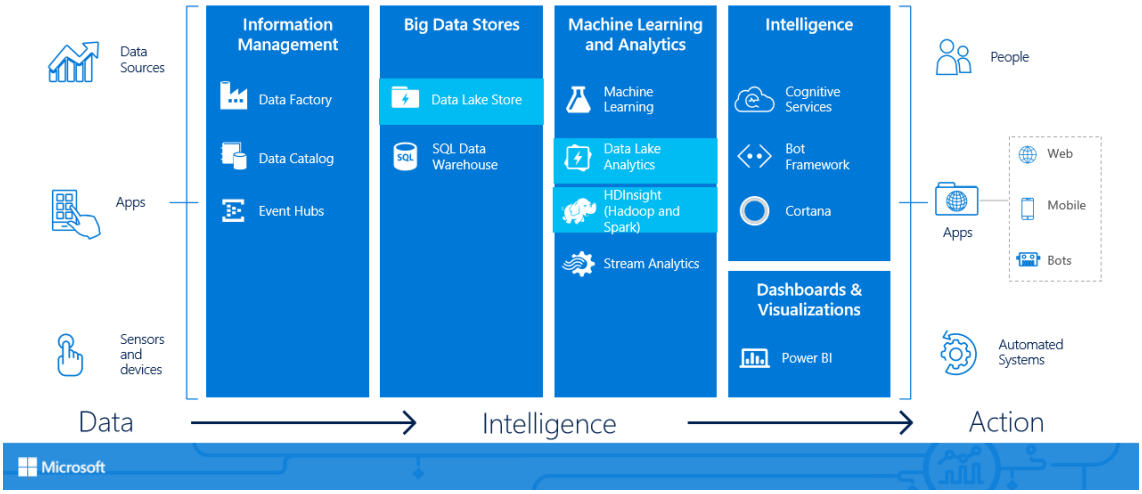
Meer informatie over de platformen uit onderstaande lijst is te vinden op de overzichtspagina van Azure Analytics. De meeste van onderstaande tools en hun onderlinge relaties worden overzichtelijk weergegeven op Figuur 1.3 (een aantal tools zijn niet aanwezig op onderstaand schema).

- Data Lake Analytics: gedistribueerde analytische services
- HDInsight: opzetten van Hadoop, Spark, R Server, HBase en Storm clusters
- Stream Analytics: real-time verwerken van data streams van IoT toestellen
- Azure Databricks: analytisch platform gebaseerd op Apache Spark
- Azure Analysis Services: analytic engine as a service voor bedrijven
- Event Hubs: Ontvang parameters van een groot aantal toestellen
- Data Factory: hybride data-integratie voor bedrijven
- SQL Data Warehouse: elastisch data warehouse as a service met speciale features voor bedrijven



Figuur 1.2: Google Cloud Platform overzicht (Jang, 2016)

- Log Analytics: verzamel, zoek en visualiseer data
- Data Lake Store: hyperscale repository (gigantische opslagplaats) voor Big Data analytische gegevens
- Power BI Embedded: voeg volledig interactieve data visualisaties toe aan je applicaties



Figuur 1.3: Azure overzicht (Kshirsagar, 2016)

Bovenstaande lijsten tonen aan dat het aanbod aan cloudplatformen gigantisch is. Daarom heeft IntoData, in het kader van dit onderzoek, ervoor gekozen om enkel de focus te leggen op het ecosysteem van Google en Amazon, meer bepaald op Amazon EMR, Google BigQuery en Google Cloud Dataproc. In Hoofdstuk 2 vindt u meer informatie over de interne werking van deze drie oplossingen.

1.1 Probleemstelling

In veel bedrijven staat data centraal. Belangrijk daarbij is:

- Data structuring (relationeel, dimensioneel modelleren)
- Data integratie (ETL-ELT, data movement, migration, replication) voor operationele systemen of datawarehouses
- Data quality en master data management (MDM)
- Data delivery (visualisatie, rapportering, BI / BA)
- Data governance

In veel bedrijven wordt dagelijks een enorme hoeveelheid aan data gegenereerd en dan rijst vaak de vraag hoe men deze data zo snel en efficiënt mogelijk kan analyseren. Het is duidelijk dat traditionele SQL-databanken hiervoor niet altijd een oplossing zijn, dit hangt sterk samen met de omvang van de data. De dataservicesmarkt evolueert razendsnel. Een bedrijf wil vaak de klant steeds een stapje voor zijn, maar daarvoor moet men niet alleen een duidelijk beeld hebben van de noden en trends die binnen deze markt doorbreken. Men moet eveneens, in een vroeg stadium, inzicht hebben in de nieuwste technologieën en tools. Dat is net wat dit onderzoek te bieden heeft. Zoals eerder aangegeven focust dit onderzoek op het vergelijken van Amazon EMR, Google BigQuery en Google Cloud Dataproc.

1.2 Onderzoeksvraag

1.2.1 Hoofdonderzoeksvragen

Zoals reeds aangegeven in Sectie 1.1 zal dit onderzoek vooral focussen op het vergelijken van de mogelijkheden van Amazon EMR, Google Cloud Dataproc en Google BigQuery, daaruit volgt dan ook de volgende hoofdonderzoeksvraag.

- Wanneer moet een bedrijf Amazon EMR verkiezen boven Google BigQuery?

Op deze onderzoeksvraag zal dit onderzoek een antwoord geven in de conclusie (zie Hoofdstuk 8).

1.2.2 Deelonderzoeksvragen

Daarnaast zijn er ook nog enkele ondersteunende deelonderzoeksvragen die bij dit onderzoek horen.

- Welke functionaliteiten en technische eigenschappen hebben Amazon EMR, Google Cloud Dataproc en Google BigQuery?
- Welke architectuur zit er onder Amazon EMR?
- Welke architectuur zit er onder Google Cloud Dataproc?
- Welke architectuur zit er onder Google BigQuery?
- Wat zijn de voor- en nadelen van deze drie oplossingen?

- Kan Google Cloud Dataproc een alternief zijn voor een Amazon EMR Apache Spark cluster?
- Hebben deze oplosssingen wel degelijk organisaties iets te bieden?

Op deze deelonderzoeksvragen zal gaandeweg door dit onderzoek een antwoord gegeven worden. Een samenvatting hiervan is te vinden in de conclusie (zie Hoofdstuk 8).

1.3 Onderzoeksdoelstelling

Het hoofddoel van dit onderzoek is een degelijk inzicht bieden in drie mogelijke cloud-oplossingen voor de verwerking van Big Data en deze in het juiste perspectief te zetten. Een tweede doel van dit onderzoekt is een duidelijk en eenduidig antwoord te geven op de hoofdonderzoeksvraag zodat bedrijven goed weten wanneer men voor welke oplossing moet kiezen. Dit onderzoek is dan ook geslaagd wanneer dit doel behaald wordt. Een derde doel van dit onderzoek is om bedrijven, die dit lezen, de mogelijkheid bieden om een stapje voor te zijn op hun concurrenten.

1.4 Hypothese

Alvorens dit onderzoek zal starten wordt eerst een hypothese opgesteld op basis van informatie verzameld tijdens het opmaken van het bachelorproefvoorstel, te vinden in Bijlage A. De eerste hypothese die dit onderzoek zal proberen te bewijzen of te ontkrachten is dat Google BigQuery sneller is dan Amazon EMR in het uitvoeren van een query. Daarnaast is een tweede hypothese dat Amazon EMR even snel is als Google Cloud Dataproc in het uitvoeren van een query.

1.5 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4, 5 en 6 vindt u voor respectievelijk Google BigQuery, Amazon EMR en Google Cloud Dataproc alle resultaten bekomen in dit onderzoek.

In Hoofdstuk 7 vindt u de volledige verwerking van alle bekomen uitvoeringstijden van de queries op elk van de drie cloudplatformen.

In Hoofdstuk 8, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

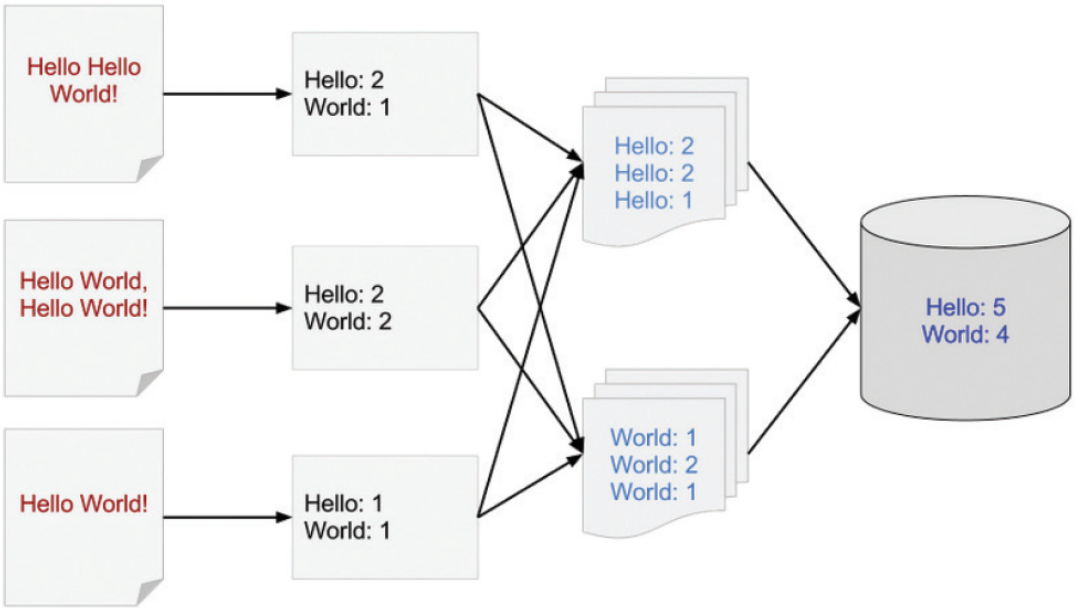
Zoals reeds in het vorige hoofdstuk aangegeven zal dit onderzoek zich richten op drie mogelijke oplossingen binnen het ruime aanbod van de diverse cloudplatformen, nl. Google BigQuery, Google Cloud Dataproc en Amazon EMR. Maar alvorens van start te gaan met de bespreking van deze oplossingen is het van belang om een inzicht te verwerven in de technologiën onder hun motorkap. Daarom wordt eerst het principe van MapReduce uitgelegd met daaropvolgend Hadoop en Apache Spark als verdere uitbreiding van het MapReduce-principe. Daarna wordt kort Apache HBase aangeraakt aangezien dit ook één van de mogelijkheden is binnen Amazon EMR. Uiteindelijk wordt de interne werking van elk van de drie oplossingen overlopen.

2.1 MapReduce

MapReduce is een framework om grote hoeveelheden data gedistribueerd te verwerken, elke computer in een MapReduce-cluster doet een stukje van het werk. De data kan zowel gestructureerd als ongestructureerd zijn en is dikwijls opgeslagen in HDFS (Hadoop Distributed File System, zie Sectie 2.2.1).

Binnen een cluster bevinden zich zogenaamde *mappers* en *reducers*. De mappers vormen de data om naar het gewenste formaat, de reducers aggregeren de verschillende resultaten bekomen door de mappers. De reducers doen hun werk steeds na de mappers. (IBM, 2018)

Dit principe wordt eenvoudig uitgelegd op Figuur 2.1. Men ziet in de eerste kolom een aantal bestanden met woorden als input. De mappers (rechthoeken) splitsen de bestanden in losse woorden, nl. ‘Hello’ en ‘world’. Daarna komen de reducers die van elke mapper het aantal voorkomens van ‘hun’ woord tellen. De eerste reducer telt het aantal voorkomens



Figuur 2.1: MapReduce Data flow (Sato, 2012)

van 'hello', de laatste het aantal voorkomens van 'world'. Uiteindelijk geven de reducers dit resultaat terug aan de gebruiker of wordt het opgeslagen.

MapReduce maakt het eenvoudig om data parallel te verwerken op een schaalbare manier zonder zich zorgen te hoeven maken over het ontwerp en de implementatie van een gedistribueerde verwerkingscluster of het aanschaffen van dure, high-end relationele databanken. De afgelopen jaren is Hadoop, de open-source implementatie van MapReduce, de populairste technologie geworden. (Sato, 2012)

Aan MapReduce zijn wel enkele nadelen verbonden. Het is niet alleen traag, maar men moet ook de hele query herstarten wanneer deze, door bv. een programmeerfout, fout liep. MapReduce is ontwikkeld als batch verwerkingsframework, daarom is het niet geschikt voor ad hoc en trial-and-error dataverwerking. Daarnaast is het grote voordeel van MapReduce wel dat het geschikt is om ongestructureerde data te verwerken. Een ander voordeel is dat men in eender welke programmeertaal MapReduce-bewerkingen kan schrijven (bv. Java, Python...).

2.2 Apache Hadoop

Hadoop werd in 2005 ontwikkeld door Doug Cutting en Mike Cafarella met als oorspronkelijk doel het Apache Nutch[1] project te ondersteunen. (Bappalige, 2014)

Volgens Bappalige (2014) bestaat Apache Hadoop uit de volgende vier componenten:

[1] Apache Nutch is een open-source zoekmachine, bedoeld als alternatief voor Google en Bing. (Apache, 2017)

- Hadoop Common: bevat bibliotheken en tools die andere Hadoop modules ook nodig hebben.
- Hadoop Distributed File System (HDFS): een gedistribueerd betandssysteeem dat data opslaat op goedkope basiscomputers, met een hoge bandbreedte doorheen een cluster als gevolg.
- Hadoop YARN: een resourcemanager die enerzijds de resources van de computers in de cluster beheert en anderzijds de applicaties van de gebruikers inplant.
- Hadoop MapReduce: een programmeermodel om grote hoeveelheden date te verwerken (zie ook Sectie 2.1).

Het hele ecosysteem van Apache Hadoop bestaat momenteel reeds uit verschillende modules, elk met hun eigen doel. Deze modules zijn onder andere Apache Pig, Apache Hive, Apache HBase...

2.2.1 Hadoop Distributed File System

Hadoop draait volledig op het Hadoop Distributed File System (HDFS), een gedistribueerd, schaalbaar en draagbaar bestandssysteem geschreven in Java. HDFS draait op een cluster bestaande uit *namenodes* en *datanodes*. De namenodes zijn de beherende nodes van de cluster, ze splitsen een bestand op in verschillende data blocks en verdelen het over de verschillende datanodes. De namenode onthoudt op zijn beurt waar elk stuk van een bestand zich bevindt in de cluster. De datanodes slaan uiteindelijke de verschillende data blocks op. Elk block heeft een grootte van maximaal 128MB (Shailna, 2017), maar dit kan in de configuratie nog steeds gewijzigd worden. (Bappalige, 2014)

Binnen HDFS wordt de data standaard in RAID 3 opgeslagen, d.w.z. dat hetzelfde data block op 3 verschillende datanodes wordt opgeslagen. Deze standaardwaarde kan nog aangepast worden in de configuratie. Sinds versie 2.x beschikt HDFS over hoge beschikbaarheid door het maken van een backup van de namenode. Indien een cluster maar één namenode heeft, is de data op de datanodes waardeloos als deze namenode faalt. Daarom is het nuttig om zowel voor de datanodes (via RAID x) als voor de namenode replicatie te implementeren of voorzien. (Bappalige, 2014)

De backup van de namenode wordt gemaakt op de secundaire namenode. Deze neemt de cluster niet over wanneer de primaire namenode faalt, maar neemt op regelmatige tijdstippen een snapshot (= a.d.h.v. checkpoints) van de primaire namenode. Deze snapshots of checkpoints worden gebruikt om de primaire namenode te herstellen wanneer deze faalt. (Bappalige, 2014)

Een grote bottleneck van HDFS kan zijn dat grote bestanden relatief traag opgehaald worden omdat de cluster slechts over één primaire namenode beschikt. Dit probleem kan echter opgelost worden door meerdere namespaces te maken die beheerd worden door meerdere aparte en zelfstandige namenodes. (Bappalige, 2014)

2.2.2 Hadoop MapReduce

Zoals reeds uitgelegd in Sectie 2.1 is MapReduce een programmeermodel om grote hoeveelheden ongestructureerde data te verwerken. Hieronder volgt meer informatie over hoe Hadoop MapReduce werkt.

Binnen Hadoop MapReduce zijn er enerzijds de *JobTrackers* en anderzijds de *TaskTrackers*. De JobTrackers krijgen een taak (= job) binnen van een client en verdelen deze taak over verschillende TaskTrackers. De verdeling wordt gedaan a.d.h.v. welke data zich op welke node bevindt. De TaskTrackers zijn dan verantwoordelijk voor de map- of reduce-taak op hun specifieke node. Zij geven uiteindelijk het resultaat van de taak terug aan de JobTracker. Wanneer een TaskTracker faalt of een vooraf gedefineerde tijdslimiet overschrijdt, dan zal de JobTracker de taak opnieuw inplannen en laten uitvoeren. (Bappalige, 2014)

2.2.3 Yet Another Resource Negotiator

Apache Hadoop onderging een grote revisie en werd omgevormd tot MapReduce 2.0 of YARN (= Yet Another Resource Negotiator). De filosofie van MapReduce 2.0 is het opsplitsen van de twee taken van de JobTracker, resource management en het plannen en monitoren van taken, in afzonderlijke processen. Daarvoor heeft men de Resource Manager (RM) en de Application Manager (AM) ontwikkeld. Per applicatie is er één Resource Manager die de resources beheert. Per taak is er één Application Manager die een klassieke taak of een MapReduce-taak beheert. Dus er is maar één ResourceManager voor een volledige cluster, maar er kunnen meerdere Application Managers zijn (één per taak als geheel). Per slaafnode in de cluster, de node die het effectieve werk doet, is er één NodeManager (NM). (Bappalige, 2014)

De Application Manager onderhandelt met de Resource Manager over de resources die nodig zijn voor de taak waarvoor hij verantwoordelijk is. Daarnaast werken de Application Managers ook samen met de Node Managers om hun taak te kunnen uitvoeren en monitoren. De Resource Manager beschikt over een planner die verantwoordelijk is voor het alloceren van de resources voor de verschillende lopende taken. De planner plant de taken op basis van de vereisten van de verschillende taken. De Node Manager is verantwoordelijk voor het opstarten van de containers voor de taken, het monitoren van het resourcegebruik en het rapporteren aan de Resource Manager. Zoals reeds eerder vermeld is de Application Manager verantwoordelijk voor het onderhandelen over de benodigde resources, hun status en de voortgang van de taak monitoren. (Bappalige, 2014)

2.3 Apache Spark

Apache Spark vindt zijn oorsprong in 2009 bij een team van ontwikkelaars aan de University of California, men wou een ééngemaakt systeem ontwikkelen om big data te verwerken en men wou de traagheid van MapReduce aanpakken. Apache Spark is gebouwd volgens het principe van MapReduce, maar beschikt daarnaast ook over Resilient Distributed

Datasets of RDD's. (Zaharia e.a., 2016) Apache Spark biedt, mede door MapReduce, de mogelijkheid om de verwerking van data te verdelen over een cluster van servers die elk een stukje van de verwerking doen.

2.3.1 Resilient Distributed Datasets

Resilient Distributed Datasets of RDD's zijn het belangrijkste onderdeel van Apache Spark, alle andere bibliotheken zijn hierop gebouwd. Daarom vindt u in deze Sectie een uitgebreide uitleg over het begrip RDD of Resilient Distributed Datasets.

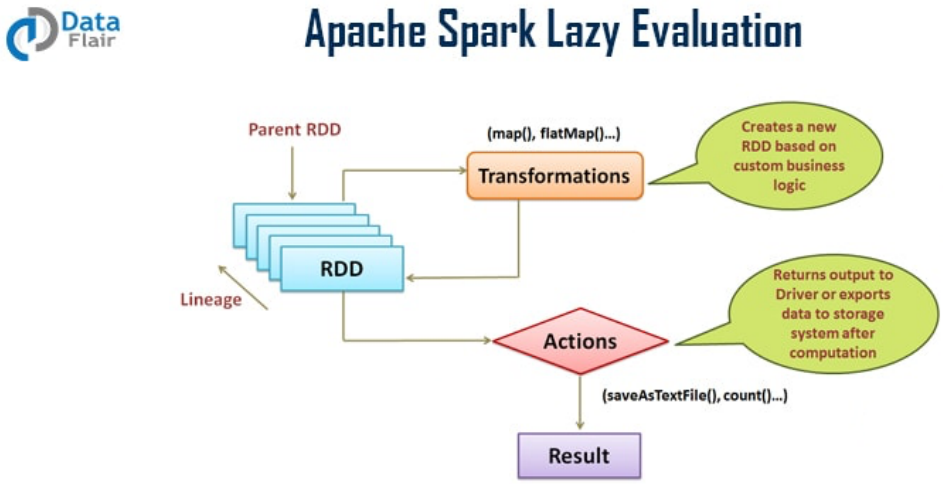
Volgens Zaharia e.a. (2012) is een RDD een gedistribueerde abstractie die programmeurs toelaat om in-memory bewerkingen uit te voeren op grote clusters op een fout-tolerante manier. Dit verklaart ook de naam Resilient Distributed Datasets. Resilient betekent fout-tolerant wat toelaat om ontbrekende of beschadigde partities, ten gevolge van falende nodes, te herstellen. Distributed staat voor gedistribueerd wat betekent dat de data verdeeld is over verschillende nodes. Datasets zijn collecties van gepartitioneerde data bestaande uit primitieve datatypes. (Laskowski, g.d.)

Met behulp van de RDD's biedt Apache Spark ondersteuning voor SQL, streaming, machine learning en bewerkingen op grafen. Voorheen was voor elk van deze een ander soort technologie nodig terwijl dit nu allemaal gegeneraliseerd is in Apache Spark.

Apache Spark's generaliteit biedt een aantal voordelen waaronder een veralgemende API. Ten tweede is Apache Spark veel sneller dan andere technologieën door het (vaak) in-memory uitvoeren van verschillende functies op dezelfde data terwijl andere technologieën eerst de data moeten opslaan vooraleer men ze kan manipuleren of verwerken. Als laatste biedt Apache Spark de mogelijkheid om nieuwe applicaties te ontwikkelen (streaming, machine learning...) die voorheen niet mogelijk waren. (Zaharia e.a., 2016) RDD's kunnen gemanipuleerd worden met Java, Python, Scala en R.

Daarnaast zijn RDD's heel performant aangezien ze lazy uitgevoerd worden. Daardoor kan Apache Spark het optimale uitvoeringsplan bepalen alvorens alle acties te hoeven uitvoeren. Elke operatie retourneert een nieuwe RDD met het resultaat van de transactie, maar dit resultaat wordt nog niet meteen berekend. Wanneer dan een nieuwe actie op het RDD wordt uitgevoerd, gaat Apache Spark bepalen wat het efficiënste uitvoeringsplan is. Daarvoor zal het rekening moeten houden met de verschillende transformaties die gedaan worden zodat het data niet onnodig doorheen de cluster hoeft rond te sturen. Intern wordt hiervoor een graaf bijgehouden met de opeenvolgende transformaties op een RDD tot een (beëindigende) actie uitgevoerd wordt. Dan wordt het optimale pad doorheen de graaf bepaald. (Zaharia e.a., 2016) Kortom in de basis zijn transformaties van RDD's lazy en acties op RDD's niet, dit principe wordt eenvoudig uitgelegd in Figuur 2.2.

Als laatste zijn RDD's ook geschikt voor het delen van data doorheen verschillende berekeningen. Apache Spark biedt de mogelijkheid om het resultaat van een actie in een RDD op te slaan in het geheugen zodat de waarde niet steeds herberekend dient te worden. (Zaharia e.a., 2016)



Figuur 2.2: Lazy evaluering van RDD's (DataFlair, 2017)

Zoals reeds eerder gezegd is alles in Apache Spark gebaseerd op RDD's. Daarom wordt in wat volgt alle onderdelen van Apache Spark die gebaseerd zijn op RDD's kort aangeraakt en omschreven. Het gaat hier om Dataframes, Spark Streaming, GraphX en MLib.

2.3.2 Dataframes

Eén van de meest gebruikte manieren om een query uit te voeren op data is door de query te schrijven in SQL. Daarvoor heeft Apache Spark ook een library voorzien die een gebruiker toelaat om pure SQL-queries uit te voeren op een dataset. Daarvoor bestaat ook een abstractie genaamd Dataframes. Dit is een representatie van een tabel uit een normale SQL databank bestaande uit RDD's van records met een gekend schema. Dit schema zegt op welke plaats in het RDD welke kolom te vinden is. Om de SQL om te zetten in transformaties en acties bestaat de Spark SQL Engine die ook enkele optimalisaties zal uitvoeren indien mogelijk. (Zaharia e.a., 2016)

2.3.3 Spark Streaming

Deze library biedt de mogelijkheid om met Apache Spark streams van data te ontvangen en te verwerken. Daarvoor maakt Apache Spark gebruik van zogenaamde *discretized streams* (discrete streams) die de data in kleine batches opsplitst. Bijvoorbeeld elke 500 milliseconden wordt de data vernieuwd. Deze batches worden dan gecombineerd met de data in de RDD's in het geheugen om een nieuw resultaat te bekomen met de nieuwe data. (Zaharia e.a., 2016)

2.3.4 Extra libraries

Andere libraries aanwezig in Apache Spark zijn GraphX en MLib. GraphX laat het toe om bewerkingen op grafen uit te voeren. MLlib biedt de mogelijkheid om aan Machine

Learning te doen a.d.h.v. gedistrubueerde modeltraining.

Als laatste wordt een tweede lid van de Apache familie kort omschreven aangezien dit één van de mogelijkheden is binnen Amazon EMR. Deze mogelijkheid wordt verder niet besproken of gebruikt aangezien dit buiten de scope van dit onderzoek ligt.

2.4 Apache HBase

Volgens Rouse (2013) is Apache HBase een kolomgeörienteerde databank gebouwd bovenop HDFS (zie ook Sectie 2.2.1), HBase is m.a.w. een NoSQL databank. Het werd ontwikkeld om grote datasets gedistribueerd te verwerken. Het is in staat om snel tabellen te updaten en om horizontaal opgeschaald te worden. Volgens Rouse (2013) is het meest vooraanstaande gebruik van Apache HBase de structuur achter Facebook's berichtensysteem.

Volgens Rouse (2013) biedt HBase een hoge consistentie bij lees- en schrijfoperaties, daarmee onderscheidt het zich van andere NoSQL-databanken. De interne structuur van HBase is enorm vergelijkbaar met Hadoop waarbij er ook één master node aanwezig is die de volledige cluster en zijn slave nodes beheert.

Zoals reeds eerder gezegd heeft IntoData ervoor gekozen om in dit onderzoek enkel de focus te leggen op Amazon EMR, Google BigQuery en Google Cloud Dataproc. Daarom vindt u in wat volgt een uitgebreide omschrijving van elk van deze oplossingen.

2.5 Amazon EMR

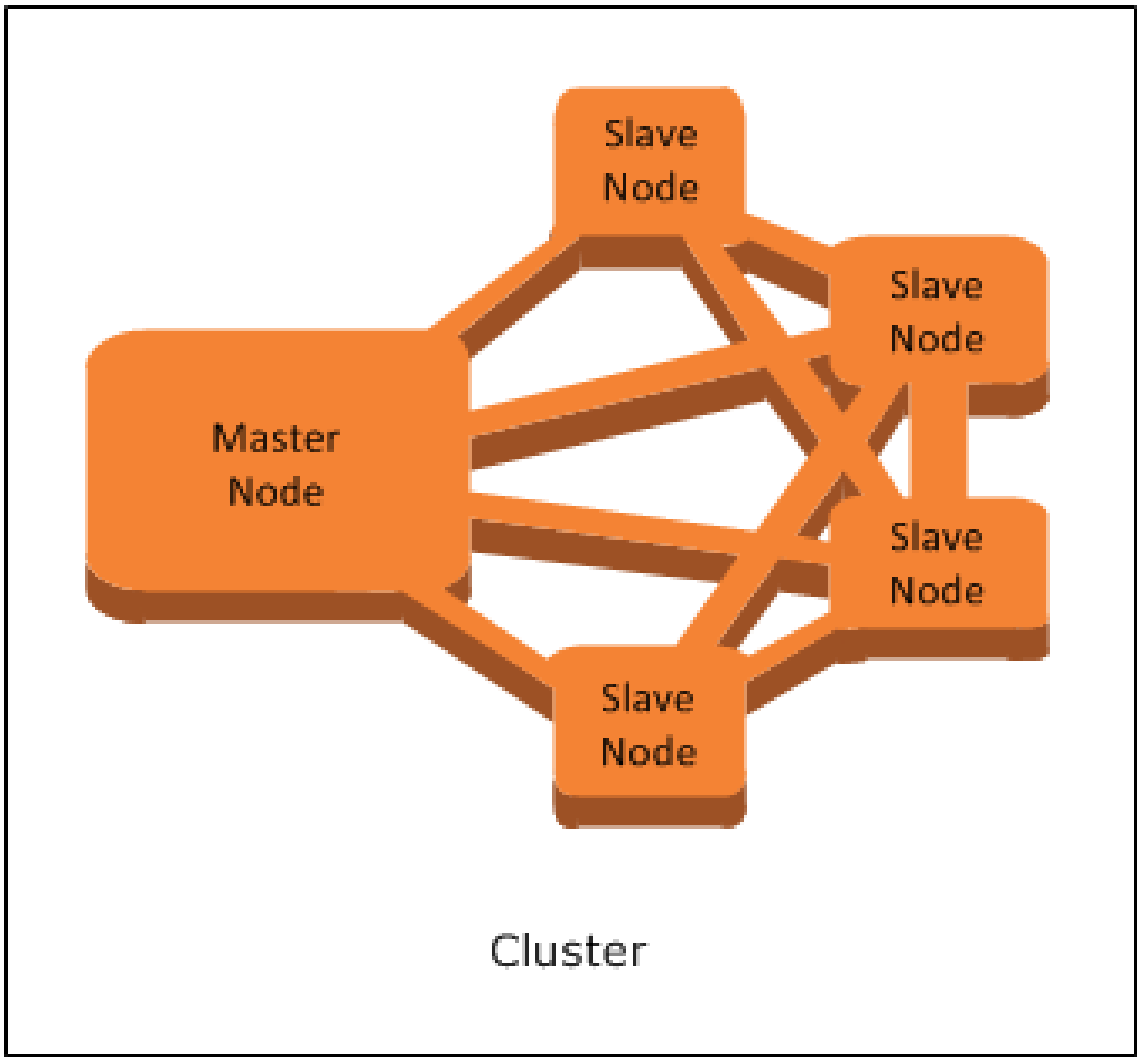
Amazon EMR staat voor Amazon Elastic MapReduce en is een Hadoop framework dat het eenvoudig maakt om grote hoeveelheden data te verwerken in Amazon EC2, Apache Spark, HBase, Presto of Flink. Daarbij is het ook mogelijk om de data uit AWS data stores zoals Amazon S3 en Amazon DynamoDB op te halen en te verwerken. In wat volgt wordt de interne strucuur van de Amazon Elastic Compute Cloud besproken.

2.5.1 Amazon Elastic Compute Cloud

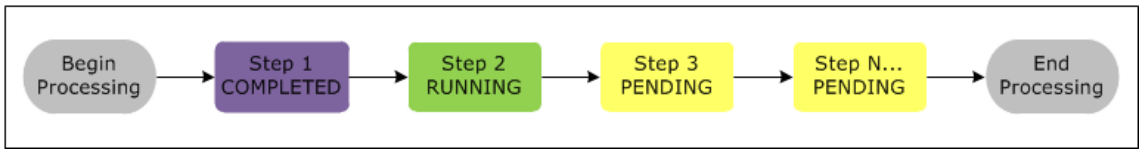
Amazon EMR verwerkt Big Data op een Hadoop cluster van virtuele servers in de Amazon Elastic Compute Cloud (EC2) en Amazon Simple Storage Service (S3) (Rouse, 2017). Het woord elastic verwijst naar het feit dat men de clusters dynamisch kan uitbreiden of inperken naar gelang wat er nodig is.

Clusters en nodes

Een Amazon EC2 cluster bestaat uit één master node en één of meerdere slave nodes, elke node is een Amazon EC2 instantie. Een schema hiervan ziet u op Figuur 2.3



Figuur 2.3: Een Amazon EC2 cluster (Amazon, 2018b)



Figuur 2.4: Verwerking a.d.h.v. een sequentie van stappen (Amazon, 2018b)

De master node

- beheert de volledige cluster,
- coördineert de distributie van data,
- verdeelt taken (verwerking),
- en monitort alle lopende taken op de slave nodes.

Indien nodig herstart de master node een corrupte of misgelopen taak op een van de slave nodes.

De slave nodes zijn ofwel core nodes of task nodes. Core nodes bevatten software die de data verwerken en deze ook opslaan in een Hadoop Distributed File System in de cluster. Task nodes bevatten enkel de sofware voor de dataverwerking, deze zijn optioneel in een cluster.

Dataverwerking

Om data te verwerken op een Amazon EC2 cluster zijn er twee manieren: de taken rechtstreeks op de cluster uitvoeren of een sequentie van stappen uitvoeren. Wanneer men de taken rechtstreeks op de cluster wil uitvoeren, connecteert men met de cluster en gebruikt men de verschillende interfaces en tools die ter beschikking zijn voor de software die draait op de cluster. Wanneer men in stappen werkt, kan men een aantal geordende stappen sturen naar de cluster die elk een deel van het werk doen a.d.h.v. instructies die de data manipuleren. Hieronder ziet u een voorbeeld van een proces met vier stappen:

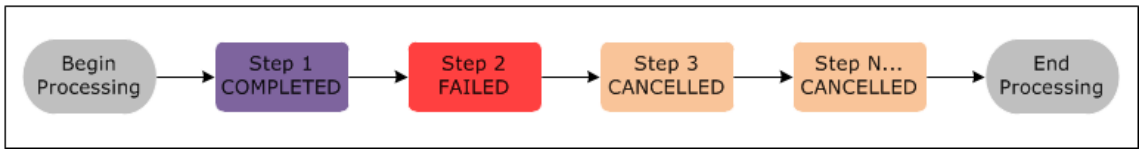
1. Een dataset inladen
2. Het resultaat van een eerste stap bepalen a.d.h.v. een Apache Pig[2] programma
3. Het resultaat van een tweede stap bepalen a.d.h.v. een Apache Hive[3] programma
4. Een dataset terugsturen als output (Amazon, 2018b)

Typisch wordt de data voor een Amazon EMR cluster opgeslagen in een bestandssysteem zoals Amazon S3 of HDFS. Deze data gaat dan doorheen alle stappen in het proces en wordt uiteindelijk door de laatste stap terug opgeslagen in het gekozen bestandssysteem.

De sequentie van stappen wordt steeds uitgevoerd in onderstaande volgorde, net zoals te zien is op Figuur 2.4:

[2] Apache Pig valt buiten de scope van dit onderzoek, maar meer informatie is te vinden op pig.apache.org.

[3] Apache Hive valt buiten de scope van dit onderzoek, maar meer informatie is te vinden op hive.apache.org.



Figuur 2.5: Een falende sequentie van stappen (Amazon, 2018b)

1. Een request om het proces te starten komt binnen.
2. Alle stappen worden op **PENDING** gezet.
3. De eerstvolgende stap van het proces start en wordt op **RUNNING** gezet, alle andere stappen blijven in **PENDING**.
4. Wanneer deze stap klaar is, wordt deze op **COMPLETED** gezet.
5. De volgende stap wordt op **RUNNING** gezet en zet zich op **COMPLETED** wanneer deze klaar is.
6. Dit patroon herhaalt zich tot alle stappen op **COMPLETED** staan en het proces dus klaar is. (Amazon, 2018b)

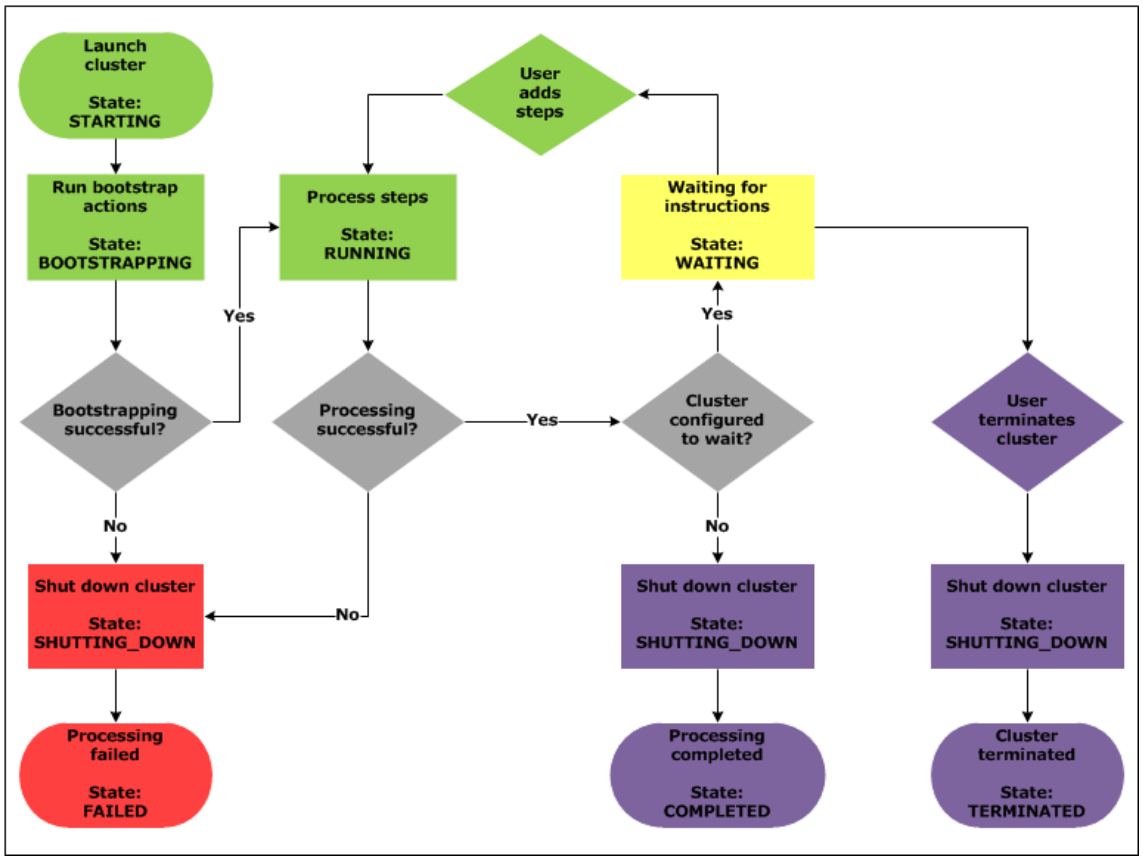
Wanneer er binnen een stap iets fout gaat, wordt de status van de stap op **TERMINATED_WITH_ERRORS** gezet. Alle volgende stappen worden op **CANCELLED** gezet en het proces wordt gestopt. Alhoewel men ook kan kiezen om de volgende stappen toch nog uit te voeren, dit wordt niet aangeraden. (Amazon, 2018b) Dit wordt duidelijk weergegeven op Figuur 2.5:

Cluster lifecycle

Een succesvolle cluster op Amazon EMR doorloopt volgende stappen:

1. Status **STARTING**: Amazon EMR start alle nodige applicaties in de cluster op.
2. Status **BOOTSTRAPPING**: een aantal door de gebruiker gedefinieerde acties worden uitgevoerd, bv. extra applicaties die geïnstalleerd moeten worden.
3. Status **RUNNING**: het bootstrappen is gedaan en de applicatie voert alle stappen sequentieel uit.
4. Status **WAITING** of **SHUTTING_DOWN**: na het uitvoeren van alle stappen kan de cluster ofwel wachten op de volgende set instructies ofwel kan deze zichzelf afsluiten. Dit kan de gebruiker zelf kiezen.

Elke fout tijdens het draaien van de cluster resulteert in een terminatie van de cluster en alle state die reeds opgebouwd was. Er zijn wel mogelijkheden tot het opslaan van de cluster indien gewenst, dit kan zowel automatisch als handmatig. De status van de cluster na een fout is **TERMINATED_WITH_ERRORS**. Wanneer state-herstel ingeschakeld is, wordt de cluster niet afgesloten. Op Figuur 2.6 ziet u een schema van de verschillende statussen van een cluster met de bijbehorende overgangen ertussen. (Amazon, 2018b)



Figuur 2.6: De cluster lifecycle (Amazon, 2018b)

2.6 Google BigQuery

Google's tegenhanger van Amazon EMR is Google BigQuery, een publieke implementatie van Dremel. Het laat de gebruiker toe om een aantal kernfunctionaliteiten van Dremel te gebruiken a.d.h.v. een REST API, command line interface, Web UI, access control, enzovoort. In wat volgt wordt de interne structuur en uitvoering van een query in Dremel besproken.

2.6.1 Dremel

Dremel is een gedistribueerd systeem ontwikkeld door Google. Het wordt gebruikt als querymechanisme in Google BigQuery en het diende als inspiratie voor Apache Drill en Apache Impala.

Volgens Melnik e.a. (2010) is Dremel een schaalbaar, interactief ad-hoc querysysteem voor de analyse van read-only geneste data. Door het combineren van multi-level uitvoeringsbomen en kolomvormige datalay-out is Dremel in staat om aggregatiequery's op biljoenen rijen aan data uit te voeren in enkele seconden.

Volgens Sato (2012) maakt Dremel gebruik van Google's infrastructuur wat toelaat om een query parallel uit te voeren op tienduizenden servers tegelijk wat resulteert in een uitvoeringstijd van enkele seconden. Als voorbeeld maakt Sato (2012) gebruik van de "wikipedia" tabel die 314 miljoen rijen bevat, goed voor 35,7 GB aan geheugen. Daarop voerde men onderstaande query uit:

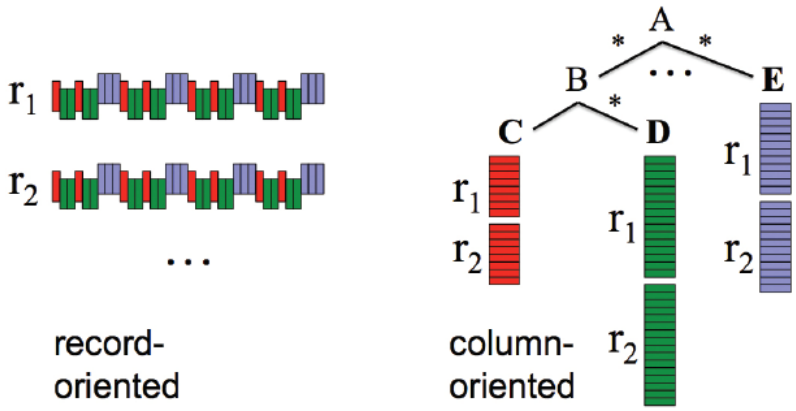
```
1 SELECT COUNT(*)
2 FROM publicdata:samples.wikipedia
3 WHERE REGEXP_MATCH(title, '[0-9]*') AND wp_namespace = 0;
```

Deze query heeft als gevolg dat de regulaire expressie op elke rij getest moet worden, alsook de andere WHERE clause. Hierbij is het van belang om te weten dat er geen indexen aanwezig waren op de tabel en dus sowieso elke rij getest moest worden. Men kreeg *223 163 387* als resultaat uit deze query na slechts 10 seconden. Volgens Sato (2012) bewijst dit de kracht van Dremel en Google BigQuery.

Kolomgeoriënteerde datastructuur

Dremel maakt gebruik van een kolomgeörienteerde datastructuur, volgens Sato (2012) laat dit toe een veel hogere compressie en scansnelheid te bereiken. Dremel verdeelt een record in kolommen en persisteert elke waarde op een andere opslagvolume, een traditionele databank zal elk record op een één opslagvolume opslaan. Dit verschil is duidelijk te zien op Figuur 2.7.

Volgens Sato (2012) zijn kolomgeoriënteeerde databanken een gangbare techniek in data-warehouses, het heeft dan ook enkele voordelen:



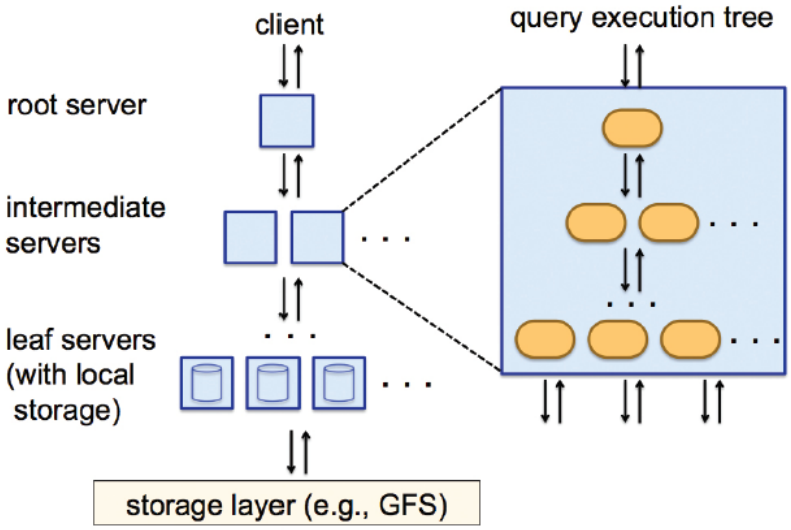
Figuur 2.7: Rijgebaseerde versus kolomgebasseerde datastructuur (Sato, 2012)

- Verminderd dataverkeer aangezien enkel de kolommen van de query opgehaald moeten worden en niet het hele record.
- Hogere compressie omdat elke kolom veel specifiekere waarden heeft dan een record van een traditionele rijgebaseerde databank.

Naast deze voordelen is er één groot nadeel verbonden aan kolomgeoriënteerde databanken: ze zijn niet efficiënt in het updaten van bestaande records. Zoals eerder gezegd maakt Dremel gebruik van read-only datasets, dit impliceert dat Dremel geen updates ondersteunt. Toch is Dremel volgens Sato (2012) een heel populair data warehouse databank aangezien het een van de eerste implementaties van kolomgeoriënteerde analytische systemen was dat toeliet om de kracht van verschillende systemen te bundelen.

Boomstructuur van query's

Volgens Sato (2012) was de grootste uitdaging voor Dremel hoe de query's verdeeld worden over de verschillende computers die de resultaten bepalen en hoe men achteraf deze resultaten aggregeert tot één resultaat voor de query. De oplossing hiervoor was de query omzetten in een boomstructuur waarbij men de query eerst naar beneden duwt tot aan de bladeren en uiteindelijk alle resultaten van de bladeren aggregeert naar boven. Op Figuur 2.8 wordt men dit principe voorgesteld. De client stuurt een query naar de root server, deze zal daarop metadata ophalen van de tabel waarop de query uitgevoerd wordt. Daarna wordt de query doorgestuurd naar een intermediate server die de query verdeelt over de verschillende leaf servers. De leaf servers gaan uiteindelijk communiceren met de persistentielaag om een resultaat te kunnen bepalen. Dit resultaat wordt daarna terug van de leaf servers via de intermediate servers en de root server teruggestuurd naar de client. Het zijn de intermediate servers die de uitvoeringsboom van de query gaan bepalen. (Melnik e.a., 2010)



Figuur 2.8: Queryuitvoering in Dremel (Sato, 2012)

BigQuery versus MapReduce

In Sectie 2.1 werd MapReduce reeds besproken, in deze Sectie wordt het verschil tussen Google BigQuery en MapReduce besproken aangezien beide op het eerste zicht veel gelijkenissen vertonen.

Volgens Sato (2012) is Dremel ontwikkeld als een interactieve data-analysetool voor grote datasets. MapReduce is ontwikkeld als een programmeermodel om grote datasets in batches te verwerken. Daarboven is Dremel ontwikkeld om een resultaat te geven voor een query in enkele seconden, MapReduce doet hier veel langer over (minstens minuten aldus Sato (2012)).

Zoals reeds eerder gezegd kan de data voor MapReduce zowel gestructureerd als ongestructureerd zijn. Bij Google BigQuery moet de data eerst gestructureerd worden in bv. een csv-bestand zodat het opgeslagen kan worden in de Google Cloud Storage waarna het doorgesluisd kan worden naar BigQuery. Bij MapReduce wordt de data verwerkt door gebruik te maken van een programmeertaal, bij BigQuery maakt men gebruik van SQL (of de JavaScript variant). BigQuery kan bijvoorbeeld gebruikt worden voor OLAP (Online Analytical Processing) of BI (Business Intelligence), hierbij zijn alle queries makkelijk en snel uitgevoerd door een aantal snelle aggregaties of filters toe te passen. MapReduce is dan weer geschikt voor applicaties die aan data mining doen waarbij men bijvoorbeeld complexe statistische berekeningen moet uitvoeren. MapReduce kan ook gebruikt worden als men veel gigabytes aan data als output heeft, zoals bij het mergen van twee grote tabellen. (Sato, 2012)

Hieronder staan enkele voorbeelden van use cases voor zowel BigQuery als voor MapReduce.

Use cases voor BigQuery ((Sato, 2012))

- Specifieke records vinden o.b.v. specifieke voorwaarden. *Bv.: request logs vinden voor een bepaald account ID.*
- Snelle aggregaties van statistieken met dynamische condities (d.w.z. condities die continu wijzigen). *Bv.: een samenvatting van het volume aan requests voor een webapplicatie van de vorige nacht a.d.h.v. een grafiek.*
- Trial-and-error data-analyse. *Bv.: de oorzaak van problemen identificeren en waarden aggregeren o.b.v. verschillende condities zoals uur, dag...*

Use cases voor MapReduce ((Sato, 2012))

- Complexe data mining uitvoeren,die verschillende iteraties en dataverwerkingspaden vereist, op Big Data.
- Grote join operaties uitvoeren op grote datasets.
- Exporteren van grote hoeveelheden data na het verwerken

Maar volgens Sato (2012) is de beste use case het combineren van beide zoals in onderstaande voorbeelden.

- Met behulp van MapReduce grote join operaties en dataconversies uitvoeren waarna BigQuery snelle aggregaties en ad hoc data-analyses uitvoert op het resultaat van de MapReduce.
- BigQuery gebruiken om een snelle data-analyse uit te voeren alvorens de productiecode te schrijven voor data mining a.d.h.v. MapReduce.

Kortom, volgens Sato (2012) is MapReduce beter geschikt voor langlopende batch processen zoals data mining. BigQuery is dan weer beter geschikt voor ad hoc OLAP- en BI-queries die snel resultaat vereisen.

2.7 Google Cloud Dataproc

Als laatste werpt deze inleiding nog een korte blik op Google Cloud Dataproc, het dichtst aanleunend platform bij een Amazon EMR Apache Spark cluster. Google (2018b) omschrijft Google Cloud Dataproc als volgt:

> Cloud Dataproc is een snelle, gebruiksvriendelijke en volledig beheerde cloudservice waarmee u Apache Spark- en Apache Hadoop-clusters eenvoudiger en voordeliger kunt uitvoeren. Bewerkingen die vroeger uren of dagen in beslag namen, duren nu nog maar seconden of minuten. Daarnaast betaalt u alleen voor de resources die u daadwerkelijk gebruikt (met facturering per seconde). Cloud Dataproc is eenvoudig te integreren met andere Google Cloud Platform-services (GCP). Zo beschikt u over een krachtig en compleet platform voor gegevensverwerking, analyse en machine learning.

Hieruit kan men opmaken dat Google Dataproc draait op Apache Spark (zie Sectie 2.3) en Apache Hadoop (zie Sectie 2.2) en zodoende ook werkt volgens het principe van

MapReduce (zie Sectie 2.1). Het grootste voordeel aan Google Cloud Dataproc is dat men maar hoeft te betalen naargelang men verbuikt of gebruikt.

Volgens Tereshko (2017) verschilt Google's implementatie van Cloud Dataproc niet zoveel van de standaard implementatie van een Apache Spark cluster. Het biedt wel de mogelijkheid om voor elke taak een aparte cluster te starten. Waarom is dit nu mogelijk? Omdat Google Cloud Dataproc het toestaat om te betalen per minuut van gebruik, omdat de clusters snel opstarten (max. 90 seconden volgens Tereshko (2017)) en omdat de opslag gescheiden is van de verwerking. Deze voordelen laten toe om per opdracht een cluster op te zetten, iets wat bij bv. Amazon EMR niet mogelijk is omdat men betaalt per cluster en per uur.

Volgens Tereshko (2017) zijn andere operationele en economische voordelen van Google Cloud Dataproc:

- Scheiding van resources, dit vermijdt eventuele niet zo voor de hand liggende bottlenecks en resource-strijden tussen verschillende taken.
- Simpel beheer, men hoeft niet zelf de cluster te beheren m.b.v. YARN bijvoorbeeld.
- Simpel geprijsd, je hoeft slechts te betalen per minuut.
- Simpel uitbreiden van de clustergrootte, men vraagt simpelweg aan Dataproc om de taak meer resources te geven.
- Simpeler problemen oplossen, resources zijn gescheiden dus niets anders kan de oorzaak zijn van een probleem.

Kortom, Google Cloud Dataproc biedt een eenvoudige manier om Apache Spark en Apache Hadoop clusters op te zetten voor (relatief) weinig geld, je betaalt slechts per minuut.

3. Methodologie

Zoals elk onderzoek start ook dit onderzoek met een uitgebreide literatuurstudie van het onderzoeksdomein en van de gebruikte frameworks en technologiën. Deze literatuurstudie is te vinden in Hoofdstuk 1.

Na de literatuurstudie volgt een requirementsanalyse waarin de verschillende requirements van IntoData verzameld en geanalyseerd worden. Deze requirements zijn van groot belang tijdens dit onderzoek aangezien ze de basis vormen om een oplossing te beoordelen. Zoals in de inleiding (zie Hoofdstuk 1) reeds aan bod kwam, focust dit onderzoek op Google BigQuery, Amazon EMR en Google Cloud Dataproc. Bijgevolg is een long en short list overbodig in het kader van dit onderzoek. Daarom wordt in wat volgt slechts een requirementsanalyse opgesteld, maar geen long en short list.

Na de requirementsanalyse volgt het onboarden van de data, m.a.w. het voorbereiden en opslaan van data. Onder het voorbereiden van de data verstaat men het opruimen van de datasets zodat deze correct geformatteerd zijn voor de verwerking. Voor dit onderzoek is gekozen voor de dataset NYC Parking Tickets. Deze dataset is vrij beschikbaar op https://www.kaggle.com/new-york-city/nyc-parking-tickets/data. Tijdens de onboarding wordt de volledige dataset gedownload en uitgepakt. Op het moment van dit onderzoek bestaat de dataset uit data uit vier csv-bestanden van de jaren 2013-2014, 2015, 2016 en 2017 (de eerste dataset is een combinatie van 2013 en 2014). De dataset bestaat uit een oplijsting van verschillende soorten boetes waaronder bijvoorbeeld parkeerboetes en boetes van door het rood licht te rijden. Elk bestand heeft onderstaande kolommen. Voor datums en tijd wordt een bepaald formaat gebruikt waarvan u de legende onder de opsomming van de kolommen vindt.

Kolommen csv-bestanden

- Summons Number (nummer): nummer van de dagvaarding
- Plate ID (tekst): numerplaat
- Registration State (tekst): staat waar de auto ingeschreven is
- Plate Type (tekst): het type nummerplaat
- Issue Date (datum): datum van het uitschrijven van de boete (formaat: DD/MM/YYYY of MM/DD/YYYY)
- Violation Code (nummer): code gekoppeld aan het type overtreding
- Vehicle Body Type (tekst): vorm van het voertuig
- Vehicle Make (tekst): merk van het voertuig
- Issuing Agency (tekst): instantie die de boete uitschreef
- Street Code1 (nummer): code gekoppeld aan de straat waarin het voertuig stond
- Street Code2 (nummer): code gekoppeld aan de straat waarin het voertuig stond
- Street Code3 (nummer): code gekoppeld aan de straat waarin het voertuig stond
- Vehicle Expiration Date (nummer): datum waarop het voertuig vervalt (formaat: YYYYMMDD)
- Violation Location (tekst): plaats van de overtreding
- Violation Precinct (nummer): nummer van het politiedistrict waarin de overtreding werd gepleegd
- Issuer Precinct (nummer): nummer van het politiedistrict van de persoon die de boete uitschreef
- Issuer Code (nummer): nummer die de uitgever van de boete identificeert
- Issuer Command (tekst): de macht van de uitgever van de boete
- Issuer Squad (tekst): het team van de uitgever van de boete
- Violation Time (tekst): tijd van de overtreding (formaat: HHmm gevolgd door A indien voormiddag of P indien namiddag)
- Time First Observed (tekst): de tijd van de eerste waarneming van de overtreding
- Violation County (tekst): staat/provincie van overtreding
- Violation In Front Of Or Opposite (tekst): letter die aangeeft waar de overtreding werd gepleegd (O voor opposite (tegengesteld), F voor front (voor))
- House Number (tekst): huisnummer van de plaats waar de overtreding werd gepleegd
- Street Name (tekst): straatnaam van de plaats waar de overtreding werd gepleegd
- Intersecting Street (tekst): straatnaam van een eventueel kruisende straat met de plaats waar de overtreding werd gepleegd
- Date First Observed (nummer): datum van de eerste waarneming van de overtreding (formaat: YYYYMMDD)
- Law Section (nummer): sectie van de wet waarop de overtreding werd gepleegd
- Sub Division (tekst): subdivisie van de rechtsectie (law section) waarop de overtreding werd gepleegd
- Violation Legal Code (tekst): geeft aan of het een overtreding van een legale wet was (T voor waar, F voor onwaar)
- Days Parking In Effect (tekst): dagen dat het voertuig reeds geparkeerd was
- From Hours In Effect (tekst): startuur
- To Hours In Effect (tekst): einduur
- Vehicle Color (tekst): kleur van het voertuig

- Unregistered Vehicle? (tekst): geeft aan of het een ingeschreven voertuig is of niet
- Vehicle Year (nummer): bouwjaar van het voertuig
- Meter Number (tekst): nummer van de meter waar de overtreding werd gepleegd
- Feet From Curb (nummer): aantal feet van de stoeprand
- Violation Post Code (tekst): postcode van de plaats van de overtreding
- Violation Description (tekst): beschrijving van de overtreding
- No Standing or Stopping Violation (tekst): geeft aan of het een overtreding was op een plaats waar men niet mag stoppen of staan
- Hydrant Violation(tekst): geeft aan of het een overtreding was op de een brandkraan (bv. de toegang verhinderen)
- Double Parking Violation(tekst): geeft aan of het voertuig dubbel geparkeerd stond

Legende datum en tijd

- YYYY: het jaar weergegeven a.d.h.v. vier cijfers, bijvoorbeeld 2018
- MM: de maand weergegeven a.d.h.v. twee cijfers, bijvoorbeeld 02 voor februari
- DD: de dag weergegeven a.d.h.v. twee cijfers, bijvoorbeeld 10 voor de 10e dag van de maand
- HH: het uur weergegeven a.d.h.v. twee cijfers, bijvoorbeeld 22 voor 10u 's avonds
- mm: de minuten weergegeven a.d.h.v. twee cijfers

De data van de NYC Parking tickets is mooi geformatteerd uitgezonderd de dataset van 2015. Daarbij zijn twee kolommen met een datum fout geformatteerd t.o.v. de andere drie datasets. Het gaat hier om de kolom Vehicle Expiration Date en de kolom Date First Observed. Om deze datums correct te formatteren maakt dit onderzoek gebruik van een klein Java programma dat de dataset van 2015 inleest, de twee kolommen correct formatteert en uiteindelijk alles terug wegschrijft naar een csv-bestand. De code voor dit programma is te vinden in Bijage B.

Eens de structuur van de data geanalyseerd is, volgt de onboarding van de data. Hierbij wordt de volledige dataset opgeslagen in een bucket op Google Cloud Storage en Amazon S3, dit is goed voor ruim 9GB aan data.

Daarna volgt de aftoetsing van de requirements aan de drie verschillende oplossingen, hierbij wordt nog niet elk requirement ingevuld aangezien sommige pas gekend zijn na uitvoering van de queries (bv. het requirement rond performantie). De argumentatie en scores bij elk van deze requirements vindt u voor Google BigQuery in Sectie 4.2, voor Amazon EMR in Sectie 5.2 en voor Google Cloud Dataproc in Sectie 6.2. Op elk van deze requirements wordt een score gegeven van 1 tot 5 waarbij 1 betekent *geen tot weinig ondersteuning* en 5 betekent *volledige ondersteuning*. Deze score wordt bepaald in samenspraak met IntoData.

Daarna volgt het bedenken van de queries die uitgevoerd zullen worden op Google BigQuery, Amazon EMR en Google Cloud Dataproc. Hieronder vindt u 10 queries die uitgevoerd zullen worden met een korte beschrijving waarom er voor de query gekozen is.

1. In welke staat worden het meeste tickets uitgeschreven?
2. Welke inbreuk wordt het meeste vastgesteld?

3. Welk politiedistrict schrijft het meeste tickets uit?
4. Welke violation code komt overeen met welke violation description?
5. Top 10 van de staten waar het meest een ticket wordt uigeschreven
6. Top 10 van het aantal inbreuken per staat per automerk
7. Op welk uur van de dag worden het meeste tickets uitgeschreven?
8. Een overzicht van het aantal tickets per maand, opgesplitst in New York en andere staten
9. Maken mensen met een niet-ingeschreven voertuig meer inbreuken? Of net minder?
10. Op welke momenten van de dag gebeuren het meeste inbreuken?

Queries 1 t.e.m. 5 zorgen voor aggregaties op verschillende kolommen met verschillende soorten datatypes. Hierdoor vraagt de ene aggregatie meer werk dan de andere. Dit is zo wanneer bv. de kolom waarop geaggregeerd wordt tekst bevat, tekst vergelijken kost namelijk meer tijd dan getallen vergelijken. Query 6 zorgt voor aggregatie op meer dan één kolom met een ordening op een derde kolom. Query 7 bevat een transformatie van de kolom Violation Time naar een notatie van enkel het uur zonder de minuten, maar met 'A' of 'P' voor respectievelijk 'AM' en 'PM'. Query 8 bevat een transformatie van de kolom IssueDate naar enkel de maand, hierbij moet slechts een deel van de Issue Date genomen worde. Daarbij moet opgelet worden met het formaat van de datum aangezien de maand niet altijd als eerste staat, maar soms als tweede. Daarnaast komt er een extra transformatie bij van de kolom Violation County naar ofwel 'NY' ofwel 'OTHER'. Deze transformaties worden gecombineerd met een aggregatie op de eerste twee kolommen (maand en staat) en een ordening op alledrie de kolommen (maand, staat en aantal). Query 9 bevat een transformatie van de kolom Unregistrerd Vehicle? gecombineerd met een aggregatie en orderning op de tweede kolom (het aantal). Query 10 bevat een uitgebreide transformatie van de kolom Violation Time naar verschillende periodes doorheen de dag (nacht, ochtend, voormiddag, middag, namiddag en avond). Daarnaast moeten ook enkel verkeerde waardes voor de kolom Violation Time weggefilterd worden. Deze transformatie en filtering wordeng gecombineerd met een aggregatie op de eerste kolom (periode) en een ordening op de tweede (het aantal).

Daarna volgt het opzetten van Google BigQuery, Amazon EMR en Google Cloud Dataproc, hiervan vindt u extra informatie in respectievelijk Sectie 4.1, Sectie 5.1 en Sectie 6.1.

Uiteindelijk volgt het schrijven van elk van deze 10 queries in ofwel SQL (voor Google BigQuery) ofwel Python voor zowel Amazon EMR als Google Cloud Dataproc. Wanneer de queries geschreven zijn, worden ze op elk van de oplossing 100 keer uitgevoerd. Hierbij wordt de uitvoeringstijd van elke individuele query opgeslagen in een csv-bestand. Een belangrijke nuance bij dit onderzoek is dat de data ondergeschikt is aan het onderzoek, d.w.z. dat de conclusies uit de data volkomen irrelavant zijn voor de conclusie van dit onderzoek. De data dient enkel ter ondersteuning om de gekozen tools te kunnen testen en beoordelen.

Wanneer alle resultaten van de uitvoeringstijden binnen zijn, wordt per oplossing en per query het gemiddelde, de mediaan, de standaardafwijking en de variatiecoëfficiënt bepaald. Voor de scores worden dezelfde gegevens aangevuld met de modus. Deze resultaten vindt u in Hoofdstuk 7.

Daarna wordt een statische analyse gemaakt van de uitvoeringstijden en de scores op de requirements. Aan de hand van deze analyse wordt een antwoord geformuleerd op de hoofdonderzoeksvraag.

Uiteindelijk wordt een conclusie gevormd uit het gevoerde onderzoek.

3.1 Requirementsanalyse

Bij elke vergelijkende studie hoort een requirementsanalyse. Hieronder vindt u een oplijsting van alles requirements overeen gekomen in samenspraak met IntoData. Deze requirements zullen gebruikt worden om de verschillende gekozen oplossingen te beoordelen. De requirements zijn opgesplitst in functionele[1] en niet-functionele[2] requirements. Elk van deze requirements zal gebruikt worden om de tools te evalueren. Bij elke tool worden de requirements afgetoetst aan de mogelijkheden en wordt een score van 1 (slecht) tot 5 (uitstekend) gegeven.

- **Functionele requirements**
  - Ondersteuning voor verschillende bestandsformaten zoals JSON, XML... als invoer
  - Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data
  - Ondersteuning voor polyglot persistence[3]
  - Ondersteuning voor pipelines voor de data onboarding[4]
  - Ondersteuning voor datastreams (real-time dataverwerking)
  - Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR[5])
  - Ondersteuning voor management en monitoring van de oplossing
  - Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)
- **Niet-functionele requirements**
  - Moet (relatief) goedkoop zijn in implementatie en gebruik
  - Moet performant zijn (snel data inlezen en resultaat van een query geven)
  - Moet flexibel zijn voor up- en downscaling
  - Moet (relatief) eenvoudig te gebruiken zijn (korte leercurve)
  - Moet onderhoudbaar zijn op lange termijn

Wanneer men bovenstaande requirements indeelt volgens de MoSCoW-techniek (Consortium, 2008) komen we onderstaande verdeling uit. De 'Must have'-requirements moeten verplicht aanwezig zijn in de gekozen oplossingen. De 'Should have'-requirements zijn niet doorslaggevend voor de keuze van een oplossingen, maar zijn liefst wel aanwezig. De

[1] Een gewenste eigenschap/functionaliteit of gewenst gedrag van een systeem
[2] Een bepaalde kwaliteitseis voor het systeem
[3] De mogelijkheid om om te gaan met verschillende soort van persistentie (SQL, NoSQL...)
[4] Het transporteren van data van een offline omgeving naar een online omgeving(Adam, 2017)
[5] Meer info over GDPR is te vinden op de website van GPDR-EU

'Could have'-requirements zijn optioneel, ze mogen dus genegeerd worden bij de keuze van de oplossingen.

- **Must have**
  - Ondersteuning voor verschillende bestandsformaten zoals JSON, CSV, XML... als invoer
  - Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data
  - Ondersteuning voor polyglot persistence[6]
  - Ondersteuning voor pipelines voor de data onboarding
  - Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR)
  - Moet (relatief) goedkoop zijn in implementatie en gebruik
  - Moet performant zijn (snel data inlezen en resultaat van een query geven)
  - Moet flexibel zijn voor up- en downscaling
  - Moet (relatief) eenvoudig te gebruiken zijn (korte leercurve)
  - Moet onderhoudbaar zijn op lange termijn

- **Should have**
  - Ondersteuning voor datastreams (real-time dataverwerking)

- **Could have (nice to have)**
  - Ondersteuning voor management en monitoring van de oplossing
  - Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)

[6] Een applicatie die het toelaat om met verschillende soorten databanken te communiceren.

4. Google BigQuery

> "Een snel, zuinig en volledig beheerd datawarehouse voor grootschalige gegevensanalyses"– Google

Als eerste wordt in deze vergelijkende studie dieper ingegaan op Google BigQuery. Deze vergelijkende studie zal dieper ingaan op de verschillende mogelijkheden van Google BigQuery. Daarvoor worden in dit onderzoek tien queries losgelaten op Google BigQuery en zullen alle mogelijkheden afgetoetst worden aan de requirements van IntoData, opgegeven in Sectie 3.1.

4.1 Google BigQuery instellen

Vooraleer men van start kan gaan met een analyse (of query) moet men eerst Google BigQuery instellen. Voor Google BigQuery heeft men sowieso een Google account nodig, het aanmaken van dit account wordt niet verder behandeld in dit onderzoek. Daarnaast heeft men ook een dataset nodig waarop men de queries kan toepassen. Voor deze dataset is er natuurlijk data nodig die ergens opgeslagen staat. Deze data kan simpelweg geüpload worden naar Google BigQuery of kan opgeslagen worden op Google Cloud Storage, Google Drive en Google Cloud Bigtable. Indien het databestand groter is dan 10MB, dan moet men uitkijken naar één van de laatste drie mogelijkheden om het bestand op te slaan, de web interface van Google BigQuery staat geen bestanden groter dan 10MB toe. Indien het bestand opgeslagen is op één van deze drie mogelijkheden, geeft men simpelweg de url naar het bestand op bij het inladen van de data in Google BigQuery. In dit onderzoek wordt de data geüpload naar Google Cloud Storage waarna het geïmporteerd wordt in Google BigQuery.

4.1.1 Data uploaden naar Google Cloud Storage

De eerste stap in het onderzoek naar Google BigQuery is het uploaden van de data naar Google Cloud Storage. Hiervoor volgt u onderstaand stappenplan. Het stappenplan gaat ervan uit dat er een gebruiker ingelogd is op de Google Cloud Console (https://console.cloud.google.com/).

1. Ga naar de console van Google Cloud Storage: https://console.cloud.google.com/storage/browser
2. Maak een nieuwe bucket, gebruik daarbij onderstaande instellingen:
   - **Naam:** kies zelf een unieke naam voor de bucket
   - **Standaardopslagklasse:** Regionaal
   - **Locatie:** europe-west2
3. Klik op *Bestanden uploaden* en selecteer alle bestanden die in de bucket moeten komen
4. Wacht tot het uploaden voltooid is

4.1.2 Dataset aanmaken op Google BigQuery

1. Ga naar het dashboard van Google Cloud Platform: https://console.cloud.google.com/home/dashboard
2. Klik bovenaan naast *Google Cloud Platform* op de lijst, er opent nu een scherm bovenop de pagina
3. Klik op het plus-icoon om een nieuw project aan te maken
4. Kies een naam en klik op *Maken*
5. Als het nieuwe project niet geselecteerd is, selecteer dit dan uit het lijstje links boven
6. Open BigQuery vanuit het hamburgermenu links boven
7. Klik in het menu links op het pijltje naast de naam van het aangemaakte project
8. Klik dan op *Create new dataset*
9. Kies een naam en klik op *OK*
10. Klik op het plus-icoon naast de naam van de dataset
11. Vul onderstaande gegevens in:
    - **Location:** kies Google Cloud Storage uit de lijst, neem als url: gs://*mijn-bucket-naam/mijn-bestand*
    - **File format:** CSV (of ander formaat indien nodig)
    - **Table name:** kies zelf een tabelnaam
    - Vul het schema in volgens de opbouw van het csv-bestand, het schema van de NYC Parking Tickets dataset is hieronder te vinden.
    - **Field delimiter:** Comma
    - **Header rows to skip**: 1
    - **Allow jagged rows:** aanvinken
    - **Ignore unknown values:** aanvinken
    - **Write preference:** Append to table selecteren uit de lijst (zo wordt bestaande data niet overschreven)
12. Klik op *Create table*

Schema NYC Parking Tickets

```
SummonsNumber: INTEGER,
```

```
PlateID:STRING,
RegistrationState: STRING,
PlateType: STRING,
IssueDate: STRING,
ViolationCode: INTEGER,
VehicleBodyType: STRING,
VehicleMake: STRING,
IssuingAgency: STRING,
StreetCode1: INTEGER,
StreetCode2: INTEGER,
StreetCode3: INTEGER,
VehicleExpirationDate: INTEGER,
ViolationLocation: STRING,
ViolationPrecinct: INTEGER,
IssuerPrecinct: INTEGER,
IssuerCode: INTEGER,
IssuerCommand: STRING,
IssuerSquad: STRING,
ViolationTime: STRING,
TimeFirstObserved: STRING,
ViolationCounty: STRING,
ViolationInFrontOfOrOpposite: STRING,
HouseNumber: STRING,
StreetName: STRING,
IntersectingStreet: STRING,
DateFirstObserved: INTEGER,
LawSection: INTEGER,
SubDivision: STRING,
ViolationLegalCode: STRING,
DaysParkingInEffect: STRING,
FromHoursInEffect: STRING,
ToHoursInEffect: STRING,
VehicleColor: STRING,
UnregisteredVehicle: STRING,
VehicleYear: INTEGER,
MeterNumber: STRING,
FeetFromCurb: INTEGER,
ViolationPostCode: STRING,
ViolationDescription: STRING,
NoStandingorStoppingViolation: STRING,
HydrantViolation: STRING,
DoubleParkingViolation: STRING
```

4.2 Requirements

Voor dit onderzoek zijn een aantal requirements opgesteld waaraan o.a. Google BigQuery moet voldoen (zie Sectie 3.1). Hieronder wordt elk van deze requirements afgetoetst aan de mogelijkheden van Google BigQuery. Deze requirements werden reeds opgesplitst in

Nice to have, *Should have* en *Could have*. Deze onderverdeling zal hieronder ook gebruikt worden om de oplossing te beoordelen.

4.2.1 Nice to have

Ondersteuning voor verschillende bestandsformaten zoals JSON, CSV, XML... als invoer

Google BigQuery laat het toe om data te importeren vanuit verschillende formaten. Men biedt ondersteuning voor CSV, JSON (Newline Delimited), Apache Avro, Apache Parquet, Cloud Datastore Backup, Google Sheets en Cloud Bigtable. Uit de voorgaande opsomming blijkt dat Google BigQuery veel mogelijkheden qua formaten aanbiedt, maar één nog vaak gebruikt formaat ontbreekt: XML. Dit kan een klein minpuntje zijn op het gebied van formaten. Niet alle opgegeven formaten zijn altijd beschikbaar, hieronder wordt opgelijst in welke gevallen welke formaten beschikbaar zijn.

- Bestand uploaden: CSV, JSON (Newline Delimited), Apache Avro en Apache Parquet
- Google Cloud Storage URL: CSV, JSON (Newline Delimited), Apache Avro, Apache Parquet en Cloud Datastore Backup
- Google Drive: CSV, JSON (Newline Delimited), Apache Avro en Google Sheets
- Google Cloud Bigtable: Cloud Bigtable

In wat volgt vindt u een korte omschrijving van elk van de aangeboden formaten.

- CSV: een bestand waarin alle kolommen door een komma (,) of punt-komma (;) gescheiden worden. Vaak bevat de eerste lijn van dit bestand de hoofdingen van de kolommen in dit bestand.
- JSON (Newline Delimited): Een bestand waarin elk JSON-object gescheiden wordt door een nieuwe lijn (\n op het einde van elk JSON-object). Een JSON-object neemt ook telkens een volledige lijn in, het is dus niet mooi geformatteerd om leesbaar te zijn voor mensen.
- Apache Avro: een framework dat data serialiseert naar een binair formaat gebruik makend van een schema beschreven in JSON. Het is ontwikkeld in het kader van hun Hadoop project. Meer informatie is te vinden in de documentatie van Apache Avro[1].
- Apache Parquet: een kolomgebasseerde opslag, ook ontwikkeld door Apache in het kader van hun Hadoop project. Meer informatie is te vinden in de documentatie van Apache Avro[2].
- Cloud Datastore Backup: Cloud Datastore is een NoSQL databank ontwikkeld door Google met als bedoeling automatisch schaalbaar te zijn met een hoge performantie en makkelijke implementatie. Een Cloud Datastore Backup is dan een backup van een databank op de Google Cloud Storage omgeving. Meer informatie is te vinden

[1] http://avro.apache.org/docs/current/
[2] https://parquet.apache.org/documentation/latest/

in de documentatie van Cloud Datastore Avro[3].
- Google Sheets: het Excel-equivalent van Google
- Cloud Bigtable: een andere NoSQL databank ontwikkeld door Google en bedoeld voor grote analytische opdrachten. Meer informatie is te vinden in de documentatie van Cloud Datastore Avro[4].

Uit de voorgaande bevindingen wordt een score van 4 gegeven aan dit requirement.

Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data

Google BigQuery biedt drie mogelijkheid om data te importeren in een tabel:

- **Write if empty**: Hierbij wordt de data enkel weggeschreven indien de tabel leeg is. Anders wordt een fout van duplicate gegevens gegooid.
- **Append to table**: Als de tabel reeds bestaat, wordt de data aan de tabel toegevoegd zonder de reeds bestaande data te wijzigen.
- **Overwrite table**: Als de tabel reeds bestaat, wordt de reeds bestaande data overschreven.

Hieruit blijkt dat Google BigQuery goed scoort op het gebied van uitbreiding van reeds bestaande data, daarom krijgt dit requirement een score van 4.

Ondersteuning voor polyglot persistence

Op dit requirement scoort Google BigQuery heel slecht. Er is enkel ondersteuning voor datasets in Google BigQuery zelf, geen datasets in andere soorten databanken. Daarbij kan er gebruik gemaakt worden van private datasets (jouw eigen datasets) of publieke datasets (vrij beschikbaar voor iedereen). Er is wel een mogelijkheid om met Google BigQuery Data Transfer Service[5] data te importeren van een Software as a Service omgeving[6] zoals Adwords, DoubleClick Campaign Manager, DoubleClick for Publishers en YouTube. Hierbij wordt de data van deze SaaS-omgevingen geïmporteerd in een Google BigQuery dataset. Dus wederom is er geen ondersteuning voor meerdere soorten databanken, men biedt enkel de mogelijkheid om data in verschillende formaten en van een andere locatie te importeren.

Door de hierboven genoemde redenen krijgt Google BigQuery voor dit requirement een score van 1.

[3] https://cloud.google.com/datastore/docs/concepts/overview

[4] https://cloud.google.com/bigtable/

[5] https://cloud.google.com/bigquery/transfer/

[6] Software as a Service is beter gekend als SaaS. Volgens Nucleus (2017) biedt SaaS de mogelijkheid om software in de cloud te draaien zonder deze op je eigen toestel te hoeven installeren.

Ondersteuning voor pipelines voor de data onboarding

Volgens Lardinois (2013) biedt Google BigQuery de mogelijkheid tot het streamen van data naar een tabel in een dataset. Daarbij maakt men gebruik van de API call **insertAll** voor een tabel. Om de data te streamen doet men een geauthenticeerde HTTP POST op onderstaande url.

https://www.googleapis.com/bigquery/v2/projects/**projectId**/datasets/**datasetId**/tables/**tableId**/insertAll

In de body van het request plaats je volgende structuur in JSON formaat, in de *rows*-property plaatst men de data die aan de tabel toegevoegd moet worden Het *insertId*-property is optioneel, het *json*-property is verplicht. Binnen het *json*-property van het *rows*-property, plaatst men alle kolommen van de tabel als key - value pair waarbij de key de naam van de kolom van de tabel is en de value de waarde in de kolom. Daarnaast kan er aangegeven worden om ongeldige/onvolledige rijen over te slaan met het *skipInvalidRows*-property, om onbekende waarden te negeren met het *ignoreUnknownValues*-property of om een suffix toe te voegen aan de tabel met het *templateSuffix*-property om zo de data van verschillende requests te scheiden. De laatste drie opgesomde properties zijn optioneel.

```
{
   "kind": "bigquery#tableDataInsertAllRequest",
   "skipInvalidRows": boolean,
   "ignoreUnknownValues": boolean,
   "templateSuffix": string,
   "rows": [
      {
         "insertId": string,
         "json": {
            (key): (value)
         }
      }
   ]
}
```

Als alles goed gaat, stuurt Google BigQuery onderstaande JSON terug. Mogelijke fouten met de reden, locatie en andere informatie worden in het *insertErrors*-property weergegeven.

```
{
   "kind": "bigquery#tableDataInsertAllResponse",
   "insertErrors": [
      {
         "index": unsigned integer,
         "errors": [
            {
               "reason": string,
               "location": string,
               "debugInfo": string,
               "message": string
            }
```

```
            ]
        }
    ]
}
```

Daarnaast bestaat er ook de mogelijkheid tot integratie met verschillende soorten services die pipelines aanbieden. Eén van deze tools is Alooma, een Data Pipeline as a Service, die de mogelijkheid biedt om data te verplaatsen en te transformeren vanop eender welke bron. Een andere gelijkaardige tool is Blendo. Daarnaast bestaan er nog heel veel andere mogelijkheden en integraties om data vanop verschillende locaties naar Google BigQuery te brengen. De volledige lijst vindt u in het requirement rond integraties verderop in dit Hoofdstuk.

Hierdoor krijgt Google BigQuery een 4 voor dit requirement.

Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR)

Google BigQuery werkt volgens de standaard van het OAuth 2.0 protocol. Bijgevolg is er de mogelijkheid om bepaalde gebruikers een bepaalde scope toe te kennen. In het onderdeel *IAM en beheer*[7] van de Google Cloud kan je beheren welke gebruikers toegang hebben tot een project in de Google Cloud en dus ook tot BigQuery. Tabel 4.1 geeft alle scopes weer met een korte omschrijving. Elke scope wordt voorafgegaan door *https://www.googleapis.com*, dit is weggelaten voor de leesbaarheid. BigQuery biedt de mogelijkheid om bepaalde e-mailadressen toe te voegen die gekoppeld zijn aan een actief Google- of een Google Apps-account. Alle scopes zijn ook te vinden op volgende link: https://developers.google.com/identity/protocols/googlescopes#bigqueryv2.

Scope	**Omschrijving**
/auth/bigquery	Bekijk en beheer data in Google BigQuery
/auth/bigquery.insertdata	Voeg data toe aan Google BigQuery
/auth/cloud-platform	Bekijk en beheer data over de Google Cloud Platform services
/auth/cloud-platform.read-only	Bekijk data over Google Cloud Platform services
/auth/devstorage.full_control	Beheer data en permissies in Google Cloud Storage
/auth/devstorage.read_only	Bekijk data in Google Cloud Storage
/auth/devstorage.read_write	Beheer data in Google Cloud Storage

Tabel 4.1: Scopes authenticatie Google BigQuery

Hierdoor krijgt Google BigQuery een score van 5 op dit requirement.

[7] https://console.developers.google.com/iam-admin/iam

Moet (relatief) goedkoop zijn in implementatie en gebruik

Een vereenvoudigd overzicht van de kosten van Google BigQuery vindt u in tabel 4.2. Daaruit blijkt dat het implementeren, gebruiken en onderhouden van Google BigQuery heel weinig kost, zeker met de quota die BigQuery gratis aanbiedt. Meer informatie is te vinden in de documentatie van Google BigQuery: https://cloud.google.com/bigquery/pricing.

Actie	Prijs	Extra info
Opslag	$0,02 per GB, per maand	De eerste 10GB is gratis iedere maand
Lange termijn opslag	$0,01 per GB, per maand	De eerste 10GB is gratis iedere maand
Streams	$0,01 per 200MB succesvol toegevoegd	Minimum grootte van de rij is 1KB
Queries	$5 per TB	Eerste 1TB per maand is gratis
Data inladen	Gratis	
Data kopiëren	Gratis	
Data exporteren	Gratis	
Metadata operaties	Gratis	List, get, patch, update en delete operaties

Tabel 4.2: Overzicht kosten Google BigQuery

Wanneer met de Google Cloud Platform Pricing Calculator[8] een kostenraming gemaakt wordt voor de huidige configuratie van de servers en opslag in dit onderzoek (zoals hieronder weergegeven), komt dit op een bedrag van $15,79 per maand (ongeveer €13). Dit kan opgesplitst worden in $10,18 voor Google BigQuery en $5,61 voor Google Cloud Storage (het opslaan van de csv-bestanden met de data). Dit is veel goedkoper in vergelijking met de andere oplossingen (zie Hoofdstuk 5 en 6).

Configuratie calculator

- Google BigQuery
  - Tabelnaam: nyc_parkings
  - Opslag: 9GB
  - Queries: 3 TB (willekeurig gekozen)
- Cloud Storage
  - Regio: europe-west2
  - Klasse: regionaal
  - Regionale opslag: 9GB
  - Aantal klasse A requests: 1 miljoen (willekeurig gekozen)
  - Aantal klasse B requests: 1 miljoen (willekeurig gekozen)

Door bovenstaande redenen en het voorbeeld in de kostenberekening, krijgt Google Big-

[8] https://cloud.google.com/products/calculator

Query een score van 5 voor dit requirement.

Moet performant zijn (snel data inlezen en resultaat van een query geven)

Google BigQuery is een een gigantisch platform met heel veel mogelijkheden waarin performantie een impressionante rol speelt. De snelheid waarmee Google BigQuery data inleest, verwerkt en persisteert in een dataset is nog lang niet zo indrukwekkend als de snelheid waarmee een query uitgevoerd wordt. Het importeren van de csv-bestanden vanop Google Cloud Storage duurde telkens tussen twee en drie minuten. Hierbij is het belangrijk om te weten dat de vier datasets een grootte hebben van ongeveer 2GB, dit maakt de tijd van het importeren heel indrukwekkend. Over de uitvoeringstijden van de queries vindt u hieronder meer informatie.

Bijgevolg krijgt Google BigQuery voor dit requirement een score van 5.

Moet flexibel zijn voor up- en downscaling

Het grote voordeel aan Google BigQuery is het ontbreken van de zorgen rond up- en downscaling. Google BigQuery zorgt zelf voor het uitbreiden van alle nodige resources zodat een query uitgevoerd kan worden of zodat een dataset gepersisteerd kan worden. Op het eerste zicht lijkt dit een beperkende factor, dit is ook wel een beetje zo op een bepaald gebied. Google BigQuery laat het niet meer toe om een bestaande dataset/tabel aan te passen. Eens je gegevens hebt geïmporteerd kunnen de instellingen niet meer gewijzigd worden. Er is enkel nog de mogelijkheid om een aantal basisinstellingen te wijzigen zoals het schema van de tabel, de omschrijving, de vervaldatum.... Dit maakt natuurlijk dat er op dat gebied een beetje lock-in optreedt. Het is ook niet mogelijk om de instellingen van een tabel aan te passen bij het uploaden of inladen van nieuwe data in een tabel van een dataset. Indien men de partitioneringsinstellingen wil aanpassen, moet men de oude tabel verwijderen en opnieuw aanmaken.

Om de hierboven genoemde redenen krijgt Google BigQuery hiervoor een score van 3.

Moet (relatief) eenvoudig te gebruiken zijn (korte leercurce)

Dit is opnieuw een requirement waarop Google BigQuery heel goed scoort. De web interface is makkelijk te gebruiken voor een nieuwe gebruiker en vraagt niet veel kennis. Daarnaast bestaat er ook heel goede documentatie over elke mogelijke taak in Google BigQuery of over elk mogelijk probleem. Maar zelfs zonder deze documentatie kan er heel veel bereikt worden door een beetje rond te dwalen en te proberen op het platform. Men hoeft zelfs geen eigen data te importeren indien men Google BigQuery wil uittesten. Er bestaan op het moment van dit onderzoek negen publieke datasets die vrij te gebruiken zijn. De groottes van deze datasets gaan van enkele 100MB tot 130GB (taxiritten van gele taxi's in New York).

Hierdoor krijgt Google BigQuery een score van 5 op dit requirement.

Moet onderhoudbaar zijn op lange termijn

Google BigQuery biedt een goede onderhoudbaarheid op lange termijn. Zoals reeds eerder gezegd bestaat er de mogelijkheid om bij het uploaden van data via een API call een suffix op te geven die de dataset over verschillende tabellen verdeelt. Hierdoor blijft de data goed gestructureerd en onderhoudbaar op lange termijn. Ook bestaat er de mogelijkheid om een vervaldatum in te stellen voor een tabel van een dataset. Hierdoor is het eenvoudig om data automatisch te laten verwijderen na een bepaalde tijd. Daarnaast bestaat er de mogelijkheid om bepaalde jobs (uploaden van data) te herhalen wanneer een nieuw bestand geüpload of ingeladen moet worden, dit bespaart enorm veel werk op lange termijn zodat niet steeds het volledige configureren van de job opnieuw gedaan hoeft te worden. Hierbij kan men eenvoudig alle instellingen van de job aanpassen en opnieuw uitvoeren. Daarna wordt er opnieuw een job toegevoegd aan de geschiedenis die men dan eventueel later kan hergebruiken. Als laatste bestaat er de mogelijkheid om queries op te slaan om later te hergebruiken. Zo hoeft men deze niet ergens anders op te slaan (in een tekstbestand bijvoorbeeld). Dit menu staat wel iets dieper in het systeem. Hiervoor dient men te klikken op *Query History* in het menu links op het dashboard van Google BigQuery, daarna klikt men op *Saved Queries* bovenaan.

Voor dit requirement krijgt Google BigQuery een score van 4.

4.2.2 Should have

Ondersteuning voor datastreams (real-time dataverwerking)

Zoals reeds eerder aangegeven in het requirement rond pipelines voor data onboarding is er ondersteuning voor streams van data. Daarom wordt hierop niet dieper ingegaan.

De score voor dit requirement is dus 4, net zoals bij het gelijkaardig requirement.

4.2.3 Could have

Ondersteuning voor management en monitoring van het cloudplatform

Ook op dit gebied komt Google BigQuery heel goed uit de test. Google biedt enorm veel monitoringsmogelijkheden op de console van de Google Cloud[9]. Dit gaat van een uitgebreid overzicht van de facturering tot een overzicht van alle activiteit op een project in de Google Cloud met gedetailleerde informatie over de gebruiker en wat deze precies deed. Ook is er een handig overzicht van alle services die eventueel uitgevallen zouden zijn. Daarnaast is er ook een mooi overzicht van het laatste nieuws rond functionaliteiten van het platform.

Hierdoor krijgt Google BigQuery een score van 4 op dit requirement.

[9] https://console.cloud.google.com

Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)

Google BigQuery biedt heel veel mogelijkheden qua integratie met tools van derde partijen. Er bestaat de mogelijkheid voor integraties op drie verschillende gebieden: data integratie, data analytics/visualisatie en developer tools.

Qua data integratie is er ondersteuning voor Alooma, Blendo, Fivetran, Funnel, Atomdata, Informatica, Keen IO, Lytics, Matillion, SAP Data Services, Segment, snapLogic, Software AG, Stitch, Striim, Switchboard, Skyvia, Talend, Treasure data en Xplenty. Qua data analytics en visualisatie is er ondersteuning voor Anodot, Arimo, AtScale, BIME, Chartio, CoolaData, Domo, IBM Cognos Analytics, iCharts, Indicative, Dundas BI, Looker, Metabase, Mode, MicroStrategy, OWOX, QlikView, re:dash, SAP Analytics Cloud, Tableau, Yellowfin en Zoomdata. Qua developer tools is er ondersteuning voor CData Software, Fluentd en Simba.

Meer informatie over deze tools is te vinden in de documentatie van Google BigQuery: https://cloud.google.com/bigquery/partners/. Zoals uit bovenstaande opsommingen blijkt, bestaan er heel veel integratiemogelijkheden in Google BigQuery.

Daarom krijgt BigQuery een score van 5 voor dit requirement.

Dit was het laatste requirement waarop Google BigQuery geëvalueerd zou worden. Daarom vindt u in tabel 4.3 een overzicht van de scores per requirement alsook een uiteindelijke eindscore (gemiddelde van alle scores). Voor de duidelijkheid werden de rijen gekleurd per soort requirement: must have, should have en could have.

4.3 Queries

Voor dit onderzoek zijn een aantal queries bedacht (zie Hoofdstuk 3) die uitgevoerd zullen worden op Google BigQuery. Google BigQuery laat het toe om queries uit te voeren in zowel SQL als in JavaScript. Het deel JavaScript wordt in dit onderzoek niet verder behandeld aangezien dit geen meerwaarde bood bovenop de SQL queries in dit onderzoek. SQL queries schrijven voor BigQuery is zeer eenvoudig. Men schrijft de query zoals men dit gewoon is in normale SQL waarna zelden nog een aanpassing gedaan hoeft te worden vooraleer deze ook uitgevoerd kan worden op Google BigQuery. Een voorbeeld van een aanpassing is het TOP-commando. In gewone SQL schrijft men *SELECT TOP 1 kolom* met GROUP BY en ORDER BY, maar in Google BigQuery is dit simpelweg *SELECT TOP(kolom, 1)* zonder GROUP BY en ORDER BY. Alle informatie rond SQL in BigQuery (zowel standard als legacy SQL) is te vinden op https://cloud.google.com/bigquery/docs/reference/standard-sql/.

De code van alle queries vindt u in bijlage D. Daarbij is in commentaar aangegeven waar elke query staat.
Alle resultaten en de verwerking van de uitvoeringstijden is te vinden in Hoofdstuk 7. Een kleine nuance bij de uitvoeringstijden van Google BigQuery is dat de uitvoeringstijden

Requirement	Score (op 5)
Ondersteuning voor verschillende bestandsformaten zoals JSON, CSV, XML... als invoer	4
Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data	4
Ondersteuning voor polyglot persistence	1
Ondersteuning voor pipelines voor de data onboarding	4
Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR)	5
Moet (relatief) goedkoop zijn in implementatie en gebruik	5
Moet performant zijn (snel data inlezen en resultaat van een query geven)	5
Moet flexibel zijn voor up- en downscaling	3
Moet (relatief) eenvoudig te gebruiken zijn (korte leercurce)	5
Moet onderhoudbaar zijn op lange termijn	4
Ondersteuning voor datastreams (real-time dataverwerking)	4
Ondersteuning voor management en monitoring van het cloudplatform	4
Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)	5
Gemiddelde	4
Mediaan	4
Modus	4
Standaardafwijking	1,12
Variatiecoëfficiënt	0,27

Tabel 4.3: Overzicht score Google BigQuery

bepaald worden door het aantal queries dat door andere gebruikers tegelijk naar Google BigQuery gestuurd worden.

5. Amazon EMR

> "Easily Run and Scale Apache Hadoop, Spark, HBase, Presto, Hive, and other Big Data Frameworks" – Amazon

De tweede oplossing waarop in deze vergelijkende studie dieper ingegaan wordt is Amazon EMR. Deze vergelijkende studie zal dieper ingaan op de verschillende mogelijkheden van Amazon EMR. Daarvoor worden in dit onderzoek tien queries losgelaten op Amazon EMR en zullen alle mogelijkheden afgetoetst worden aan de requirements van IntoData, opgegeven in Sectie 3.1.

5.1 Amazon EMR instellen

Alvorens van start te kunnen gaan met het uitvoeren van analyses is het nodig om een cluster aan te maken op Amazon EMR. Hiervoor gebruikt u onderstaand commando. Daarin vervangt u <*cluster-name*> door de naam van de cluster, <*key-pair-name*> door de naam van een Amazon EC2 Key Pair, <*subnet-id*> door het id van een subnet en <*your-s3-bucket*> door de naam van een S3 bucket.

```
1 aws emr create-cluster --release-label emr-5.13.0 \
2    --name '<cluster-name>' \
3    --applications Name=Hadoop Name=Spark \
4    --ec2-attributes KeyName=<key-pair-name>,SubnetId=<subnet-
        id>,InstanceProfile=EMR_EC2_DefaultRole \
5    --service-role EMR_DefaultRole \
6    --instance-groups \
```

```
7     InstanceGroupType=MASTER,InstanceCount=1,InstanceType=m4.
         xlarge \
8     InstanceGroupType=CORE,InstanceCount=2,InstanceType=m4.
         xlarge \
9     --region eu-west-2 \
10    --log-uri s3://<your-s3-bucket>/emr-logs/ \
11    --bootstrap-actions \
12    Name='Install Jupyter notebook',Path="s3://install-jupiter-
         london/install-jupyter-emr5.sh",Args=[--r,--julia,--
         toree,--torch,--ruby,--ds-packages,--ml-packages,--
         python-packages,'ggplot nilearn',--port,8885,--password,
         jupyter,--jupyterhub,--jupyterhub-port,8005,--cached-
         install,--notebook-dir,s3://bachelorproef-logs/notebooks
         /,--copy-samples]
```

Daarna kan men eenvoudig verbinding maken met de master node van de cluster door onderstaand commando uit te voeren. Hiervoor is het wel nodig om de AWS CLI te installeren, instructies hiervoor kan u vinden op https://docs.aws.amazon.com/cli/latest/userguide/installing.html. Hierin vervangt u <*ssh-key-location*> door de locatie van het pem-bestand dat u verkregen hebt bij het aanmaken van een Amazon EC2 Key Pair en <*master-public-dns*> door het publiek adres van de master node van de aangemaakte cluster (te vinden in de details van de cluster).

```
1 ssh -i <ssh-key-location> <master-public-dns>
```

Daarna kan u door het commando *pyspark* te typen een console openen voor Spark in Python, daarmee kan u eenvoudig kleine stukjes code uittesten alvorens deze in een step te laten draaien op de cluster.

5.2 Requirements

Voor dit onderzoek zijn een aantal requirements opgesteld waaraan o.a. Amazon EMR moet voldoen (zie Sectie 3.1). Hieronder wordt elk van deze requirements afgetoetst aan de mogelijkheden van Amazon EMR. Deze requirements werden reeds opgesplitst in *Nice to have*, *Should have* en *Could have*. Deze onderverdeling zal hieronder ook gebruikt worden om de tool te beoordelen.

5.2.1 Nice to have

Ondersteuning voor verschillende bestandsformaten zoals JSON, CSV, XML... als invoer

Volgens Intellipaat (g.d.) is er voor verschillende soorten bestandsformaten zoals simpele tekst, JSON, CSV, XML, Object files... ondersteuning, alsook voor verschillende soorten databanken zoals Apache Hive, Cassandra, HBase en Elasticsearch. Voor deze laatste

optie is er wel enige configuratie nodig specifiek voor de gebruikte programmeertaal, in het geval van Java is dit JDBC (Java DataBase Connectivy).

Hierdoor krijgt Amazon EMR een score van 4 voor dit requirement.

Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data

Bij Amazon EMR bestaat de mogelijkheid om data te importen vanop verschillende locaties waaronder bijvoorbeeld Amazon S3, Amazon DynamoDB.... Dit laat het toe om eenvoudig data toe te voegen aan reeds bestaande datasets. Men voegt gewoon de nieuwe data toe op de locatie waar deze is opgeslagen en importeert deze in Amazon EMR. Op het eerste zicht lijkt dit eenvoudig, maar in de realiteit vergt dit wel enige configuratie vooraleer dit volledig goed werkt.

Daarom krijgt Amazon EMR een score van 4 voor dit requirement.

Ondersteuning voor polyglot persistence

Volgens Amazon (2018a) is er ondersteuning voor verschillende soorten opslag waaronder Amazon S3, Hadoop Distributed Files System, Amazon DynamoDB, Amazon Redshift, Amazon Glacier en Amazon Relational Database Service.

Dit bewijst het feit dat er keuze genoeg is qua databanken en opslag om de input van te halen en de output naar weg te schrijven.

Hierdoor krijgt Amazon EMR een score van 5 voor dit requirement.

Ondersteuning voor pipelines voor de data onboarding

Op dit requirement scoort Amazon EMR ook goed. Om pipelines mogelijk te maken biedt Amazon AWS Data Pipeline[1]. Deze service laat het toe om data vanop verschillende locaties (Amazon S3, Amazon Redshift, MySQL databank...) te transformeren en op te slaan in Amazon S3, Amazon RDS, Amazon DynamoDB en Amazon EMR. AWS Data Pipeline is fout tolerant en hoog beschikbaar, zo hoeft de gebruiker zich geen zorgen te maken over de beschikbaarheid van de data en eventuele fouten tijdens het transformeren en opslaan van de data.

Daarom krijgt Amazon EMR een score van 5 voor dit requirement.

Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR)

Amazon beschikt over een volledige IAM console om de authenticatie en authorisatie in te stellen per gebruiker. Dit laat het toe om bepaalde gebruikers bepaalde acties te

[1] https://aws.amazon.com/datapipeline/

weigeren of net toe te staan. Amazon heeft een heel uitgebreid scala aan mogelijke permissies/rollen voor gebruikers. Daarnaast is er ook ondersteuning voor Kerberos en SSH. Naast de mogelijkheden qua authenticatie en authorisatie bestaat er de mogelijkheid om data te encrypteren om zo opgeslagen data of dataverkeer te beveiligen. Als laatste zijn er nog security groups in Amazon EMR. Deze dienen als een virtuele firewall voor Amazon EMR cluster instanties. Hiermee kan men dus het inkomend en uitgaand verkeer limiteren en beheren. Meer informatie is te vinden op volgende link: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-security.html.

Hierdoor krijgt Amazon EMR een score van 5 op dit requirement.

Moet (relatief) goedkoop zijn in implementatie en gebruik

Op dit gebied scoort Amazon EMR het slechts van de drie oplossingen die getest zijn. De AWS Simply Monthly Calculator[2] is niet zou makkelijk in het gebruik als de variant van Google. Wanneer in deze calculator enkel het aantal EC2 instances in Londen gebruikt in dit onderzoek ingegeven wordt, dus zonder de opslag op S3, komt dit op een totaal van $509,49 per maand indien ze ook alle dagen van de maand gebruikt worden (100% gebruik gedurende één maand). Voor de opslag van de 9GB data op Amazon S3 betaalt men $5,94 (ook voor 1 000 000 requests per type).

Daarom scoort Amazon EMR een 1 op dit requirement.

Moet performant zijn (snel data inlezen en resultaat van een query geven)

Tijdens het uitvoeren van de queries werd al snel duidelijk dat Apache Spark, waarop de Amazon EMR cluster in dit onderzoek draait, vele malen trager werkt dan Google BigQuery. Dit blijkt ook uit de resultaten te zien in Hoofdstuk 7. Het duurt meer dan 5 minuten alvorens de data ingeladen is in een RDD waarna het vaak nog eens even lang duurt alvorens een query volledig is uitgevoerd. Dit blijkt ook uit de resultaten verderop in dit Hoofdstuk (zie Sectie 5.3).

Hierdoor krijgt Amazon EMR een score van 1.

Moet flexibel zijn voor up- en downscaling

Amazon EMR biedt ondersteuning voor zowel automatisch als handmatig schalen van een cluster. Daarbij is het wel belangrijk om te weten dat enkel het aantal nodes in de Apache Spark cluster uitgebreid kan worden en niet het type node. Daarbij is het dus niet mogelijk om bijvoorbeeld een opschaling te doen naar een type node met meer processors en geheugen, enkel het aantal instanties van een type node kunnen omhoog of omlaag gehaald worden. Bij het automatisch schalen kunnen een aantal voorwaarden ingevuld worden die eerst behaald moeten worden vooraleer het aantal instanties verhoogd of verlaagd wordt. Voor het ontbreken van de mogelijkheid tot het aanpassen van de specificaties van

[2] https://calculator.s3.amazonaws.com/index.html

de nodes bestaat wel een logische verklaring. Indien men dit wenst te doen, moet men een volledig nieuwe cluster alloceren in Amazon EC2 en zodoende een nieuwe server opstarten. Na deze operatie moet de oude cluster verwijderd worden, waarschijnlijk is dit één van de redenen waarom het aanpassen van de specificaties van de cluster niet mogelijk is. Voor meer informatie over automatisch opschalen vindt u uitgebreide documentatie op volgende link: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-automatic-scaling.html. Met onderstaande stappen kan een cluster handmatig aangepast worden.

- Ga naar het overzicht van alle clusters door in te loggen op https://aws.amazon.com/emr
- Klik op de **cluster** die je wil uitbreiden/inperken
- Klik op het tabblad **Hardware**
- Klik in de kolom **Instance count** op **Resize**
- Vul het gewenste aantal instanties in in het verschenen tekstvak
- Klik op het vinkje om je wijzigingen op te slaan.

Omdat de mogelijkheden tot up- of downscaling beperkt zijn, krijgt Amazon EMR hiervoor een score van 2.

Moet (relatief) eenvoudig te gebruiken zijn (korte leercurce)

Amazon EMR is een heel uitgebreid platform met enorm veel mogelijkheden, daarom duurt het in het begin een tijdje vooraleer men het hele platform begrijpt en hiermee kan werken. Daarnaast is er ook enige configuratie nodig vooraleer men kan starten met het uitvoeren van queries (= jobs) op de aangemaakte cluster. Een voordeel is wel dat er veel documentatie beschikbaar is voor allerhande mogelijke situaties, dat maakt het leren van het platform een stuk eenvoudiger.

Maar toch krijgt Amazon EMR een score van slechts 3 voor dit requirement wegens de lange leercurve.

Moet onderhoudbaar zijn op lange termijn

Het onderhouden van een Amazon EMR cluster is op zich geen probleem, na de uitvoering van een Spark applicatie wordt alle tijdelijke data automatisch verwijderd. Daarentegen zijn er wel veel logbestanden die gecreëerd worden op de Amazon S3 bucket van de cluster. Het is ook moeilijk om hier een overzicht over te behouden en om te weten welke logbestanden weg mogen en welke niet. Daarnaast is het ook niet zo mooi als men bv. een cluster stopt, deze blijft nog enkele dagen tot weken staan in het overzicht van alle clusters. Bijna hetzelfde geldt voor de steps (Spark applicaties) die uitgevoerd zijn en eventueel gefaald zijn, alleen verdwijnen deze niet. Daarom blijft dat overzicht vol staan met gefaalde en succesvolle steps die er eventueel niet meer hoeven te staan. Het handige hieraan is wel dat het eenvoudig is om een succesvolle step nog eens te herhalen zonder de configuratie ervan opnieuw te hoeven doen, hiervoor bestaat de mogelijkheid om een step te klonen.

Hierdoor krijgt Amazon EMR een score van 3 op dit requirement.

5.2.2 Should have

Ondersteuning voor datastreams (real-time dataverwerking)

Aangezien Amazon EMR de mogelijkheid biedt om een Apache Spark cluster op te zetten is er bijgevolg ook meteen ondersteuning voor streams van data. Dit omdat Apache Spark de mogelijkheid van streaming built in heeft in een module.

Daarom krijgt Amazon EMR een score van 5 voor dit requirement.

5.2.3 Could have

Ondersteuning voor management en monitoring van het cloudplatform

Amazon EMR beschikt over uitgebreide logging m.b.v. Amazon CloudWatch, maar ook over een aantal samenvattingen hiervan op de detailpagina van elke cluster. Voor de logging op Amazon CloudWatch wordt een S3 bucket aangemaakt om de logbestanden op te slaan. CloudWatch bevat voor elk van de nodes in de cluster uitgebreide logbestanden waarnaar, indien gewenst, gekeken kan worden om bepaalde fouten op te lossen. Daarnaast is er ook de mogelijkheid om via SSH in te loggen op de master node van een cluster (zie Sectie 5.1). Daarbij kan er via onderstaand commando door een volledig logbestand van een applicatie (= step) gelopen worden m.b.v. alle mogelijkheden van het *less*-commando. Vervang <*application-id*> door het id van de applicatie waarvan men de logbestanden wil bekijken. Via deze logbestanden is het bijvoorbeeld mogelijk om in detail te zien wat er precies misgelopen is a.d.h.v. de stacktrace van een Exception bijvoorbeeld.

```
1 yarn logs --applicationId <application-id> | less
```

Daarom krijgt Amazon EMR een score van 5 voor dit requirement.

Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)

Binnen Amazon EMR is er ondersteuning voor extra plugins of modules en dit in de vorm van bootstrap actions die gespecifieerd kunnen worden wanneer de cluster aangemaakt wordt. Deze bootstrap actions worden steeds uitgevoerd wanneer de cluster correct opgestart is, het is de laatste stap vooraleer de cluster klaar en beschikbaar is voor verwerking van data. Deze bootstrap actions zijn bash scripts die de server zal uitvoeren. Er bestaat de mogelijkheid om zelf zo'n script te schrijven en op te geven, maar er zijn ook een heleboel voorbeelden voorhanden op de GitHub pagina van Amazon AWS: https://github.com/aws-samples/emr-bootstrap-actions. Het zelf implementeren van de bootstrap actions vereist dus wel een kennis van bash scripts waardoor de gebruiksvriendelijkheid van de mogelijkheid wel daalt.

Hierdoor krijgt Amazon EMR een score van 3 voor dit requirement.

Dit was het laatste requirement waarop Amazon EMR geëvalueerd zou worden. Daarom

vindt u in tabel 5.1 een overzicht van de scores per requirement alsook een uiteindelijke eindscore (gemiddelde van alle scores). Voor de duidelijkheid werden de rijen gekleurd per soort requirement: must have, should have en could have.

Requirement	**Score (op 5)**
Ondersteuning voor verschillende bestandsformaten zoals JSON, CSV, XML... als invoer	4
Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data	4
Ondersteuning voor polyglot persistence	5
Ondersteuning voor pipelines voor de data onboarding	5
Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR)	5
Moet (relatief) goedkoop zijn in implementatie en gebruik	1
Moet performant zijn (snel data inlezen en resultaat van een query geven)	1
Moet flexibel zijn voor up- en downscaling	2
Moet (relatief) eenvoudig te gebruiken zijn (korte leercurce)	3
Moet onderhoudbaar zijn op lange termijn	3
Ondersteuning voor datastreams (real-time dataverwerking)	5
Ondersteuning voor management en monitoring van het cloudplatform	5
Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)	3
Gemiddelde	3
Mediaan	4
Modus	5
Standaardafwijking	1,5
Variatiecoëfficiënt	0,43

Tabel 5.1: Overzicht score Amazon EMR

5.3 Queries

Voor dit onderzoek zijn een aantal queries bedacht (zie Hoofdstuk 3) die uitgevoerd zullen worden op Amazon EMR. De code van alle queries vindt u in bijlage F. Daarbij is in commentaar aangegeven waar elke query staat. In het script wordt de data ingeladen vanop een Amazon S3 bucket, hiervoor is er een acces key en secret key nodig om de toegang tot de S3 bucket mogelijk te maken. De keys gebruikt in dit onderzoek zijn uiteraard niet publiek beschikbaar gemaakt in dit document, daarom vervangt u *YOUR-ACCESS-KEY* en *YOUR-SECRET-KEY* respectievelijk door de access key en secret key van uw Amazon S3 bucket.
Alle resultaten en de verwerking van de uitvoeringstijden zijn te vinden in Hoofdstuk 7. Door een programmeerfout worden per iteratie de uitvoeringstijden van alle 10 queries in een apart bestand opgeslagen. Om deze bestanden samen te voegen werd onderstaand bash-commando uitgevoerd. Vervang <*pad-naar-uitvoeringstijden*> door het pad van de map waarin de uitvoeringstijden opgeslagen zijn. Het tweede commando plakt alle

bestanden samen met na elke lijn een enter, de samengevoegde uitvoeringstijden zijn te vinden in het bestand *out.csv* in dezelfde map als waarin de uitvoeringstijden apart opgeslagen zijn.

```
cd <pad-naar-uitvoeringstijden>
paste -d "\n" part-* > out.csv
```

6. Google Cloud Dataproc

> "Een snellere, eenvoudigere en voordelige manier om Spark en Hadoop uit te voeren"– Google

Als laatste wordt in deze vergelijkende studie dieper ingegaan op de Google Cloud Dataproc. In deze vergelijkende studie zullen tien queries losgelaten worden op een Apache Spark cluster op Google Cloud Dataproc en zullen alle mogelijkheden afgetoetst worden aan de requirements van IntoData, opgegeven in Sectie 3.1.

6.1 Google Cloud Dataproc instellen

Vooraleer men van start kan gaan met een analyse (of query) op Google Cloud Dataproc moet men eerst een cluster aanmaken. Voor Google Cloud Dataproc heeft men sowieso een Google-account nodig, net als bij BigQuery wordt het aanmaken van dit account niet behandeld in dit onderzoek. Naast een cluster is er ook een dataset nodig om de analyses op uit te voeren. Hiervoor wordt gebruik gemaakt van de reeds aangemaakte dataset in BigQuery. Om dit mogelijk te maken is de BigQuery connector nodig die de link legt tussen Cloud Dataproc en BigQuery.

6.1.1 Cluster aanmaken

De eerste stap in het onderzoek naar Google Cloud Dataproc is het aanmaken en instellen van een cluster. Hiervoor wordt gebruik gemaakt van het *gcloud* commando. Om dit commando te installeren vindt u meer informatie op volgende link: https://cloud.google.com/sdk/docs/.

In dit onderzoek zal gebruik gemaakt worden van een cluster van het type *n1-standard-4* met een ingebouwd jupyter notebook om het schrijven van de queries in Apache Spark te vereenvoudigen. Deze cluster beschikt over 4 vCPU's en 15GB werkgeheugen. Om deze cluster op te zetten voert u onderstaand commando uit op de command line. Vervang <*cluster-naam*> door de naam van de aangemaakte cluster.

```
gcloud dataproc clusters create <cluster-naam> \
   --initialization-actions \
      gs://dataproc-initialization-actions/jupyter/jupyter.sh
         \
   --num-workers 2 \
   --worker-machine-type=n1-standard-4 \
   --master-machine-type=n1-standard-4 \
   --region=europe-west2
```

6.1.2 BigQuery connector instellen

In dit geval is er geen configuratie nodig voor de BigQuery connector aangezien deze standaard geïnstalleerd wordt op een cluster op Google Cloud Dataproc. Daarnaast is ook de Cloud Storage connector nodig om de BigQuery connector te doen werken. Deze is ook standaard geïnstalleerd op een Cloud Dataproc cluster.

6.1.3 Verbinden met jupyter notebook

Om het jupyter notebook te kunnen gebruiken, voert u onderstaand commando uit om een SSH tunnel te maken naar de aangemaakte cluster. Indien er geen SSH keypair aanwezig is op uw lokaal toestel, zal het *gcloud*-command er één aanmaken en opslaan. De locatie van dit SSH keypair wordt afgeprint in de console. Vervang <*cluster-naam*> door de naam van de aangemaakte cluster, <*project-id*> door het id van het project en <*cluster-zone*> door de zone van de aangemaakte cluster.

```
gcloud compute ssh "<cluster-naam>-m" \
   --project <project-id> \
   --zone=<cluster-zone> \
   -- -D 10000 -N
```

Daarna voert u onderstaand commando uit om de jupyter notebook te openen op je lokaal toestel. Vervang <*browser-executable-path*> door de locatie op uw toestel van een browser naar keuze. Vervang wederom <*cluster-naam*> door de naam van de aangemaakte cluster.

```
<browser-executable-path> \
   "http://<cluster-naam>-m:8123" \
   --proxy-server="socks5://localhost:10000" \
   --host-resolver-rules="MAP * 0.0.0.0 , EXCLUDE localhost" \
   --user-data-dir=/tmp/
```

6.1.4 Taak verzenden en uitvoeren

Indien men een pyspark taak wil naar de server wil verzenden voor uitvoering, voert men onderstaand commando uit. In dit commando vervangt u <*location-task-file*>, <*cluster-name*> en <*cluster-region*> respectievelijk door de locatie van het uit te voeren Python-bestand, de naam van de cluster en de regio van de cluster.

```
gcloud dataproc jobs submit pyspark <location-task-file> --
    cluster <cluster-name> --region <cluster-region>
```

Bovenstaand commando zal het id van de job (= taak) afprinten en de console geblokkeerd houden met de uitvoer van de taak. Door de sneltoets Ctrl+C te gebruiken kan dit commando gestopt worden zonder dat de taak wordt gestopt op de cluster. Nadien kan eenvoudig terug de output in de console verkregen worden met onderstaand commando. Vervang <*job-id*>, <*project-id*> en <*cluster-region*> respectievelijk door het id van de job (= taak), het id van het project waarin de cluster draait en de regio van de cluster.

```
gcloud dataproc jobs wait <job-id> --project <project-id> --
    region <cluster-region>
```

6.2 Requirements

Voor dit onderzoek zijn een aantal requirements opgesteld waaraan o.a. Google Cloud Dataproc moet voldoen (zie Sectie 3.1). Hieronder wordt elk van deze requirements afgetoetst aan de mogelijkheden van Google Cloud Dataproc. Deze requirements werden reeds opgesplitst in *Nice to have*, *Should have* en *Could have*. Deze onderverdeling zal hieronder ook gebruikt worden om de tool te beoordelen.

6.2.1 Nice to have

Ondersteuning voor verschillende bestandsformaten zoals JSON, CSV, XML... als invoer

Voor dit eerste requirement kan dezelfde uitleg gegeven worden dan voor Amazon EMR. Daarom wordt u verwezen naar de uitleg bij Amazon EMR voor dit requirement.

Om dezelfde redenen dan gegeven bij Amazon EMR krijgt Google Cloud Dataproc een score van 4 voor dit requirement.

Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data

De uitleg bij dit requirement vindt u bij het volgende requirement aangezien beide samenhangen in Google Cloud Dataproc.

Ondersteuning voor polyglot persistence

Een groot voordeel aan Google Cloud Dataproc is dat er een aantal connectors voorzien worden door Google die het mogelijk maken om data vanop verschillende locaties te hergebruiken in Apache Spark. Het gaat hier om een connector voor BigQuery, Cloud Storage en Cloud Bigtable. Deze connectors laten het toe om data slechts op één locatie op te slaan en daarna te hergebruiken in de verschillende services van Google. Wel een kleine kanttekening hierbij is dat men door deze connectors meer kosten kan krijgen omdat er voor beide platformen betaald moet worden. Bijvoorbeeld: indien men de connector voor BigQuery gebruikt, kan men extra kosten krijgen door opslag, verwerking... van de data binnen BigQuery. De connectors bewijzen meteen dat Google Cloud Dataproc ondersteuning biedt voor polyglot persistence, nl. voor BigQuery, Cloud Storage, Cloud Bigtable en natuurlijk Hadoop.

Omdat Google Cloud Dataproc enkel connectors voorziet voor Google services, krijgt het een score van 3 op dit requirement.

Ondersteuning voor pipelines voor de data onboarding

Op dit gebied scoort Google Cloud Dataproc niet zo goed. Volgens Google (2018b) biedt Google Cloud Dataproc geen ondersteuning voor streams van data tijdens de verwerking van de data. Daarvoor bieden ze Google Cloud Dataflow aan, Google Cloud Dataproc dient enkel voor batchverwerking. Dus ook voor onboarding van de data a.d.h.v. een pipeline is Google Cloud Dataproc niet geschikt, hiervoor dient ook weer Google Cloud Dataflow. Een mooie vergelijking tussen de workloads voor beide systemen vindt u in tabel 6.1.

Daarom krijgt Google Cloud Dataproc een 1 voor dit requirement.

Workloads	**Cloud Dataproc**	**Cloud Dataflow**
Streamverwerking (ETL)		X
Batchverwerking (ETL)	X	X
Iteratieve verwerking en notebooks	X	
Machine learning met Spark ML	X	
Voorbereiden data voor machine learning		X

Tabel 6.1: Vergelijking workloads Google Cloud Dataproc en Google Cloud Dataflow (Google, 2018a)

Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR)

Binnen de Google Cloud Console bestaat er een IAM module waarin per gebruiker ingesteld kan worden welke rechten hij/zij heeft binnen deze cluster. Dit kan gaan van enkel leesrechten tot alle mogelijke rechten. Zo wordt het mogelijk om verschillende gebruikers andere dingen toe te laten of net niet toe te laten. De sterkte van deze module wordt pas duidelijk wanneer men voor de eerste keer een cluster wil aanmaken. Ik ben de

eigenaar van het project, maar in de web interface kon ik geen cluster aanmaken. Maar met het *gcloud* commando ging dit wel, dit commando vroeg of het de juiste rechten moest toekennen aangezien ik de eigenaar was van het project. Deze vraag wordt niet gesteld aan iemand die niet genoeg rechten heeft.

Daarom krijgt Google Cloud Dataproc een score van 5 voor dit requirement.

Moet (relatief) goedkoop zijn in implementatie en gebruik

Gebruikers van Google Cloud Dataproc worden per seconde gefactureerd, met een minimum verbruik van één minuut. In tabel 6.2 vindt u een overzicht van de kosten van een standaardcluster voor België. Alle prijzen zijn weergegeven per uur.

Type cluster	Virtuele CPUs	Geheugen	Dataproc Premium (USD)
n1-standard-1	1	3,75GB	$0,010
n1-standard-2	2	7,5GB	$0,020
n1-standard-4	4	15GB	$0,040
n1-standard-8	8	30GB	$0,80
n1-standard-16	16	60GB	$0,160
n1-standard-32	32	120GB	$0,320
n1-standard-64	64	240GB	$0,640

Tabel 6.2: Prijsoverzicht standaardcluster Google Cloud (Google, 2018c)

Wanneer met de Googe Cloud Platform Pricing Calculator[8] een kostenraming gemaakt wordt voor de huidige configuratie van de servers en opslag in dit onderzoek (zoals hieronder weergegeven), komt dit op een bedrag van $92,01 per maand (ongeveer €77). Dit kan opgesplitst worden in $86,40 voor Google Cloud Dataproc en $5,61 voor Google Cloud Storage (het opslaan van de csv-bestanden met de data). Dit is eigenlijk niet zo goedkoop dan de prijzen op het eerste zicht lijken. Google biedt wel een handig overzicht aan van de kosten van de huidige periode of eventuele andere periodes. Dit overzicht is veel duidelijker dan bv. bij Amazon EMR waarbij alles wat door elkaar staat.

Configuratie calculator

- Cluster Google Cloud Dataproc
  - Master node type: n1-standaard-4
  - Worker node type: n1-standard-4
  - Aantal worker nodes: 2
  - Aantal uur per maand: 720 (24 uur * 30 dagen)
- Cloud Storage
  - Regio: europe-west2
  - Klasse: regionaal
  - Regionale opslag: 9GB
  - Aantal klasse A requests: 1 miljoen (willekeurig gekozen)
  - Aantal klasse B requests: 1 miljoen (willekeurig gekozen)

Door bovenstaande redenen en voorbeeld kostenberekening, krijgt Google Cloud Dataproc een score van 3 voor dit requirement.

Moet performant zijn (snel data inlezen en resultaat van een query geven)

Voor dit requirement kan dezelfde argumentatie gebruikt worden als voor dit requirement in Amazon EMR: Apache Spark is vele malen trager dan Google BigQuery. Dit blijkt ook uit de resultaten te zien in Hoofdstuk 7. Het duurt meer dan 5 minuten alvorens de data ingeladen is in een RDD waarna het vaak tussen de 2 à 5 minuten duurt alvorens een query volledig is uitgevoerd. Dit blijkt ook uit de resultaten verderop in dit Hoofdstuk (zie Sectie 6.3).

Hierdoor krijgt Google Cloud Dataproc een score van 1.

Moet flexibel zijn voor up- en downscaling

Google Cloud Dataproc laat net als Amazon EMR enkel toe om het aantal nodes in de cluster omhoog of omlaag te doen. Ook hier ontbreekt de mogelijkheid om de specificaties van de cluster aan te passen, waarschijnlijk ook om dezelfde reden als bij Amazon EMR. Meer informatie over het schalen van een cluster vindt u op volgende link: https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/scaling-clusters. Voor het automatisch schalen van een cluster in Google Cloud Dataproc bestaat er geen mogelijkheid ingebouwd in Cloud Dataproc zelf, maar volgens Laufer (2018) zou er een plugin genaamd *Shamash* bestaan die deze mogelijkheid wel aanbiedt. Maar hierop wordt niet dieper ingegaan in dit onderzoek.

Door de hierboven genoemde redenen krijgt Google Cloud Dataproc een score van 2 op dit requirement.

Moet (relatief) eenvoudig te gebruiken zijn (korte leercurce)

Het aanleren van Google Cloud Dataproc ging sneller dan Amazon EMR. Dit komt voor een deel omdat de web interface van Google Cloud Dataproc veel eenvoudiger is dan die van Amazon EMR, maar daardoor ook minder flexibiliteit te bieden heeft. Wil men iets ingewikkeldere configuraties of acties doen, gaan die soms sneller en makkelijker a.d.h.v. het *gcloud* commando. Daarnaast was het heel eenvoudig om een weg te vinden doorheen de interface en een cluster was snel opgezet, hetzelfde geldt voor een taak starten op de cluster. Voor de rest is er weinig kennis vereist om overweg te kunnen met de Google Cloud Console en Google Cloud Dataproc, juist een kennis van Apache Spark (in Python of Java bijvoorbeeld) is vereist indien men verder wil gaan met uitgebreide analyses.

Omdat de web interface soms beperkt is, krijgt Google Cloud Dataproc een score van 4 voor dit requirement.

Moet onderhoudbaar zijn op lange termijn

De Google Cloud Console bieden een handig overzicht aan voor elk van de verschillende platformen die aangeboden worden, zo ook voor Google Cloud Dataproc. Er is een duidelijk overzicht van de taken met eventuele fouten onderweg. Er is een duidelijk overzicht van alle clusters die momenteel draaien met hun huidige configuraties erbij. Alsook bestaat er de mogelijkheid om overal voor elke actie het bijbehorende command line commando op te vragen, hiervoor gebruikt men wederom het *gcloud* commando. Daarnaast is bij elke cluster duidelijk weergegeven waar de data in Google Cloud Storage opgeslagen wordt. Zo is het eenvoudig om door te klikken naar de desbetreffende bucket en eventuele data op te halen of te verwijderen. Het eenvoudige aan Google Cloud Dataproc is dat er weinig tot geen onderhoud vereist is. Enkel de Google Cloud Storage moet een beetje op orde gehouden worden zodat er geen onnodige (tijdelijke) bestanden van een Spark applicatie blijven staan, maar voor de rest doet Google Cloud Dataproc alles.

Wel zou het handig zijn dat er iets meer mogelijkheden zouden zijn op het gebied van configuratie van de cluster, daarom krijgt Google Cloud Dataproc een score van 3 voor dit requirement.

6.2.2 Should have

Ondersteuning voor datastreams (real-time dataverwerking)

Zoals reeds eerder gezegd biedt Google Cloud Dataproc geen ondersteuning voor het verwerken van datastreams, hiervoor bieden ze Google Cloud Dataflow aan.

Daarom krijgt Google Cloud Dataproc een 1 voor dit requirement.

6.2.3 Could have

Ondersteuning voor management en monitoring van het cloudplatform

Google Cloud Dataproc beschikt over uitgebreide mogelijkheden tot het monitoren van de opgezette clusters. Daarbij is het mogelijk om verschillende soorten statistieken te configureren zodat deze op een overzichtelijke grafiek verschijnen op de console van Google Cloud. Hierbij is het onder andere mogelijk om het CPU-gebruik, het aantal bytes verwerkt... weer te geven op een grafiek. Daarbij beschikt Google Cloud Dataproc over een uitgebreid logbestand waarnaar enorm veel gegevens geschreven worden en waaruit men enkel het nuttig kan filteren a.d.h.v. enkele filters.
Daarnaast bestaat er ook de mogelijkheid om foutrapportage aan te zetten a.d.h.v. Stackdriver. Op Stackdriver krijgt men gratis toegang voor 30 dagen zodat men het platform kan ontdekken. Deze functionaliteit is niet onderzocht binnen het kader van dit onderzoek.

Kortom, er zijn heel veel mogelijkheden tot monitoring en management van een cluster op Google Cloud Dataproc en in de volledige Google Cloud, daarom krijgt Google Cloud Dataproc een score van 5 voor dit requirement.

Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)

Aangezien Google Cloud Dataproc een Hadoop en Apache Spark cluster opzet, kan elke bibliotheek of tool van dit hele ecosysteem gebruikt worden binnen een cluster op Google Cloud Dataproc. Het enige wat dit vereist is dat de gewenste bibliotheek geïnstalleerd wordt op de cluster. Dit kan eenvoudig gedaan worden a.d.h.v. een SSH verbinding zoals hierboven reeds is uitgelegd. Op de configuratie van deze bibliotheken wordt niet verder ingegaan aangezien dit buiten de scope van dit onderzoek ligt. Daarnaast zijn er ook een heleboel partners die mogelijkheden bieden tot integratie met Google Cloud Dataproc, een overzicht hiervan is te vinden op https://cloud.google.com/dataproc/docs/resources/partners.

Daarom krijgt Google Cloud Dataproc een score van 5 voor dit requirement.

Dit was het laatste requirement waarop Google Cloud Dataproc geëvalueerd zou worden. Daarom vindt u in tabel 6.3 een overzicht van de scores per requirement alsook een uiteindelijke eindscore (gemiddelde van alle scores). Voor de duidelijkheid werden de rijen gekleurd per soort requirement: must have, should have en could have.

Requirement	**Score (op 5)**
Ondersteuning voor verschillende bestandsformaten zoals JSON, CSV, XML... als invoer	4
Ondersteuning voor uitbreiding van data zonder problemen met reeds bestaande data	3
Ondersteuning voor polyglot persistence	3
Ondersteuning voor pipelines voor de data onboarding	1
Ondersteuning voor authenticatie en authorisatie om zo data af te schermen van partijen die de data niet mogen zien (ten gevolgde van GDPR)	5
Moet (relatief) goedkoop zijn in implementatie en gebruik	3
Moet performant zijn (snel data inlezen en resultaat van een query geven)	1
Moet flexibel zijn voor up- en downscaling	2
Moet (relatief) eenvoudig te gebruiken zijn (korte leercurce)	4
Moet onderhoudbaar zijn op lange termijn	3
Ondersteuning voor datastreams (real-time dataverwerking)	1
Ondersteuning voor management en monitoring van het cloudplatform	5
Ondersteuning voor extra's in modules (modulariteit van invoegtoepassingen/integraties)	5
Gemiddelde	3
Mediaan	3
Modus	3
Standaardafwijking	1,5
Variatiecoëfficiënt	0,49

Tabel 6.3: Overzicht score Google Cloud Dataproc

6.3 Queries

De code van alle queries vindt u in bijlage E. Daarbij is in commentaar aangegeven waar elke query staat.
Alle resultaten en de verwerking van de uitvoeringstijden is te vinden in Hoofdstuk 7.

7. Verwerking resultaten

In dit hoofdstuk zullen de resultaten van de uitvoeringstijden van elke oplossing verwerkt worden om zo tot een statistisch correct antwoord te komen op de hoofdonderzoeksvraag uit Sectie 1.2.

7.1 Resultaten requirements

Hieronder vindt u per oplossing een overzicht van het gemiddelde, de mediaan, de modus, de standaardafwijking en de variatiecoëfficiënt van alle scores gegeven op de requirements.

	Google BigQuery	**Google Cloud Dataproc**	**Amazon EMR**
Gemiddelde	4	3	3
Mediaan	4	3	4
Modus	4	3	5
Standaardafwijking	1,12	1,5	1,5
Variatiecoëfficiënt	0,27	0,49	0,43

Tabel 7.1: Statistieken requirements

Om deze resultaten te vergelijken gaan we gebruik maken van een niet-gepaarde t-test. Stel μ_1 gelijk aan het populatiegemiddelde van de scores van Google BigQuery en μ_2 gelijk aan het populatiegemiddelde van de scores van Amazon EMR. Hiermee willen we aantonen dat Google BigQuery beter scoort dan Amazon EMR. We nemen een $\alpha = 5\%$. De hypotheses kunnen als volgt voorgesteld worden:

$$H_0 : \mu_1 - \mu_2 = 0 \; en \; H_1 : \mu_1 - \mu_2 > 0$$

Uit deze t-test komt een p-waarde van 0,1557 wat hoger ligt dan het significantieniveau van 0,05, dus moeten we onze nulhypothese aanvaarden en stellen dat Google BigQuery niet beter scoort dan Amazon EMR.

Kijken we nu naar Google Dataproc en Amazon EMR, dan kunnen we opnieuw een niet-gepaarde t-test uitvoeren. Stel μ_1 gelijk aan het populatiegemiddelde van de scores van Google Cloud Dataproc en μ_2 gelijk aan het populatiegemiddelde van de scores van Amazon EMR. Hiermee willen we aantonen dat Google Cloud Dataproc minder goed scoort dan Amazon EMR. We nemen een $\alpha = 5\%$. De hypotheses kunnen als volgt voorgesteld worden:

$$H_0 : \mu_1 - \mu_2 = 0 \; en \; H_1 : \mu_1 - \mu_2 < 0$$

Uit deze t-test komt een p-waarde van 0,2205 wat hoger ligt dan het significatieniveau van 0,05, dus moeten we onze nulhypothese aanvaarden en stellen dat Google Cloud Dataproc niet slechter scoort dan Amazon EMR.

Kijken we als laatste naar Google BigQuery en Google Cloud Dataproc met μ_1 gelijk aan het populatiegemiddelde van de scores van Google Cloud Dataproc en μ_2 gelijk aan het populatiegemiddelde van de scores van Google BigQuery. Hiermee willen we aantonen dat Google BigQuery beter scoort dan Google Cloud Dataproc. We nemen een $\alpha = 5\%$. De hypotheses kunnen als volgt voorgesteld worden:

$$H_0 : \mu_1 - \mu_2 = 0 \; en \; H_1 : \mu_1 - \mu_2 < 0$$

Uit deze t-test komt een p-waarde van 0,0332 wat lager ligt dan het significantieniveau van 0,05, dus we kunnen de nulhypothese verwerpen en stellen dat Google BigQuery beter scoort dan Google Cloud Dataproc.

Bovenstaande resultaten werden behaald met onderstaande code in R.

```
scoresBQ <- c(4,4,1,4,5,5,5,3,5,4,4,4,5)

# Gemiddelde, mediaan, standaardafwijking en
   variantiecoefficient
mean(scoresBQ)
median(scoresBQ)
sd(scoresBQ)
sd(scoresBQ)/mean(scoresBQ)

scoresDataproc <- c(4,3,3,1,5,3,1,2,4,3,1,5,5)

```

```
# Gemiddelde, mediaan, standaardafwijking en
    variantiecoefficient
mean(scoresDataproc)
median(scoresDataproc)
sd(scoresDataproc)
sd(scoresDataproc)/mean(scoresDataproc)

scoresEMR <- c(4,4,5,5,5,1,1,2,3,3,5,5,3)

# Gemiddelde, mediaan, standaardafwijking en
    variantiecoefficient
mean(scoresEMR)
median(scoresEMR)
sd(scoresEMR)
sd(scoresEMR)/mean(scoresEMR)

# Scoort Google BigQuery beter dan Amazon EMR?
t.test(scoresBQ, scoresEMR, paired = FALSE, alternative = "
    greater", mu=0)

# Scoort Google Cloud Dataproc slechter dan Amazon EMR?
t.test(scoresDataproc, scoresEMR, paired = FALSE, alternative
    = "less", mu=0)

# Scoort Google BigQuery beter dan Google Cloud Dataproc?
t.test(scoresBQ, scoresDataproc, paired = FALSE, alternative =
     "greater", mu=0)
```

7.2 Resultaten queries

Hieronder vindt u per oplossing een overzicht van het gemiddelde, de mediaan, de standaardafwijking en de variatiecoëfficiënt per query. Voor eenvoud tijdens het lezen worden hieronder alle uitgevoerde queries opgesomd zoals gedefinieerd in Hoofdstuk 3.

1. In welke staat worden het meeste tickets uitgeschreven?
2. Welke inbreuk wordt het meeste vastgesteld?
3. Welk politiedistrict schrijft het meeste tickets uit?
4. Welke violation code komt overeen met welke violation description?
5. Top 10 van de staten waar het meest een ticket wordt uigeschreven
6. Top 10 van het aantal inbreuken per staat per automerk
7. Op welk uur van de dag worden het meeste tickets uitgeschreven?
8. Een overzicht van het aantal tickets per maand, opgesplitst in New York en andere staten
9. Maken mensen met een niet-ingeschreven voertuig meer inbreuken? Of net minder?
10. Op welke momenten van de dag gebeuren het meeste inbreuken?

De volledige lijst van uitvoeringstijden per query is te vinden op de GitHub pagina[1] van dit onderzoek in de map uitvoeringstijden. Een kleine nuance bij de uitvoeringstijden van Google BigQuery is dat de uitvoeringstijden bepaald worden door het aantal queries dat door andere gebruikers tegelijk naar Google BigQuery gestuurd worden. Daarnaast is het belangrijk om te weten dat de resultaten uit de queries irrelevant zijn voor dit onderzoek, deze worden enkel weergegeven in bijlage C voor de volledigheid maar hebben geen nut in het kader van dit onderzoek. Een laatste opmerking bij de uitvoeringstijden is dat de queries op Amazon EMR geen 100 keer uitgevoerd zijn door een netwerkfout naar de Amazon S3 Bucket waarop de uitvoeringstijden opgeslagen werden. Daarom zijn de queries slechts 86 keer uitgevoerd op Amazon EMR. Om die reden worden de uitvoeringstijden van Google BigQuery en Google Cloud Dataproc ook ingekort tot 86 keer om de vergelijking correct te kunnen maken. Door een programmeerfout worden de resultaten van query 4 niet getoond bij Google Cloud Dataproc en Amazon EMR, deze waren namelijk identiek aan die van query 5. Het gaat hier om dezelfde index te gebruiken voor query 4 en query 5 in de array *queryTimes*, deze programmeerfout is opgelost in de code die te vinden is in bijlage E en F.

Resultaten Google BigQuery

	Query 1	**Query 2**	**Query 3**	**Query 4**	**Query 5**
Gemiddelde (ms)	2 669,06	2 794,31	2 746,89	2 957,51	2 777,74
Mediaan (ms)	2 536	2 694	2 617,5	2 740	2 589,5
Standaardafwijking	312,94	395,56	376,84	1065,63	447,28
Variatiecoëfficiënt	0,12	0,14	0,14	0,36	0,16
	Query 6	**Query 7**	**Query 8**	**Query 9**	**Query 10**
Gemiddelde (ms)	4 064,59	3 390,31	3 892,54	2 785,55	5 308,80
Mediaan (ms)	3 952	3 343,5	3 636,5	2 727	5 152,5
Standaardafwijking	295,94	278,52	1660,13	336,71	719,72
Variatiecoëfficiënt	0,07	0,08	0,43	0,12	0,14

Tabel 7.2: Statistieken Google BigQuery

[1] https://github.com/thomasaelbrecht/serverless-big-data

Resultaten Google Cloud Dataproc

	Query 1	**Query 2**	**Query 3**	**Query 4**	**Query 5**
Gemiddelde (ms)	287 019,8	295 919,2	287 076,2	/	287 105,4
Mediaan (ms)	283 479	293 867,5	284 564,5	/	283 484
Standaardafwijking	14 399,11	14 656,61	14 430,99	/	13 760,36
Variatiecoëfficiënt	0,05	0,05	0,05	/	0,05
	Query 6	**Query 7**	**Query 8**	**Query 9**	**Query 10**
Gemiddelde (ms)	298 695,4	292 436	305 890,6	284 774,2	302 895,5
Mediaan (ms)	295 391	289 460	302 416,5	281 979,5	299 388,5
Standaardafwijking	15 028,11	14 479,21	15 366,05	12 031,20	16 308,92
Variatiecoëfficiënt	0,05	0,05	0,05	0,04	0,05

Tabel 7.3: Statistieken Google Cloud Dataproc

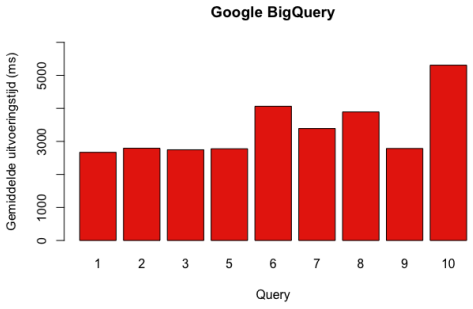
Resultaten Amazon EMR

	Query 1	**Query 2**	**Query 3**	**Query 4**	**Query 5**
Gemiddelde (ms)	512 948,2	511 062,9	511 306,5	/	509 737,6
Mediaan (ms)	509 787,5	512 060	509 505	/	511 210
Standaardafwijking	21 339,71	9 088,71	18 632,07	/	9 062,11
Variatiecoëfficiënt	0,04	0,02	0,04	/	0,02
	Query 6	**Query 7**	**Query 8**	**Query 9**	**Query 10**
Gemiddelde (ms)	512 187,3	514 225,6	519 928,8	507 786,3	518 061,6
Mediaan (ms)	511 562	512 617,5	519 579,5	508 771	518 025
Standaardafwijking	12 979,75	18 047,15	11 922,54	7 900,73	17 778,18
Variatiecoëfficiënt	0,03	0,04	0,02	0,02	0,03

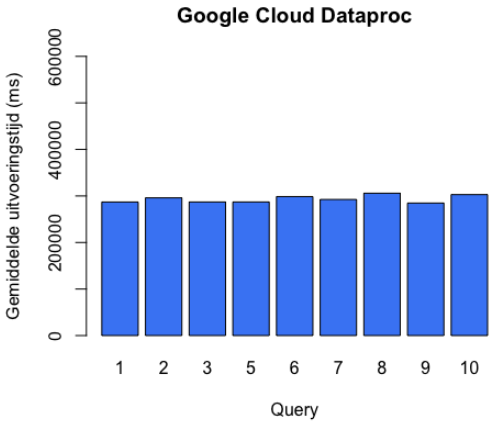
Tabel 7.4: Statistieken Amazon EMR

Wat opvalt uit bovenstaande waarden is dat volgens de variatiecoëfficiënten de uitvoeringstijden van Google Cloud Dataproc en Amazon EMR veel minder gespreid zijn dan de uitvoeringstijden van Google BigQuery. Dit wil zeggen dat men bij Google Cloud Dataproc en bij Amazon EMR beter een beeld heeft van welke uitvoeringstijd men kan verwachten. Bij Google BigQuery kan deze bredere spreiding te verklaren zijn door het feit dat de uitvoeringstijd afhankelijk is van het aantal queries dat tegelijk uitgevoerd wordt op Google BigQuery door verschillende gebruikers.

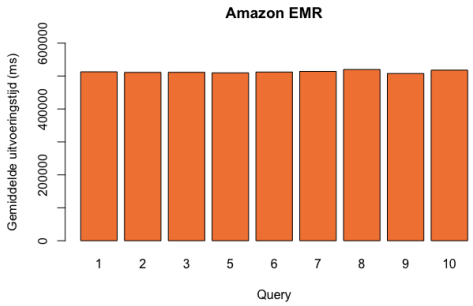
Alle uitvoeringstijden worden duidelijk weergegeven op Figuur 7.1a, 7.1b en 7.1c. Merk op: de as van de grafiek op Figuur 7.1a verschilt van de andere twee grafieken aangezien de grafiek anders niet te zien was. Op de x-as wordt telkens de query weergegeven waarbij op de y-as de uitvoeringstijd in milliseconden weergegeven wordt.
Deze drie grafieken zijn samengevoegd in Figuur 7.2. Op de x-as wordt opnieuw de query weergegeven waarbij op de y-as de uitvoeringstijd in seconden weergegeven wordt.



(a) Google BigQuery

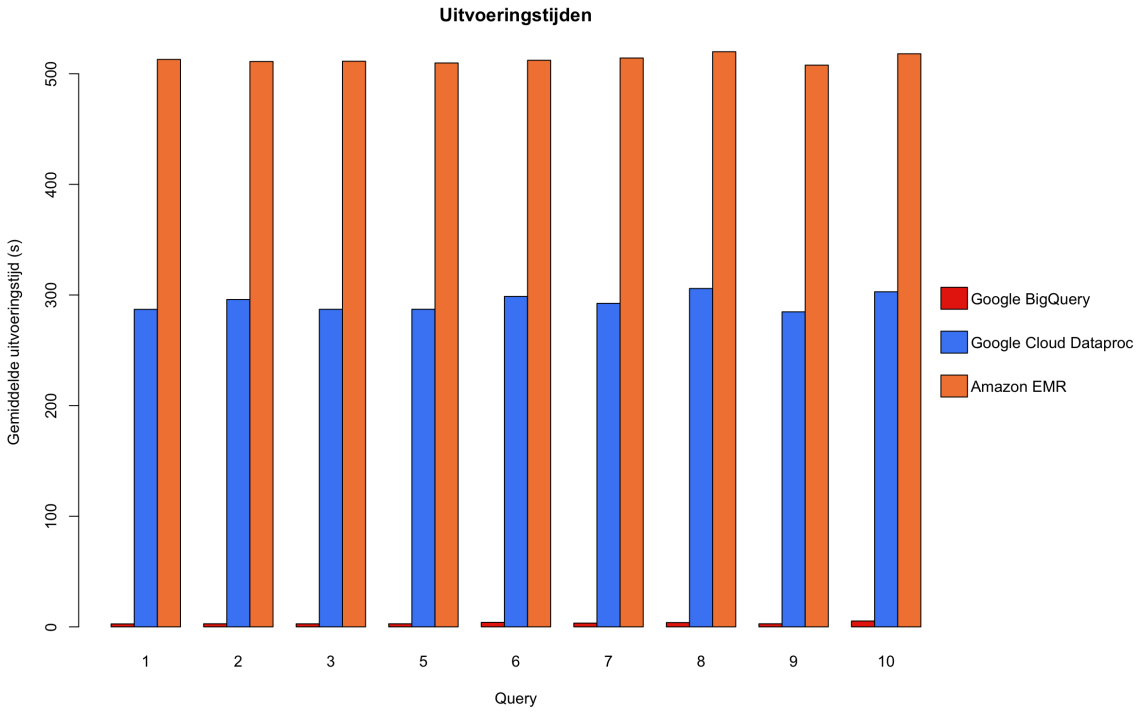


(b) Google Cloud Dataproc



(c) Uitvoeringstijden EMR

Figuur 7.1: Uitvoeringstijden per oplossing



Figuur 7.2: Overzicht uitvoeringstijden

In wat volgt worden de hypotheses, gegeven in Sectie 1.4, getoetst. Hierbij wordt het gemiddelde van query 4 voor Google BigQuery weggelaten aangezien er voor Amazon EMR geen gemiddelde is om de hierboven genoemde reden.
Voor het toetsen van de hypotheses wordt gebruik gemaakt van een niet-gepaarde t-test. Deze test is aangewezen aangezien we twee steekproeven met elkaar dienen te vergelijken.

Stel μ_1 gelijk aan het gemiddelde van alle gemiddeldes van de uitvoeringstijden van de queries van Google BigQuery en μ_2 idem voor Amazon EMR. Hiermee willen we aantonen dat Google BigQuery sneller is in het uitvoeren van queries dan Amazon EMR. Dan kunnen we de hypotheses noteren als:

$$H_0 : \mu_1 - \mu_2 = 0 \; en \; H_1 : \mu_1 - \mu_2 < 0$$

Indien we de gemiddeldes van Google BigQuery en Amazon EMR invoeren in R en de functie *t.test* uitvoeren, krijgen we een p-waarde die kleiner is dan $2,2 * 10^{-16}$. Deze p-waarde ligt onder het significantieniveau van 0,05, dus we kunnen de nulhypothese verwerpen en we mogen de alternatieve hypothese aanvaarden. Dit heeft tot gevolg dat de gestelde hypothese waar is en we dus kunnen stellen dat Google BigQuery sneller is dan Amazon EMR.

Stel nu μ_1 gelijk aan het gemiddelde van alle gemiddeldes van de uitvoeringstijden van de queries van Google Cloud Dataproc en μ_2 idem voor Amazon EMR. Dan kunnen we de hypotheses opnieuw noteren als:

$$H_0 : \mu_1 - \mu_2 = 0 \; en \; H_1 : \mu_1 - \mu_2 < 0$$

Uit deze t-test volgt opnieuw een p-waarde die kleiner is dan $2,2 * 10^{-16}$. Dus kunnen we wederom de nulhypothese verwerpen en de alternatieve hypothese aanvaarden. Dit ontkracht meteen de tweede hypothese gesteld in dit onderzoek. Er is dus weldegelijk een verschil in uitvoeringstijd tussen queries op Google Cloud Dataproc en Amazon EMR. Dit is een opmerkelijk resultaat en biedt de mogelijkheid tot verder onderzoek om hiervoor een gepaste verklaring te vinden.

Bovenstaande resultaten werden behaald met onderstaande code in R.

Uitvoeringstijden verwerken

```
bigquery <- read.csv(file = '/Users/thomasaelbrecht/Desktop/R/
   bigquery.csv', header = TRUE, sep = ',')
apply(bigquery, 2, mean)
apply(bigquery, 2, median)
apply(bigquery, 2, sd)

dataproc <- read.csv(file = '/Users/thomasaelbrecht/Desktop/R/
   dataproc.csv', header = TRUE, sep = ';')
apply(dataproc, 2, mean)
apply(dataproc, 2, median)
apply(dataproc, 2, sd)

emr <- read.csv(file = '/Users/thomasaelbrecht/Desktop/R/emr.
   csv', header = TRUE, sep = ';')
apply(emr, 2, mean)
apply(emr, 2, median)
apply(emr, 2, sd)
```

Hypotheses toetsen

```
gemBQ <- c(2669.058,2794.314,2746.884,2777.744,
           4064.593,3390.314,3892.535,2785.547,
           5308.802)
gemEMR <- c(512948.2,511062.9,511306.5,509737.6,
            512187.3,514225.6,519928.8,507786.3,
            518061.6)
gemDataproc <- c(287019.8,295919.2,287076.2,287105.4,
                 298695.4,292436.0,305890.6,284774.2,
                 302895.5)

# Eerste hypothese
t.test(gemBQ, gemEMR, alternative = "less", mu=0)

# Tweede hypothese
t.test(gemDataproc, gemEMR, alternative = "less", mu=0)
```

8. Conclusie

In dit onderzoek wordt een antwoord gegeven op de onderzoeksvraag ‘Wanneer moet een bedrijf Amazon EMR verkiezen boven Google BigQuery?’Hiervoor is een vergelijkende studie uitgevoerd naar de verschillende functionaliteiten en technische eigenschappen van Google BigQuery, Google Cloud Dataproc en Amazon EMR.

Uit de resultaten van de requirementsanalyse bleek dat Google BigQuery niet significant beter scoort dan Amazon EMR, het scoort wel significant beter dan Google Cloud Dataproc. Toch is er wel degelijk een verschil tussen Google BigQuery en Amazon EMR dat duidelijk werd tijdens het praktisch deel van dit onderzoek. Het is eenvoudiger en sneller om een analyse te starten op Google BigQuery dan op Amazon EMR. Daarbij hebben beiden wel specifieke use cases: Google BigQuery dient vooral voor interactieve en ad-hoc queries terwijl Amazon EMR beter geschikt is voor langlopende batchopdrachten zoals bijvoorbeeld machine learning of data mining. Hetzelfde geldt voor Google Cloud Dataproc.

Uit de resultaten van de uitvoeringstijden blijkt wel dat Google BigQuery significant sneller is dan Amazon EMR. Het gaat over een verschil in uitvoeringstijden van enkele seconden voor Google BigQuery met enkele minuten voor Amazon EMR, hetzelfde geldt voor Google Cloud Dataproc. Snelheid is dus één van de grote voordelen van Google BigQuery tegenover Amazon EMR en Google Cloud Dataproc. Een nadeel aan Google BigQuery is wel dat alle mogelijke libraries beschikbaar voor Apache Spark niet te gebruiken zijn, de enige mogelijkheden voor Google BigQuery zijn SQL en JavaScript. Daarnaast bestaat er ook wel de mogelijkheid om Google BigQuery aan te spreken vanuit Java-code zonder naar de Web UI te hoeven gaan. Het grote voordeel dan voor Google Cloud Dataproc en Amazon EMR is de mogelijkheid tot het gebruiken van alle libraries die ter beschikking zijn voor Apache Spark.

Een tweede groot voordeel voor Google BigQuery is het ontbreken van de zorgen over de server die nodig is voor de verwerking. Google BigQuery zal er automatisch voor zorgen dat de query snel uitgevoerd kan worden met de nodige resources om dit te verwezenlijken. Bij Amazon EMR en Google Cloud Dataproc moet de gebruiker zelf bepalen welke resources er nodig zijn voor de taken die uitgevoerd moeten worden. Daarbij komt nog een nadeel voor zowel Amazon EMR als Google Cloud Dataproc dat er geen mogelijkheid is tot het dynamisch uitbreiden van de resources van de server. Enkel bij Amazon EMR bestaat een kleine mogelijkheid tot het aanpassen van het aantal instanties van de slaafnodes, deze mogelijkheid bestaat niet bij Google Cloud Dataproc.

Een ander nadeel voor Google Cloud Dataproc en Amazon EMR is dat de kosten voor het draaien van de cluster en het uitvoeren van de taken snel oplopen tot boven de €100. Dit nadeel viel al meteen op bij het starten van het praktisch deel van dit onderzoek bij Google Cloud Dataproc en Amazon EMR.

Uit deze vergelijkende studie blijkt dat elk van deze oplossingen zijn eigen specifieke use cases heeft. Google BigQuery is vooral voor interactieve en ad-hoc queries (voor analyses die snel resultaat vereisen), Google Cloud Dataproc en Amazon EMR zijn vooral voor lang lopende batchverwerkingen. Wel is duidelijk dat Google BigQuery de absolute winnaar is op het gebied van uitvoeringssnelheid van de queries maar dit komt omdat Amazon EMR en Google Cloud Dataproc draaien op Apache Spark wat op zijn beurt gebaseerd is op MapReduce wat op zijn beurt een tragere verwerking van de data oplevert.

Bij de start van dit onderzoek had ik deze uitslag wel al verwacht aangezien elk artikel over Google BigQuery en Google Cloud Dataproc stelde dat Google BigQuery een absolute snelheidsduivel was. Ik had ook verwacht dat deze vaststelling doorgetrokken kon worden naar Amazon EMR.

Dit onderzoek biedt zeker een meerwaarde voor bedrijven die zich in dit vakgebied willen specialiseren. Het geeft een duidelijk beeld van wat de specifieke gevallen zijn om te kiezen voor Google BigQuery, Google Cloud Dataproc of Amazon EMR. Natuurlijk biedt dit onderzoek nog veel ruimte voor verder onderzoek. Mogelijk vervolgonderzoek kan gaan over de verschillende mogelijkheden tot integratie met streamingservices voor de streaming van data of integratie met pipelines tijdens de onboarding van de data. Een ander mogelijk vervolgonderzoek kan zijn of er meer uit Amazon EMR of Google Cloud Dataproc gehaald kan worden indien er gebruik gemaakt wordt van bijvoorbeeld Spark SQL i.p.v. operaties op RDD's. Ander mogelijk vervolgonderzoek kan ook zijn wat mogelijke tegenhangers zijn van Google BigQuery en hoe het scoort naast deze tools. Google BigQuery was namelijk een grote kanshebber om het te winnen tegen Apache Spark, maar hoe scoort het naast andere implementaties van Dremel of gelijkwaardige alternatieven voor Google BigQuery? Een laatste mogelijk onderwerp voor verder onderzoek werd gegeven in Hoofdstuk 7. Er werd een significant verschil waargenomen in de uitvoeringstijden van de queries op Google Cloud Dataproc en Amazon EMR. Mogelijk verder onderzoek zou een oorzaak voor deze vaststelling kunnen zoeken.

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

In veel bedrijven staat data centraal. Belangrijk daar bij is:

- Data structuring (relationeel, dimensioneel modelleren)
- Data integratie (ETL-ELT, data movement, migration, replication) voor operationele systemen of datawarehouses
- Data quality en master data management (MDM)
- Data delivery (visualisatie, rapportering, BI / BA)
- Data governance

De dataservicesmarkt evolueert razendsnel. Een bedrijf wil vaak de klant steeds een stapje voor zijn, maar daarvoor moet men niet alleen een duidelijk beeld hebben van de noden en trends die binnen deze markt doorbreken. Men moet eveneens, in een vroeg stadium, inzicht hebben in de nieuwste technologieën en tools. Dat is net wat dit onderzoek te bieden heeft. Dit onderzoekt focust op het vergelijken van een Amazon EMR Spark Cluster en Google BigQuery.

De onderzoeksvragen zijn:

- Wat heeft een Apache Spark cluster op Amazon EMR te bieden?
- Wat heeft Google BigQuery te bieden?

- Welke architectuur zit er onder Google BigQuery?
- Kan Google Cloud Dataproc een alternief zijn voor een Amazon EMR Apache Spark cluster?
- Kan een HBase cluster een alternatief zijn voor een Amazon EMR Apache Spark cluster?
- Hebben deze technologieën weldegelijk organisaties iets te bieden?
- Wat zijn de voor- en nadelen van beide?
- Welke functionaliteiten en technische eigenschappen hebben ze?

A.2 Literatuurstudie

A.2.1 Wat is Apache Spark?

Apache Spark is een open-source, gedistribueerd verwerkingssysteem, vaak gebruikt voor big data workloads. Apache Spark gebruikt in-memory caching en een geoptimaliseerde uitvoering voor snelle performantie. Het ondersteunt algemene batchverwerking, streaminganalyses, machine learning, graph-databases en ad hoc-queries. (Amazon, 2017b) Vroeger maakte men voor Big Data gebruik van een MapReduce cluster, maar hieraan zijn enkele limitaties verbonden:

- Batchverwerking (niet interactief)
- Ontwikkeld voor een bepaald probleem
- MapReduce programming paradigma is niet volledig begrepen
- Gebrek aan opgeleide, ondersteunende professionals (Langit, 2013)

Om een oplossing te bieden voor deze limitaties is Apache Spark ontwikkeld in 2012. Apache Spark laat zowel iteratieve algoritmes toe als herhaaldelijk bevragen van de data op databank-stijl. Apache Spark ondersteunt ook verschillende talen zoals Java, Scale en Python. Daarnaast is er volledige ondersteuning voor pure SQL (of HiveQL) queries.

A.2.2 Amazon EMR

Met Amazon EMR kan je snel en makkelijk de capaciteit, die je datawarehouse nodig heeft, voorzien. Het is ook eenvoudig om deze capaciteit uit te breiden of in te perken indien nodig. De twee mogelijke opties om capaciteit toe te voegen of te verwijderen zijn ofwel meerdere clusters deployen of een draaiende cluster veranderen van grootte. Volgens Amazon (2017a) is Amazon EMR ook low cost, maar dit wordt uiteraard onderzocht in dit onderzoek. Daarnaast biedt Amazon EMR verschillende flexibele data stores aan, waaronder Amazon S23, Hadoop Distributed File System, Amazon Dynamo DB.... Andere features zijn Deep Learning mogelijkheden, Hadoop Streaming, inspelen op events (Amazon CloudWatch Events)...

A.2.3 Amazon EMR Spark cluster

Amazon AWS biedt onder de koepel van Amazon EMR de mogelijkheid om een Apache Spark cluster op te zetten. Volgens Amazon kan je Apache Spark gebruiken voor stream processing, d.w.z. het consumeren en verwerken van real-time data van verschillende soorten data streams. Daarnaast is het mogelijk om aan Machine Learning te doen, hiervoor heeft Amazon EMR MLlib. Deze bibliotheek bevat een aantal schaalbare machine learning algoritmes, daarnaast is er de mogelijkheid om nog steeds je eigen bibliotheken te gebruiken. Als laatste biedt Spark nog de mogelijkheid om pure SQL (of HiveQL) queries uit te voeren op een dataset.

A.2.4 Google BigQuery

Google BigQuery biedt de mogelijkheid om gigantische datasets te verwerken in een minimum van tijd in vergelijking met andere Big Data frameworks. Google BigQuery biedt ook de mogelijkheid om SQL queries uit te voeren op de data. Google BigQuery kan geraadpleegd worden via de WebUI, een command line tool (CLI) of door een BigQuery REST API te maken met bijvoorbeld Java, .NET of Python. Er zijn ook een aantal third-party tools om de data te visualiseren of de data in te laden. Het grootste verschil tussen een Amazon AWS Spark cluster is dat Google BigQuery een publieke implementatie is van Dremel, aldus Google (2012).

A.2.5 Google Cloud Dataproc

Google Cloud Dataproc is een cloudservice die de mogelijkheid biedt om eenvoudig met Apache Spark- en Apache Hadoopclusters te werken. Volgens Google (2017) duurt het via Google Dataproc slechts enkele seconden of minuten om bewerkingen uit te voeren die vroeger uren of dagen duurden. Daarnaast is Google Dataproc integreerbaar in alle andere Google Cloud Platform-services.

A.3 Methodologie

Dit onderzoek start eerst met een uitgebreide literatuurstudie naar wat Amazon EMR en Google BigQuery eigenlijk zijn. Daarna worden voor beide technologieën de voor- en nadelen onderzocht. Uiteindelijk worden de mogelijkheden en de beperkingen van de gebruikte tools en technologieën getest. Tijdens deze literatuurstudie wordt ook een korte blik geworpen op Google Cloud Dataproc en Amazon EMR Hbase zodat kan bewezen worden wat een waardig alternatief kan zijn voor Amazon EMR. Als laatste bestaat deze bachelorproef uit een Proof of Concept waarin allerlei basisgegevens (hartslag, aantal stappen, uren slaap...) vanuit verschillende toestellen worden verzameld. Op beide cloud platformen zullen dan analyses gedaan worden van de verzamelde gegevens. Bij deze analyses worden parameters verzameld zoals performantie, geheugengebruik, tijd... van de technologieën. Uit deze resultaten wordt uiteindelijk een conclusie getrokken.

Mogelijke analyses van de verzamelde data zijn:

- De verandering van de hartslag gedurende de dag (Wanneer is deze het hoogste en laagst?)
- De verandering van het aantal stappen gedurende de dag (Wanneer worden er meer stappen afgelegd?)
- Hoe verandert het aantal uren slaap doorheen een week?
- ...

Tijdens deze bachelorproef zal gebruik gemaakt worden van volgende tools en technologieën: Talend, Amazon AWS (Apache Spark en HBase clusters), Google BigQuery, Tableau, Google Cloud Dataproc.... Voor de analyses wordt gebruik gemaakt van Python, R of Talend.

A.4 Verwachte conclusies

Het is moeilijk om aan de hand van de websites van Amazon EMR en Google BigQuery uit te maken welke van de twee de meeste functionaliteiten heeft of welke het makkelijkst in implementatie is. Ze hebben beiden hun sterktes en zwaktes, hun specifieke use cases en doelpubliek. Voor de use case van de Proof of Concept van dit onderzoek wordt verwacht dat de Amazon EMR Spark cluster als beste uit het onderzoek zal komen aangezien Google BigQuery echt ontwikkeld is voor gigantische datasets. Dus er wordt verwacht dat Google Big Query wat overkill is voor de use case van dit onderzoek. Maar er wordt wel verwacht dat wanneer de grootte van de dataset enorm toeneemt, dat Google BigQuery wel de beste van de twee zal zijn. Deze is eigenlijk ontwikkeld voor tera- of petabytes aan informatie en we verwachten dus dat deze performanter zal zijn dan de Amazon EMR Spark cluster. Daarnaast wordt wel verwacht dat Google Dataproc een sterke tegenstander is van de Amazon EMR Spark cluster aangezien dit eigenlijk hetzelfde aanbiedt. De vraag is dan welke van de twee het eenvoudigst en snelst geïmplementeerd is en welke het meeste kost in implementatie. Dit zijn dingen die op voorhand moeilijk te voorspellen aangezien we niet weten hoeveel data er gegenereerd zal worden en hoeveel we van de system zullen moeten gebruiken om de gewenste analyses en transformaties (formatering van de data) uit te voeren.

B. CSV opruimen

In deze appendix wordt de Java code, gebruikt voor het opruimen van de NYC Parking tickets dataset van 2015, weergegeven.

B.1 Vereisten

Om deze code te kunnen uitvoeren zijn er wel enkele vereisten. De eenvoudigste manier is om een maven project aan te maken en onderstaande inhoud van het *pom.xml*-bestand te plakken in u *pom.xml*-bestand.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>be.intodata</groupId>
    <artifactId>dataonboarding</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>9</source>
```

```
                    <target>9</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <!-- Apache commons csv -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-csv</artifactId>
            <version>1.5</version>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.shared</groupId>
            <artifactId>maven-shared-utils</artifactId>
            <version>0.3</version>
        </dependency>
    </dependencies>
</project>
```

Daarnaast is er ook een mapje *data* nodig in de root van dit project. In deze map plakt men het bestand dat men wil opruimen, in dit geval de data van 2015. Indien de bestandsnaam gewijzigd is, gelieve dit aan te passen in de code van het input-bestand.

B.2 Code

Deze code kan simpelweg in een **Main.java**-bestand geplakt worden om zo uit te voeren.

```
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVPrinter;
import org.apache.commons.csv.CSVRecord;

import java.io.BufferedWriter;
import java.io.File;
import java.io.IOException;
import java.io.Reader;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class Main {

    public static final String VEHICLE_EXPIRATION_DATE = "
       Vehicle Expiration Date";
```

```
18      public static final String DATE_FIRST_OBSERVED = "Date
           First Observed";
19
20      // alle nodige keys om te behouden na het opruimen
21      private static String[] keys = {
22              "Summons Number", "Plate ID", "Registration State"
                   , "Plate Type", "Issue Date", "Violation Code",
23              "Vehicle Body Type", "Vehicle Make", "Issuing
                   Agency", "Street Code1", "Street Code2", "
                   Street Code3",
24              "Vehicle Expiration Date", "Violation Location", "
                   Violation Precinct", "Issuer Precinct", "Issuer
                    Code",
25              "Issuer Command", "Issuer Squad", "Violation Time"
                   , "Time First Observed", "Violation County",
26              "Violation In Front Of Or Opposite", "House Number
                   ", "Street Name", "Intersecting Street",
27              "Date First Observed", "Law Section", "Sub
                   Division", "Violation Legal Code", "Days
                   Parking In Effect",
28              "From Hours In Effect", "To Hours In Effect", "
                   Vehicle Color", "Unregistered Vehicle?", "
                   Vehicle Year",
29              "Meter Number", "Feet From Curb", "Violation Post
                   Code", "Violation Description",
30              "No Standing or Stopping Violation", "Hydrant
                   Violation", "Double Parking Violation"
31      };
32
33      public static void main(String[] args) throws IOException
           {
34          // maak een bestand voor de input
35          final File input = new File("data/
                Parking_Violations_Issued_-_Fiscal_Year_2015.csv");
36
37          // maak een bestand voor de output
38          final File output = new File("data/output_2015.csv");
39
40          // Maak een CSVPrinter om de nieuwe csv-records naar
                te schrijven
41          BufferedWriter writer = Files.newBufferedWriter(output
                .toPath());
42          final CSVPrinter csvPrinter = new CSVPrinter(writer,
                CSVFormat.DEFAULT
43                  .withFirstRecordAsHeader());
44
45          // maak de CSVParser voor het input-bestand
46          CSVParser parser = readFile(input);
47          cleanData(parser, csvPrinter);
```

```
48      }
49
50      /**
51       * Lees het meegegeven bestand in en geef een CSVParser
             terug.
52       * Men retourneert een CSVParser omdat het zo mogelijk is
             om
53       * elk record apart in te laden i.p.v. alle records in een
              keer.
54       *
55       * @param file Het bestand om de CSVParser voor te maken
56       * @return Een CSVParser voor het meegegeven bestand
57       * @throws IOException
58       */
59      private static CSVParser readFile(File file) throws
             IOException {
60          Reader reader = Files.newBufferedReader(file.toPath())
                ;
61          return new CSVParser(reader, CSVFormat.newFormat(',').
                withQuote('"').withFirstRecordAsHeader());
62      }
63
64      /**
65       * Overloop alles in de CSVParser en schrijf het record
             meteen weg met de CSVPrinter.
66       * Deze methode update het Vehicle Expiration Date naar
             het juiste formaat en haalt
67       * de foute rijen uit het bestand (bijvoorbeeld een foute
             datum).
68       *
69       * @param parser
70       * @param csvPrinter
71       */
72      private static void cleanData(CSVParser parser, CSVPrinter
             csvPrinter) {
73          // write out the header first
74          try {
75              csvPrinter.printRecord(keys);
76              csvPrinter.flush();
77          } catch (IOException e) {
78              System.out.println(e.getMessage());
79          }
80
81          for (CSVRecord record : parser) {
82              if (record.isConsistent()) { // eerste check of
                    correctheid van de rij
83                  try {
84                      String expirationDate = parseDate(record.
                            get(VEHICLE_EXPIRATION_DATE));
```

```
85                      String dateFirstObserved = parseDate(
                            record.get(DATE_FIRST_OBSERVED));
86
87                      // vorm het record om naar een map en
                            steek de nieuwe waarde erin
88                      Map<String, String> recordMap = record.
                            toMap();
89                      recordMap.put(VEHICLE_EXPIRATION_DATE,
                            expirationDate);
90                      recordMap.put(DATE_FIRST_OBSERVED,
                            dateFirstObserved);
91
92                      // steek alle te behouden kolommen in een
                            lijst
93                      List<String> newRecord = new ArrayList<>()
                            ;
94                      for (String key : keys) {
95                          if (key.equals("Days Parking In Effect
                                ")) {
96                              // this column is not in the file
97                              newRecord.add("");
98                          } else {
99                              newRecord.add(recordMap.get(key));
100                         }
101                     }
102
103                     // schrijf dit record weg
104                     csvPrinter.printRecord(newRecord);
105                     csvPrinter.flush();
106                 } catch (Exception e) {
107                     // mogelijke exceptions:
108                     //       - verkeerd gevormde rij of
                            verkeerd formaat datum
109                     //       - array index out of bound
110                     System.out.println(e.getMessage());
111                 }
112             }
113         }
114     }
115
116     /**
117      * Vorm een datum om naar het gewenste formaat.
118      * Voorbeeld van een datum: 01/01/20151231 12:00:00 PM
119      * Wij hebben nodig: 01/01/2015 om uiteindelijk om te
             vormen naar 20150101
120      * @param dateToParse De datum te omvormen
121      * @return De omgevormde datum
122      */
123     private static String parseDate(String dateToParse) {
```

```
124         String[] parts = dateToParse.split(" ");
125         String firstPart = parts[0];
126         String[] partsOfDate = firstPart.split("/");
127         String year = partsOfDate[2].substring(0, 4);
128 
129         // soms is het jaartal 8888, dit aanvaarden we niet
               als geldig
130         if (year.equals("8888")) {
131             throw new IllegalArgumentException(String.format("
                   Wrong year: %s", year));
132         }
133 
134         return String.format("%s%s%s", year, partsOfDate[1],
               partsOfDate[0]);
135     }
136 }
```

C. Resultaten queries

Hieronder vindt u alle resultaten van elke query. Daarbij is het belangrijk om te weten dat de resultaten uit de queries irrelevant zijn voor dit onderzoek, deze worden enkel gegeven voor de volledigheid maar hebben geen nut in het kader van dit onderzoek.

Query 1: In welke staat worden het meeste tickets uitgeschreven?

Rij	Staat	aantal
1	NY	13 580 718

Tabel C.1: Resultaat query 1

Query 2: Welke inbreuk wordt het meeste vastgesteld?

Rij	Code	aantal
1	21	5 772 887

Tabel C.2: Resultaat query 2

Query 3: Welk politiedistrict schrijft het meeste tickets uit?

Rij	Politiedistrict	aantal
1	T	29 451 961

Tabel C.3: Resultaat query 3

Query 4: Welke violation code komt overeen met welke violation description?

De resultaten van deze query zijn ingekort, slechts de eerste 10 resultaten worden hieronder weergegeven.

Rij	Code	Description
1	1	01-No intercity Pmt Displ
2	2	02-No operator N/A/PH
3	3	03-Unauth passenger pick-up
4	4	04A-Downtown Bus Area, Non-Bus
5	4	04-Downtozn Bus Area, 3 hr Lim
6	4	04B-Downtown Bus Area, No Prmt
7	5	BUS LANE VIOLATION
8	6	06-Nighttime PKG (Trailer)
9	7	FAILURE TO STOP AT RED LIGHT
10	8	08-Engine Idling

Tabel C.4: Resultaat query 4

Query 5: Top 10 van de staten waar het meest een ticket wordt uigeschreven

Rij	Staat	aantal
1	NY	13 580 718
2	K	8 532 548
3	Q	7 414 380
4	BX	4 415 498
5	BK	962 987
6	QN	735 547
7	R	420 059
8	ST	147 943
9	MN	106 034
10	KINGS	21

Tabel C.5: Resultaat query 5

Query 6: Top 10 van het aantal inbreuken per staat per automerk

Rij	Staat	Automerk	aantal
1	NY	FORD	2 444 448
2	NY	CHEVR	1 175 900
3	K	TOYOT	1 075 367
4	K	HONDA	973 419
5	NY	FRUEH	952 629
6	NY	TOYOT	908 619
7	Q	TOYOT	867 098
8	Q	HONDA	852 938
9	K	FORD	851 883
10	K	NISSA	815 503

Tabel C.6: Resultaat query 6

Query 7: Op welke momenten van de dag gebeuren het meeste inbreuken?

De resultaten van deze query zijn ingekort, slechts de eerste 10 resultaten worden hieronder weergegeven. De kolom tijd is geformatteerd als volgt: het uur van de dag gevolgd door de letter A voor AM (voor de middag) of de letter P voor PM (na de middag).

Rij	Tijd	aantal
1	09A	4 318 962
2	11A	4 239 001
3	01P	3 924 483
4	08A	3 755 491
5	12P	3 543 793
6	10A	3 513 950
7	02P	3 361 911
8	03P	2 406 097
9	04P	2 258 608
10	07A	2 034 590

Tabel C.7: Resultaat query 7

Query 8: Een overzicht van het aantal tickets per maand, opgesplitst in New York en andere staten

De resultaten van deze query zijn ingekort, slechts de eerste 10 resultaten worden hieronder weergegeven.

Rij	Maand	Staat	aantal
1	01	NY	1 545 083
2	01	OTHER	2 678 982
3	02	NY	1 385 772
4	02	OTHER	2 313 242
5	03	NY	1 171 947
6	03	OTHER	1 921 334
7	04	NY	1 145 691
8	04	OTHER	1 745 791
9	05	NY	1 147 842
10	05	OTHER	2 014 613

Tabel C.8: Resultaat query 8

Query 9: Maken mensen met een niet-ingeschreven voertuig meer inbreuken? Of net minder?

Rij	Ingeschreven?	aantal
1	ja	35 745 395
2	nee	4 869 362

Tabel C.9: Resultaat query 9

Query 10: Op welke momenten van de dag gebeuren het meeste inbreuken?

Rij	Moment	aantal
1	Voormiddag (9AM tot 12AM)	12 071 886
2	Namiddag (2PM tot 6PM)	9 658 743
3	Ochtend (6AM tot 9AM)	6 638 530
4	Nacht (10PM tot 6AM)	6 053 743
5	Middag (12PM tot 2PM)	3 926 174
6	Avond (6PM tot 10PM)	2 257 437

Tabel C.10: Resultaat query 10

D. Code Google BigQuery

Query 1: In welke staat worden het meeste tickets uitgeschreven?

```
SELECT TOP(ViolationCounty, 1), COUNT(*) AS aantal
FROM nyc_parkings.tickets
WHERE ViolationCounty IS NOT NULL;
```

Query 2: Welke inbreuk wordt het meeste vastgesteld?

```
SELECT TOP(ViolationCode, 1), COUNT(*) AS aantal
FROM nyc_parkings.tickets
WHERE ViolationCode > 0 AND ViolationCode IS NOT NULL;
```

Query 3: Welk politiedistrict schrijft het meeste tickets uit?

```
SELECT TOP(IssuingAgency, 1), COUNT(*) AS aantal
FROM nyc_parkings.tickets
WHERE IssuingAgency IS NOT NULL;
```

Query 4: Welke violation code komt overeen met welke violation description?

```
SELECT ViolationCode, ViolationDescription
FROM nyc_parkings.tickets
WHERE ViolationCode > 0 AND ViolationDescription IS NOT NULL
GROUP BY ViolationCode, ViolationDescription
ORDER BY ViolationCode;
```

Query 5: Top 10 van de staten waar het meest een ticket wordt uigeschreven

```
SELECT TOP(ViolationCounty, 10), COUNT(*) AS aantal
```

```
FROM nyc_parkings.tickets
WHERE ViolationCounty IS NOT NULL;
```

Query 6: Top 10 van het aantal inbreuken per staat per automerk

```
SELECT ViolationCounty, VehicleMake, COUNT(SummonsNumber) as
   aantal
FROM nyc_parkings.tickets
WHERE ViolationCounty IS NOT NULL AND VehicleMake IS NOT NULL
GROUP BY ViolationCounty, VehicleMake
ORDER BY aantal DESC;
```

Query 7: Op welk uur van de dag worden het meeste tickets uitgeschreven?

```
SELECT SUBSTR(ViolationTime, 1, 2) + SUBSTR(ViolationTime, -1,
    1) AS time, COUNT(*) AS aantal
FROM nyc_parkings.tickets
WHERE ViolationTime IS NOT NULL
GROUP BY time
ORDER BY aantal DESC;
```

Query 8: Een overzicht van het aantal tickets per maand, opgesplitst in New York en andere staten

```
SELECT
CASE
WHEN INTEGER(SUBSTR(IssueDate, 4, 2)) > 2 THEN SUBSTR(
   IssueDate, 1, 2)
ELSE SUBSTR(IssueDate, 4, 2)
END AS maand,
CASE
WHEN ViolationCounty = 'NY' THEN 'NY'
ELSE 'OTHER'
END AS staat,
COUNT (*) AS aantal
FROM nyc_parkings.tickets
WHERE ViolationCounty IS NOT NULL
GROUP BY maand, staat
ORDER BY maand, staat, aantal DESC;
```

Query 9: Maken mensen met een niet-ingeschreven voertuig meer inbreuken? Of net minder?

```
SELECT
CASE
WHEN UnregisteredVehicle = '0' THEN 'no'
ELSE 'yes'
END AS registred, COUNT(*) AS aantal
FROM nyc_parkings.tickets
GROUP BY registred
ORDER BY aantal DESC;
```

Query 10: Op welke momenten van de dag gebeuren het meeste inbreuken?

```
SELECT
CASE
WHEN SUBSTR(ViolationTime, -1, 1) = 'A' AND INTEGER(SUBSTR(
   ViolationTime, 1, 2)) < 6 THEN 'Nacht (10PM tot 6AM)'
WHEN SUBSTR(ViolationTime, -1, 1) = 'A' AND INTEGER(SUBSTR(
   ViolationTime, 1, 2)) < 9 THEN 'Ochtend (6AM tot 9AM)'
WHEN SUBSTR(ViolationTime, -1, 1) = 'A' AND INTEGER(SUBSTR(
   ViolationTime, 1, 2)) < 12 THEN 'Voormiddag (9AM tot 12AM)'
WHEN SUBSTR(ViolationTime, -1, 1) = 'P' AND INTEGER(SUBSTR(
   ViolationTime, 1, 2)) < 2 THEN 'Middag (12PM tot 2PM)'
WHEN SUBSTR(ViolationTime, -1, 1) = 'P' AND INTEGER(SUBSTR(
   ViolationTime, 1, 2)) < 6 THEN 'Namiddag (2PM tot 6PM)'
WHEN SUBSTR(ViolationTime, -1, 1) = 'P' AND INTEGER(SUBSTR(
   ViolationTime, 1, 2)) < 10 THEN 'Avond (6PM tot 10PM)'
ELSE 'Nacht (10PM tot 6AM)'
END AS moment,
COUNT (*) AS aantal
FROM nyc_parkings.tickets
WHERE ViolationTime IS NOT NULL AND NOT (ViolationTime
   CONTAINS '.') AND NOT (ViolationTime CONTAINS '0+')
GROUP BY moment
ORDER BY aantal DESC;
```

E. Code Google Cloud Dataproc

```
from __future__ import absolute_import
import json
import pprint
import subprocess
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row
import time

currentTimeInMillis = lambda: int(round(time.time() * 1000))

sc = SparkContext.getOrCreate()

# Gebruik een Google Cloud Storage bucket voor tijdelijk
    opslag van data
bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.
    bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id'
    )

# Maak een tijdelijke map aan voor de input bestanden van
    Hadoop (omdat we de data importeren van BigQuery)
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.
    format(bucket)

# Maak een tijdelijke map aan voor de uitvoeringstijden
output_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_output
    '.format(bucket)
output_files = output_directory + '/part-*'
```

```
23
24
25 # definieer alle nodige mappers voor straks
26 def formatCountyAndIssueDate(record):
27     # take the month like in a Belgian date format (dd/MM/yyyy
       )
28     month = record[0][0][3:5]
29     # if the month is to big, take the first two numbers (
       format: MM/dd/yyyy)
30     if (int(month) > 12):
31         month = record[0][0][:2]
32     if (record[0][1] is 'NY'):
33         return ((month, 'NY'), record[1])
34     else:
35         return ((month, 'OTHER'), record[1])
36
37 def formatRegistered(record):
38     if (record[0] is '0'):
39         return ('no', record[1])
40     else:
41         return ('yes', record[1])
42
43 def formatViolationTime(record):
44     hour = int(record[0][:2])
45
46     if record[0][-1:] is 'A':
47         if hour < 6:
48             return ('Nacht (10PM tot 6AM)', record[1])
49         elif hour < 9:
50             return ('Ochtend (6AM tot 9AM)', record[1])
51         else:
52             return ('Voormiddag (9 AM tot 12AM)', record[1])
53     else:
54         if hour < 2:
55             return ('Middag (12PM tot 2PM)', record[1])
56         elif hour < 6:
57             return ('Namiddag (2PM tot 6PM)', record[1])
58         elif hour < 10:
59             return ('Avond (6PM tot 10PM)', record[1])
60         else:
61             return ('Nacht (10PM tot 6AM)', record[1])
62
63 # Maak de configuratie voor het importeren van data van
     BigQuery
64 conf = {
65     # Input Parameters.
66     'mapred.bq.project.id': project,
67     'mapred.bq.gcs.bucket': bucket,
68     'mapred.bq.temp.gcs.path': input_directory,
```

```
69      'mapred.bq.input.project.id': 'bachelorproef-199913',
70      'mapred.bq.input.dataset.id': 'nyc_parkings',
71      'mapred.bq.input.table.id': 'tickets',
72  }
73
74  # Definieer de output parameters
75  output_dataset = 'test_dataset'
76  output_table = 'test_output'
77
78  # Laad de data in van BigQuery
79  table_data = sc.newAPIHadoopRDD(
80      'com.google.cloud.hadoop.io.bigquery.
            JsonTextBigQueryInputFormat',
81      'org.apache.hadoop.io.LongWritable',
82      'com.google.gson.JsonObject',
83      conf=conf)
84
85  # Neem enkel de JSON, niet het rijnummer (rijnummer staat op
        index 0)
86  table_data = table_data.map(lambda record: json.loads(record
        [1]))
87
88  # Maak een array aan voor alle uitvoeringstijden van de
        queries
89  queryTimes = []
90
91  for x in range(0, 100):
92      queryTimes = []
93
94      # QUERY 1 In welke staat worden het meeste tickets
            uitgeschreven?
95      start = currentTimeInMillis()
96      (table_data
97          .map(lambda record: (record.get('ViolationCounty', '')
                , 1))
98          .filter(lambda record: (record[0] is not None) | (
                record[0] is not ''))
99          .reduceByKey(lambda x, y: x + y)
100         .sortBy(lambda r: -r[1])
101         .take(1))
102     duration = currentTimeInMillis() - start
103     list.append(queryTimes, duration)
104
105     # QUERY 2 Welke inbreuk wordt het meeste vastgesteld?
106     start = currentTimeInMillis()
107     (table_data
108         .map(lambda record: (record['ViolationCode'], 1))
109         .filter(lambda record: (record[0] is not None) & (int(
                record[0]) > 0))
```

```
110         .reduceByKey(lambda x, y: x + y)
111         .sortBy(lambda r: -r[1])
112         .take(1))
113     duration = currentTimeInMillis() - start
114     list.append(queryTimes, duration)
115 
116     # QUERY 3 Welk politiedistrict schrijft het meeste tickets
            uit?
117     start = currentTimeInMillis()
118     (table_data
119         .map(lambda record: (record['IssuingAgency'], 1))
120         .filter(lambda record: (record[0] is not None) | (
                record[0] is not ''))
121         .reduceByKey(lambda x, y: x + y)
122         .sortBy(lambda r: -r[1])
123         .take(1))
124     duration = currentTimeInMillis() - start
125     list.append(queryTimes, duration)
126 
127     # QUERY 4 Welke violation code komt overeen met welke
            violation description?
128     start = currentTimeInMillis()
129     (table_data
130         .map(lambda record: (record.get('ViolationCode', 0),
                record.get('ViolationDescription', '')))
131         .filter(lambda record: ((int(record[0]) > 0)) & ((
                record[1] is not None) | (record[1] is not '')))
132         .reduceByKey(lambda r1, r2: r1)
133         .take(10))
134     duration = currentTimeInMillis() - start
135     list.append(queryTimes, duration)
136 
137     # QUERY 5 Top 10 van de staten waar het meest een ticket
            wordt uigeschreven
138     start = currentTimeInMillis()
139     (table_data
140         .map(lambda record: (record.get('ViolationCounty', '')
                , 1))
141         .filter(lambda record: (record[0] is not None) | (
                record[0] is not ''))
142         .reduceByKey(lambda x, y: x + y)
143         .sortBy(lambda r: -r[1])
144         .take(10))
145     duration = currentTimeInMillis() - start
146     list.append(queryTimes, duration)
147 
148     # QUERY 6 Op welk uur van de dag worden het meeste tickets
            uitgeschreven?
149     start = currentTimeInMillis()
```

```
150     (table_data
151         .map(lambda record: ((record.get('ViolationCounty', '')
              ), record.get('VehicleMake', '')), 1))
152         .filter(lambda record: ((record[0][0] is not None) | (
              record[0][0] is not '')) & ((record[0][1] is not
              None) | (record[0][1] is not '')))
153         .reduceByKey(lambda x, y: x + y)
154         .sortBy(lambda r: -r[1])
155         .take(10))
156     duration = currentTimeInMillis() - start
157     list.append(queryTimes, duration)
158 
159     # QUERY 7 Op welke momenten van de dag gebeuren het meeste
          inbreuken?
160     start = currentTimeInMillis()
161     (table_data
162         .map(lambda record: (record.get('ViolationTime', ''),
              1))
163         .filter(lambda record: (record[0] is not None) | (
              record[0] is not ''))
164         .map(lambda record: (record[0][:2] + record[0][-1:],
              record[1]))
165         .reduceByKey(lambda x, y: x + y)
166         .sortBy(lambda r: -r[1])
167         .take(10))
168     duration = currentTimeInMillis() - start
169     list.append(queryTimes, duration)
170 
171     # QUERY 8 Een overzicht van het aantal tickets per maand,
          opgesplitst in New York en andere staten
172     start = currentTimeInMillis()
173     (table_data
174         .map(lambda record: ((record['IssueDate'], record.get(
              'ViolationCounty', '')), 1))
175         .filter(lambda record: ((record[0][0] is not None) | (
              record[0][0] is not '')) & ((record[0][1] is not
              None) | (record[0][1] is not '')))
176         .map(formatCountyAndIssueDate)
177         .reduceByKey(lambda x, y: x + y)
178         .sortBy(lambda r: r[0][0])
179         .take(10))
180     duration = currentTimeInMillis() - start
181     list.append(queryTimes, duration)
182 
183     # QUERY 9 Maken mensen met een niet-ingeschreven voertuig
          meer inbreuken? Of net minder?
184     start = currentTimeInMillis()
185     (table_data
186         .map(lambda record: (record.get('UnregisteredVehicle',
```

```
            ''), 1))
187         .filter(lambda record: (record[0] is not None) | (
                record[0] is not ''))
188         .map(formatRegistered)
189         .reduceByKey(lambda x, y: x + y)
190         .sortBy(lambda r: -r[1])
191         .take(10))
192     duration = currentTimeInMillis() - start
193     list.append(queryTimes, duration)
194
195     # QUERY 10 Op welke momenten van de dag gebeuren het
            meeste inbreuken?
196     start = currentTimeInMillis()
197     (table_data
198         .map(lambda record: (record.get('ViolationTime', ''),
                1))
199         .filter(lambda record: (record[0] is not None) & (len(
                record[0]) >= 3) & ('+' not in record[0]))
200         .filter(lambda record: ('.' not in record[0]))
201         .map(formatViolationTime)
202         .reduceByKey(lambda x, y: x + y)
203         .sortBy(lambda r: -r[1])
204         .take(10))
205     duration = currentTimeInMillis() - start
206     list.append(queryTimes, duration)
207
208     # Print de uitvoeringstijden en maak er een RDD met ÃⓒÃⓒn
            Row van
209     pprint.pprint(queryTimes)
210     pprint.pprint("Iteration {} finished".format(x))
211     rdd = sc.parallelize([Row(query1=queryTimes[0], query2=
            queryTimes[1], query3=queryTimes[2], query4=queryTimes
            [3], query5=queryTimes[4], query6=queryTimes[5], query7
            =queryTimes[6], query8=queryTimes[7], query9=queryTimes
            [8], query10=queryTimes[9])])
212
213     # Map alle uitvoeringstijden naar een dataframe om het zo
            weg te schrijven naar Google BigQuery
214     sql_context = SQLContext.getOrCreate(sc)
215     (sql_context.createDataFrame(rdd)
216      .write.format('json').save(output_directory))
217
218      # Stuur de uitvoeringstijden van deze 10 queries naar
             BigQuery
219     try:
220         subprocess.check_call(
221             'bq load --source_format NEWLINE_DELIMITED_JSON '
222             '--noreplace '
223             '--autodetect '
```

```
224             '{dataset}.{table} {files}'.format(dataset=
                    output_dataset, table=output_table, files=
                    output_files).split())
225     except subprocess.CalledProcessError as e:
226         raise RuntimeError("command '{}' return with error (
                code {}): {}".format(e.cmd, e.returncode, e.output)
                )
227
228     # Verwijder de output map zodat de volgende iteratie niet
           faalt
229     output_path = sc._jvm.org.apache.hadoop.fs.Path(
           output_directory)
230     output_path.getFileSystem(sc._jsc.hadoopConfiguration()).
           delete(
231         output_path, True)
232
233 # Ruim alle tijdelijke bestanden op, anders blijven deze
       bestaan en zal een volgende uitvoering van dit script falen
234 # omdat de bestanden nog aanwezig zijn
235 input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory
       )
236 input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete
       (input_path, True)
237 output_path = sc._jvm.org.apache.hadoop.fs.Path(
       output_directory)
238 output_path.getFileSystem(sc._jsc.hadoopConfiguration()).
       delete(
239     output_path, True)
```

F. Code Amazon EMR

```
import pprint
import time
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

sc = SparkContext.getOrCreate()
sqlContext = SQLContext.getOrCreate(sc)

# maak een functie om de huidige tijd in milliseconden te
   bepalen
currentTimeInMillis = lambda: int(round(time.time() * 1000))

# definieer alle nodige mappers voor straks
def formatCountyAndIssueDate(record):
    # take the month like in a Belgian date format (dd/MM/yyyy
       )
    month = record[0][0][3:5]
    # if the month is to big, take the first two numbers (
       format: MM/dd/yyyy)
    if (int(month) > 12):
        month = record[0][0][:2]
    if (record[0][1] is 'NY'):
        return ((month, 'NY'), record[1])
    else:
        return ((month, 'OTHER'), record[1])

def formatRegistered(record):
    if (record[0] is '0'):
```

```
26          return ('no', record[1])
27      else:
28          return ('yes', record[1])
29
30  def formatViolationTime(record):
31      hour = int(record[0][:2])
32
33      if record[0][-1:] is 'A':
34          if hour < 6:
35              return ('Nacht (10PM tot 6AM)', record[1])
36          elif hour < 9:
37              return ('Ochtend (6AM tot 9AM)', record[1])
38          else:
39              return ('Voormiddag (9 AM tot 12AM)', record[1])
40      else:
41          if hour < 2:
42              return ('Middag (12PM tot 2PM)', record[1])
43          elif hour < 6:
44              return ('Namiddag (2PM tot 6PM)', record[1])
45          elif hour < 10:
46              return ('Avond (6PM tot 10PM)', record[1])
47          else:
48              return ('Nacht (10PM tot 6AM)', record[1])
49
50  # access en secret key van een gebruiker met voldoende rechten
51  access_key = "YOUR-ACCESS-KEY"
52  secret_key = "YOUR-SECRET-KEY"
53
54  # voeg de access en secret key toe aan de configuratie van
        Hadoop
55  sc._jsc.hadoopConfiguration().set("fs.s3a.access.key",
        access_key)
56  sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key",
        secret_key)
57
58  # maak de url naar de S3 Bucket waarop alle csv bestanden
        staan
59  # en maak er een RDD van a.d.h.v. een dataframe, hierdoor zijn
        alle types van de kolommen reeds correct
60  file_url = "s3a://thomas-aelbrecht-stage/*.csv"
61
62  # verwijder de header van het csv bestand
63  df = (sqlContext
64      .read
65      .format("com.databricks.spark.csv")
66      .option("header", "true")
67      .option("mode", "DROPMALFORMED")
68      .load(file_url))
69
```

```
70 # maak een RDD van het dataframe en map elke rij naar een
      tupel i.p.v. een instantie van Row (uit pyspark.sql)
71 table_data = df.rdd.map(tuple)
72
73 # maak een array om de uitvoeringstijden van de huidige
      iteratie bij te houden
74 queryTimes = []
75
76 for x in range(0, 100):
77     queryTimes = []
78
79     # QUERY 1 In welke staat worden het meeste tickets
          uitgeschreven?
80     start = currentTimeInMillis()
81     (table_data
82     .map(lambda record: (record[21], 1))
83     .filter(lambda record: (record[0] is not None) | (record
          [0] is not ''))
84     .reduceByKey(lambda x, y: x + y)
85     .sortBy(lambda r: -r[1])
86     .take(1))
87     duration = currentTimeInMillis() - start
88     list.append(queryTimes, duration)
89
90     # QUERY 2 Welke inbreuk wordt het meeste vastgesteld?
91     start = currentTimeInMillis()
92     (table_data
93     .map(lambda record: (record[5], 1))
94     .filter(lambda record: (record[0] is not None) | (int(
          record[0]) > 0))
95     .reduceByKey(lambda x, y: x + y)
96     .sortBy(lambda r: -r[1])
97     .take(1))
98     duration = currentTimeInMillis() - start
99     list.append(queryTimes, duration)
100
101    # QUERY 3 Welk politiedistrict schrijft het meeste tickets
          uit?
102    start = currentTimeInMillis()
103    (table_data
104    .map(lambda record: (record[8], 1))
105    .filter(lambda record: (record[0] is not None) | (record
          [0] is not ''))
106    .reduceByKey(lambda x, y: x + y)
107    .sortBy(lambda r: -r[1])
108    .take(1))
109    duration = currentTimeInMillis() - start
110    list.append(queryTimes, duration)
111
```

```
112     # QUERY 4 Welke violation code komt overeen met welke
            violation description?
113     start = currentTimeInMillis()
114     (table_data
115     .map(lambda record: (record[5], record[39]))
116     .filter(lambda record: ((int(record[0]) > 0)) & ((record
            [1] is not None) | (record[1] is not '')))
117     .reduceByKey(lambda r1, r2: r1)
118     .take(10))
119     duration = currentTimeInMillis() - start
120     list.append(queryTimes, duration)
121
122     # QUERY 5 Top 10 van de staten waar het meest een ticket
            wordt uigeschreven
123     start = currentTimeInMillis()
124     (table_data
125     .map(lambda record: (record[21], 1))
126     .filter(lambda record: (record[0] is not None) | (record
            [0] is not ''))
127     .reduceByKey(lambda x, y: x + y)
128     .sortBy(lambda r: -r[1])
129     .take(10))
130     duration = currentTimeInMillis() - start
131     list.append(queryTimes, duration)
132
133     # QUERY 6 Top 10 van het aantal inbreuken per staat per
            automerk
134     start = currentTimeInMillis()
135     (table_data
136     .map(lambda record: ((record[21], record[8]), 1))
137     .filter(lambda record: ((record[0][0] is not None) | (
            record[0][0] is not '')) & ((record[0][1] is not None)
            | (record[0][1] is not '')))
138     .reduceByKey(lambda x, y: x + y)
139     .sortBy(lambda r: -r[1])
140     .take(10))
141     duration = currentTimeInMillis() - start
142     list.append(queryTimes, duration)
143
144     # QUERY 7 Op welke momenten van de dag gebeuren het meeste
            inbreuken?
145     start = currentTimeInMillis()
146     (table_data
147     .map(lambda record: (record[19], 1))
148     .filter(lambda record: record[0] is not None)
149     .filter(lambda record: record[0] is not '')
150     .map(lambda record: (record[0][:2] + record[0][-1:],
            record[1]))
151     .reduceByKey(lambda x, y: x + y)
```

```
152     .sortBy(lambda r: -r[1])
153     .take(10))
154     duration = currentTimeInMillis() - start
155     list.append(queryTimes, duration)
156 
157     # QUERY 8 Een overzicht van het aantal tickets per maand,
           opgesplitst in New York en andere staten
158     start = currentTimeInMillis()
159     (table_data
160     .map(lambda record: ((record[4], record[21]), 1))\
161     .filter(lambda record: record[0][0] is not None)
162     .filter(lambda record: record[0][0] is not '')
163     .filter(lambda record: record[0][1] is not None)
164     .filter(lambda record: record[0][1] is not '')
165     .map(formatCountyAndIssueDate)
166     .reduceByKey(lambda x, y: x + y)
167     .sortBy(lambda r: r[0][0])
168     .take(10))
169     duration = currentTimeInMillis() - start
170     list.append(queryTimes, duration)
171 
172     # QUERY 9 Maken mensen met een niet-ingeschreven voertuig
           meer inbreuken? Of net minder?
173     start = currentTimeInMillis()
174     (table_data
175     .map(lambda record: (record[34], 1))
176     .filter(lambda record: record[0] is not None)
177     .filter(lambda record: record[0] is not '')
178     .map(formatRegistered)
179     .reduceByKey(lambda x, y: x + y)
180     .sortBy(lambda r: -r[1])
181     .take(10))
182     duration = currentTimeInMillis() - start
183     list.append(queryTimes, duration)
184 
185     # QUERY 10 Op welke momenten van de dag gebeuren het
           meeste inbreuken?
186     start = currentTimeInMillis()
187     (table_data
188     .map(lambda record: (record[19], 1))
189     .filter(lambda record: record[0] is not None)
190     .filter(lambda record: (len(record[0]) >= 3) & ('+' not in
            record[0]))
191     .filter(lambda record: ('.' not in record[0]))
192     .map(formatViolationTime)
193     .reduceByKey(lambda x, y: x + y)
194     .sortBy(lambda r: -r[1])
195     .take(10))
196     duration = currentTimeInMillis() - start
```

```
197     list.append(queryTimes, duration)
198
199     # voeg alle uitvoeringstijden aan de een csv-bestand in
            onze S3 bucket toe
200     pprint.pprint(queryTimes)
201     pprint.pprint("Iteration {} finished".format(x))
202     rdd = sc.parallelize([Row(query1=queryTimes[0], query2=
            queryTimes[1], query3=queryTimes[2], query4=queryTimes
            [3], query5=queryTimes[4], query6=queryTimes[5], query7
            =queryTimes[6], query8=queryTimes[7], query9=queryTimes
            [8], query10=queryTimes[9])])
203     (sqlContext.createDataFrame(rdd)
204     .write.mode("append").format('csv').save("s3a://thomas-
            aelbrecht-stage/executionTimes.csv"))
```

Bibliografie

Adam, H. (2017, maart 26). Back to Basics: What is Data Onboarding? Verkregen van https://www.lotame.com/back-basics-data-onboarding/

Amazon. (2017a). Amazon EMR Product Details. Verkregen van https://aws.amazon.com/emr/details/

Amazon. (2017b). Apache Spark om Amazon EMR. Verkregen van https://aws.amazon.com/emr/details/spark/

Amazon. (2017c, mei 2). Data warehousing in the era of Big Data: Deep Dive into Amazon Redshift. Verkregen van https://www.slideshare.net/AmazonWebServices/data-warehousing-in-the-era-of-big-data-deep-dive-into-amazon-redshift

Amazon. (2018a). Amazon EMR Product Details. Verkregen van https://aws.amazon.com/emr/details/

Amazon. (2018b). Overview of Amazon EMR. Verkregen van https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html

Apache. (2017). Apache Nutch. Verkregen van http://nutch.apache.org/

Bappalige, S. P. (2014, augustus 26). An introduction to Apache Hadoop for big data. Verkregen van https://opensource.com/life/14/8/intro-apache-hadoop-big-data

Consortium, A. B. (2008). MoSCoW Prioritisation. Verkregen van https://www.agilebusiness.org/content/moscow-prioritisation-0

DataFlair. (2017, april 1). Lazy Evaluation in Apache Spark – A Quick guide. Verkregen van https://data-flair.training/blogs/apache-spark-lazy-evaluation/

Google. (2012). An Inside Look at Google BigQuery. Verkregen van https://cloud.google.com/files/BigQueryTechnicalWP.pdf

Google. (2017). Cloud Dataproc. Verkregen van https://cloud.google.com/dataproc/

Google. (2018a). Cloud Dataflow. Verkregen van https://cloud.google.com/dataflow/

Google. (2018b). Cloud Dataproc. Verkregen van https://cloud.google.com/dataproc/?hl=nl

Google. (2018c). Cloud Dataproc Pricing. Verkregen van https://cloud.google.com/dataproc/pricing

IBM. (2018). What is MapReduce? Verkregen van https://www.ibm.com/analytics/hadoop/mapreduce

Intellipaat. (g.d.). Loading and Saving your Data. Verkregen van https://intellipaat.com/tutorial/spark-tutorial/loading-and-saving-your-data/

Jang, C. (2016, maart 31). Google Cloud Platform: Data & Analytics. Verkregen van https://www.slideshare.net/cjang99/google-cloud-platform-rockplace-big-data-eventmar312016

Kshirsagar, M. M. (2016, september 17). Cortana Intelligence Suite Services Workshop. Verkregen van https://blogs.msdn.microsoft.com/maheshkshirsagar/2016/09/17/cortanaintelligencecontent/

Langit, L. (2013, mei 18). Hadoop MapReduce Fundamentals. Verkregen van https://www.slideshare.net/lynnlangit/hadoop-mapreduce-fundamentals-21427224/77-Comparing_RDBMS_vs_HadoopTraditional_RDBMS

Lardinois, F. (2013, september 18). Google's BigQuery Introduces Streaming Inserts And Time-Based Queries For Real-Time Analytics. Verkregen van https://techcrunch.com/2013/09/18/googles-bigquery-introduces-streaming-inserts-and-time-based-queries-for-real-time-analytics/

Laskowski, J. (g.d.). RDD — Resilient Distributed Dataset. Verkregen van https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd.html

Laufer, A. (2018, februari 5). Autoscaling Google Dataproc Clusters. Verkregen van https://blog.doit-intl.com/autoscaling-google-dataproc-clusters-21f34beaf8a3

Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M. & Vassilakis, T. (2010). Dremel: Interactive Analysis of Web-Scale Datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases* (pp. 330–339). Verkregen van http://www.vldb2010.org/accept.htm

Nucleus. (2017, januari 16). Wat is het verschil tussen IaaS, PaaS, SaaS en UaaS. Verkregen van https://www.nucleus.be/blog/uptime-as-a-service/verschil-iaas-paas-saas-en-uaas/

Rouse, M. (2013). Apache HBase. Verkregen van http://searchdatamanagement.techtarget.com/definition/Apache-HBase

Rouse, M. (2017). Amazon Elastic MapReduce (Amazon EMR). Verkregen van http://searchaws.techtarget.com/definition/Amazon-Elastic-MapReduce-Amazon-EMR

Sato, K. (2012). *An Inside Look at Google BigQuery*. Google.

Shailna, P. (2017, april 26). Data Block in HDFS | HDFS Blocks & Data Block Size. Verkregen van https://data-flair.training/blogs/data-block/

Tereshko, T. (2017, januari 5). Why Dataproc — Google's managed Hadoop and Spark offering is a game changer. Verkregen van https://hackernoon.com/why-dataproc-googles-managed-hadoop-and-spark-offering-is-a-game-changer-9f0ed183fda3

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... Scott Shenker, I. S. (2012, april 25). *Resilient Distributed Datasets: A Fault-Tolerant Abstraction forIn-Memory Cluster Computing*. University of California, Berkeley.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... Stoica, I. (2016). Apache Spark: A Unified Engine For Big Data Processing. *Communications of the*

ACM, *59*(11). Verkregen van https://cacm.acm.org/magazines/2016/11/209116-apache-spark/abstract