



HoGent

Faculteit Bedrijf en Organisatie

NoSQL: Cassandra

Lorenz Verschingel

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Sabine De Vreese
Co-promotor:
Jean-Jacques De Clercq

Instelling: HoGent

Academiejaar: 2015-2016

Tweede examenperiode

Faculteit Bedrijf en Organisatie

NoSQL: Cassandra

Lorenz Verschingel

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Sabine De Vreese
Co-promotor:
Jean-Jacques De Clercq

Instelling: HoGent

Academiejaar: 2015-2016

Tweede examenperiode

Samenvatting

In de laatste jaren zijn de NoSQL databases aan een enorme opmars bezig. Hierbij stellen bepaalde van deze databanken hun gebruiksgemak en kunnen nogal vaak zeer positief voor.

In deze bachelorproef wordt onderzoek gedaan naar de schaalbaarheid en de betrouwbaarheid van de NoSQL database Cassandra. Dit onderzoek wordt uitgevoerd aan de hand van een virtuele cluster die opgezet is met Vagrant. Een ander punt dat onderzocht werd was hoe data modellering in Cassandra in zijn werk gaat.

De schaalbaarheid wordt nagegaan door te kijken hoe gemakkelijk het is om een cluster op te zetten en om hieraan later nodes toe te voegen en te verwijderen. In een virtuele omgeving is dit zeer gemakkelijk na te gaan. Hierbij werd vastgesteld dat het opzetten van de cluster, het toevoegen van nodes en het verwijderen van nodes niet zo gemakkelijk gaat als wordt voorgesteld als men Apache Cassandra gebruikt. Als men hier echter hulpprogramma's gaat gebruiken zoals het OpsCenter van DataStax gaat dit wel zeer vlot.

Om de betrouwbaarheid na te gaan werden opzettelijk nodes uitgezet om te kijken hoe Cassandra hierop reageert. In deze testen doet Cassandra exact wat beloofd wordt. De data blijft beschikbaar en wijzigingen worden bij het online komen van de node doorgegeven.

Bij de data modellering werden enkele eigenaardigheden vastgesteld zoals het feit dat de primaire sleutel restricties oplegt aan de WHERE clause. Hierdoor moet men de data binnen Cassandra modelleren naar de query's die uitgevoerd zullen worden.

Voorwoord

Deze bachelorproef markeert het einde van mijn driejarige opleiding Toegepaste Informatica aan de Hogeschool Gent. Ik heb dit onderwerp, de NoSQL databank Cassandra, gekozen omdat ik er tijdens mijn stage mee aan de slag moest en deze technologie me enorm aansprak.

Het schrijven van een bachelorproef neemt veel tijd in beslag en zonder de hulp van heel wat mensen zou ik dit veel moeilijker tot een goed einde gebracht hebben. Aan hen wil ik mijn grote dank uitdrukken.

Eerst en vooral wil ik de universiteit Gent bedanken voor het aanbieden van de data en in het bijzonder mijn co-promotor, de heer Jean-Jacques De Clercq, voor de technische ondersteuning en het aanhalen van het bedrijf DataStax. Ik wil hem ook bedanken voor het sturen van het onderwerp in de richting van de mogelijkheden van Cassandra.

Verder wil ik ook mevrouw Sabine De Vreese, de promotor van deze bachelorproef bedanken voor de ondersteuning bij het opstellen van de planning en de opvolging vanuit de school.

Ten slotte wil ik ook nog mijn familie en vrienden bedanken voor de hulp bij het nalezen van mij bachelorproef, de morele ondersteuning, het eenvoudig formuleren van bepaalde ingewikkelde technische aspecten. . .

Tijdens deze bachelorproef heb ik veel bijgeleerd en ik hoop dat ook anderen informatie uit deze bachelorproef zullen kunnen halen.

Lorenz Verschingel

Student Toegepaste Informatica

2016

Inhoudsopgave

1	Inleiding	4
1.1	NoSQL	4
1.2	Cassandra	5
1.3	Probleemstelling en Onderzoeksvragen	6
2	Methodologie	7
3	Architectuur	9
3.1	Partitionering	9
3.1.1	Problemen met de ByteOrderedPartitioner	10
3.2	Replicatie	10
3.2.1	Replicatie strategieën	10
3.3	Snitches	12
3.4	Seed node	13
4	Data structuren in Cassandra	14
4.1	Column	14
4.2	Row	15
4.3	Table	15
4.4	Keyspace	15
5	Opzetten van de Cassandra cluster	16
5.1	Apache Cassandra	16
5.2	DataStax OpsCenter	16
5.3	Toevoegen van een node	20
5.4	Verwijderen van een node	21
6	Datamodellering in Cassandra	24
6.1	Primaire sleutel	24
6.2	Partitie kolom	24
6.3	Clustering kolom	25

6.4	De WHERE clause	25
6.4.1	Restricties opgelegd door de partitie kolommen	25
6.4.2	Restricties opgelegd door de clustering kolommen	26
6.5	Doelen bij datamodellering in Cassandra	27
6.6	Datamodel opstellen voor Cassandra	27
7	Data importeren in Cassandra	30
7.1	Importeren via cqlsh	30
7.2	Importeren via sstableloader	31
7.3	Importeren via cassandra-loader	31
7.4	Testen van de verschillende loaders	31
8	Gedrag bij uitvallen van een node	33
8.1	Gossip en foutdetectie	33
8.2	Herstelmechanisme	33
8.2.1	Hinted handoff	34
8.2.2	Anti-entropy repair	34
8.2.3	Read repair	34
8.3	Ondervindingen bij het uitzetten van een node	35
9	Back-ups in Cassandra	37
9.1	Snapshots	37
9.2	Nood aan back-ups in Cassandra	38
10	Use cases voor Cassandra	39
10.1	Messaging	39
10.2	Fraude bestrijding	39
10.3	Product catalogi en afspeellijsten	40
10.4	Internet Of Things en sensor data	40
10.5	Aanbevelingen en personalisatie	40
10.6	Markten voor Cassandra	41
11	Conclusie	42

Hoofdstuk 1

Inleiding

1.1 NoSQL

Sinds de term in 1998 voor het eerst gebruikt werd door Carlo Strozzi, ging een grote bal aan het rollen rond NoSQL databanken. Toen hij deze term de wereld instuurde, bedoelde Strozzi dat de databank die op dat moment besproken werd geen SQL interface aanbood. In 2009 werd de term NoSQL opnieuw gebruikt door Johan Oskarsson als hashtag voor een meetup waar de problemen met relationele databanken en de huidige manier van programmeren besproken gingen worden. Nu wordt NoSQL begrepen als 'Not only SQL', wat erop wijst dat er meerdere manieren zijn om data op te slaan. (Fowler, 2013)

NoSQL databanken vinden hun oorsprong in de verschillen tussen een relationeel model en het object georiënteerd programmeren, het zogenoemde 'impedance mismatch', het feit dat relationele databanken vaak niet goed werken op een cluster of niet met grote hoeveelheden realtime data om kunnen gaan... Hierdoor maken NoSQL databanken vaak geen gebruik van een relationeel model, zijn ze gemaakt om op clusters te werken, zijn ze schema-less... Kortom databanken onder de noemer NoSQL zijn aangepast om de problemen, die zich nu binnen de Informatica manifesteren, aan te pakken (Fowler, 2012).

Sadalage (2014) zegt dat binnen de NoSQL databanken vier grote types naar voor geschoven zijn, namelijk key-value stores (Riak, Redis...), document stores (MongoDB, CouchDB...), Column Family Stores (Cassandra, HBase...) en Graph databases (Neo4J, Infinite Graph...). Elk van deze types heeft zijn eigen specifieke use cases. Zelfs binnen de verschillende types NoSQL databanken komen er nog verschillende specifieke use cases voor.

Binnen NoSQL geldt het CAP theorema.

- **Consistency:** De data is op ieder moment op alle nodes gelijk
- **Availability:** Als een node uitvalt, dan beïnvloedt dit de andere nodes niet.
- **Partition Tolerance:** Het systeem blijft werken ook al zijn er netwerkfouten.

Brewer (2000) stelde dat het onmogelijk was om binnen een gedistribueerd systeem deze drie zaken samen te bekomen. Hij gaf hiermee de aanzet dat er slechts twee van de drie punten in het CAP theorema verkregen konden worden.

1.2 Cassandra

In het vervolg van deze bachelorproef wordt de focus gelegd op de NoSQL databank Cassandra. Cassandra is een Column Family database die focust op schaalbaarheid en beschikbaarheid, zonder aan prestatie in te boeten. Cassandra is op dit moment een productiewaardige database. Enkele bekende gebruikers van Cassandra zijn Facebook, Apple, Netflix, GitHub, Instagram, GoDaddy...

Cassandra is een project dat zijn oorsprong vond bij Facebook. In 2008 was Cassandra, bedacht door Lakshman en Malik, de oplossing voor het Inbox Search probleem van Facebook. De moeilijkheid hierbij was dat er een systeem nodig was die een hoge throughput nodig heeft, die miljarden write operaties per dag moet aankunnen en die kan mee schalen met het aantal gebruikers (Lakshman and Malik, 2010). Om tot deze oplossing te komen baseerden Lakshman en Malik zich op twee andere projecten.

Een eerste project waarop Cassandra gebaseerd is, is Google BigTable. Google BigTable had namelijk al een oplossing voor een eerste probleem dat Lakshman en Malik moesten oplossen: een schaalbare database die geen realtime antwoorden opoffert (Chang et al., 2008). Lakshmans vorige project, Amazon Dynamo, was de tweede inspiratiebron voor Cassandra. Dynamo zorgde eerder al voor een hoge betrouwbaarheid bij een schaalbare database (DeCandia et al., 2007).

Hoewel Cassandra niet verder uitgebreid moest worden omdat het nog steeds voldeed aan de voorwaarden om het probleem van Facebook op te lossen, is Cassandra verder blijven groeien sinds 2008. Zo kan Cassandra nu bijvoorbeeld ook overweg met gestructureerde, semi-gestructureerde en ongestructureerde data (Kan, 2014).

Als men Cassandra binnen het CAP theorema moet gaan classificeren, kan men stellen dat de focus hier ligt op beschikbaarheid en partitie tolerantie. Binnen Cassandra kan er toch een keuze gemaakt worden tussen snelheid en consistentie. Hiermee kan een zeer hoge consistentie en ook een aanvaardbare snelheid verkregen worden (Ellis,

2009). Met het oorspronkelijk CAP theorema is dit echter niet mogelijk. Brewer (2012) herformuleerde echter het CAP theorema omdat hij vond dat het twee-uit-drie-concept misleidend was. Hier stelde hij dat een keuze tussen de drie mogelijkheden mogelijk was als er goed nagedacht werd over de partities.

1.3 Probleemstelling en Onderzoeksvragen

Cassandra belooft een groot aantal zaken en binnen deze bachelorproef is het de bedoeling om deze beloften na te gaan. Eerst en vooral zal de schaalbaarheid van Cassandra gecontroleerd worden. Is het werkelijk eenvoudig om de database op verschillende eenvoudige servers te installeren? Een tweede punt dat nagegaan wordt is de betrouwbaarheid van Cassandra. Is er werkelijk geen "single point of failure" en in hoeverre zijn back-ups nodig binnen deze NoSQL omgeving? Ook zal er stilgestaan worden bij de data modellering in Cassandra.

Hoofdstuk 2

Methodologie

Om een antwoord te bieden op alle onderzoeksvragen werd deze bachelorproef opgesplitst in twee luiken. Het eerste luik omvat het eerder theoretisch gedeelte, waar een literatuurstudie aan te pas kwam. Het tweede luik omvat het praktisch gedeelte die nodig was om op een aantal vragen een antwoord te bieden.

In het theoretisch gedeelte komt zoals eerder vermeld de literatuurstudie aan bod. Hierin wordt dieper ingegaan op:

- Hoe wordt de data fysiek opgeslagen binnen Cassandra?
- Hoe wordt de data logisch opgeslagen binnen Cassandra?
- Wat wordt er juist bedoeld met het meervoudig opslaan van data?
- Hoe belangrijk zijn back-ups binnen dit systeem?
- Voor welke problemen biedt Cassandra een oplossing?

Om dit alles na te kunnen gaan werd het praktisch gedeelte opgezet. Eerst moest er een Cassandra cluster opgezet worden. Dit gebeurde aan de hand van Vagrant virtuele machines. Er werd geopteerd voor Vagrant omdat dit een snelle manier is om verschillende identieke virtuele machines op te zetten. Ook kon aan de hand van één enkel script de volledige omgeving gecontroleerd worden. Voor de installatie van Cassandra werd eerst gekozen om met de Apache versie te werken. Het idee hiervan was om met de meest recente versie te werken. Om praktische redenen werd later gekozen om via het OpsCenter Community Edition van DataStax te werken. Deze tool maakte het mogelijk om via een webinterface de databank Cassandra te beheren en te observeren. Door gebruik te maken van deze opzet konden binnen de cluster snel nodes toegevoegd of verwijderd worden en via deze opzet kon de schaalbaarheid makkelijk getest worden.

Toen deze cluster opgezet was, werd de data die aangeboden werd door de Universiteit van Gent ingeladen in deze virtuele cluster. Voor deze data kan ingeladen worden, moest er eerst stilgestaan worden bij het datamodel van deze data. Deze data werd dus ook gebruikt om uit te leggen hoe je het best een datamodel opstelt binnen Cassandra. Hier werd eveneens kort stil gestaan bij het verschil tussen datamodellering binnen een relationele databank en Cassandra.

In een laatste deel moest ook nog de betrouwbaarheid van Cassandra getest worden. Hier werd doelbewust één van de virtuele machines (een van de nodes van de databank) uitgeschakeld om te zien hoe Cassandra hierop reageert. Doordat dit in een virtuele omgeving gebeurde is er geen risico op verlies van kritieke data.

Hoofdstuk 3

Architectuur

3.1 Partitionering

Een van de belangrijkste zaken bij Cassandra is het horizontaal schalen. Om dit te kunnen doen moet de data dynamisch gepartitioneerd worden over de nodes van een cluster. Cassandra krijgt dit voor elkaar door het gebruik van partitioners. De partitioner wordt ingesteld op cluster niveau. Deze partitioners maken gebruik van hash functies om te bepalen op welke node de data geplaatst moet worden. Bij consistente hashing bij Cassandra wordt de output behandeld als een "ring". Elke node in de cluster krijgt een willekeurige waarde toegewezen en deze waarde bepaalt dan de plaats in de ring (Lakshman and Malik, 2010).

Er zijn drie soorten partitioners in Cassandra:

1. **RandomPartitioner**: Dit was de standaard partitioner tot Cassandra 1.2. Aan de hand van de MD5 hash van de rij sleutel probeert deze partitioner de data evenwichtig over alle nodes te verspreiden. Omdat de MD5 Hash functie vrij traag is, werd er een andere partitioner als standaard aangesteld.
2. **Murmur3Partitioner**: Sinds Cassandra 1.2 is dit de standaard partitioner van deze databank. Deze partitioner maakt gebruik van de MurmurHash functie, waardoor de hash een 64-bit hashwaarde van de rij sleutel wordt.
3. **ByteOrderedPartitioner**: Zoals de naam doet vermoeden wordt deze partitioner gebruikt voor geordende partitionering. Deze partitioner ordent de rijen volgens de bytes van de rij sleutel. De tokens worden berekend aan de hand van de hexadecimale representatie van de leidende tekens van de rij sleutel. Op deze manier kan men, net zoals men met een cursor in een SQL omgeving zou doen,

de tabel geordend overlopen, gesorteerd op primaire sleutel.

3.1.1 Problemen met de ByteOrderedPartitioner

De ByteOrderedPartitioner blijkt een oplossing te bieden om tabellen te verkrijgen die gesorteerd zijn op de primaire sleutel. Ondanks dit feit zijn er toch een aantal problemen met deze partitioner:

- Sequentiële writes kunnen voor "hot spots" zorgen. Als een aantal rijen ongeveer gelijktijdig toegevoegd of geüpdatet worden, dan worden deze quasi zeker op dezelfde node weggeschreven. Dit vormt bijvoorbeeld een groot probleem bij tijdreeksen (Kan, 2014).
- Met de ByteOrderedPartitioner is het zeer moeilijk om een gebalanceerde cluster te krijgen. De enige manier waarop dit verkregen kan worden is om dit manueel te doen. Als men dit echter niet doet is de kans vrij groot dat de data op slechts enkele nodes van de cluster opgeslagen wordt. Als men dan nog eens verschillende types voor de primaire sleutels definieert is het zo goed als onmogelijk om een gebalanceerde cluster te krijgen (Bauer, 2013).

3.2 Replicatie

Om de hoge beschikbaarheid te verkrijgen maakt Cassandra gebruik van data replicatie. Door de replicatie wordt de kans op een "Single point of Failure" enorm beperkt. Cassandra biedt een methode aan om de replicatie factor te blijven behouden zelfs indien bepaalde nodes uitgevallen zouden zijn. Deze replicatie factor wordt ingesteld op keyspace niveau. De replicatie factor bepaalt hoeveel replica's van de data bijgehouden worden. Op basis van de partitioner en de replicatie strategie wordt bepaald waar de replica's opgeslagen worden. Om deze replicatie in goede banen te leiden zijn er een aantal strategieën uitgewerkt (Kan, 2014).

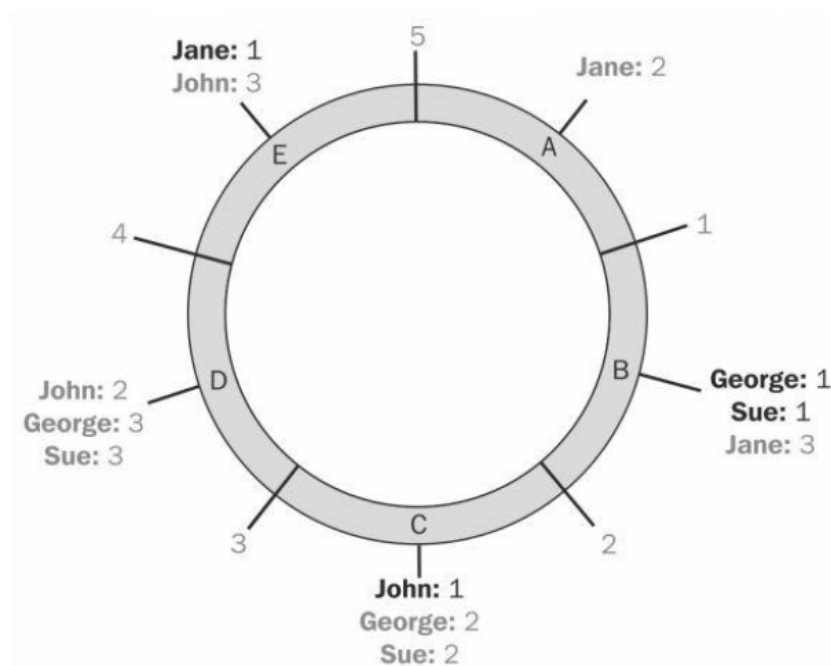
3.2.1 Replicatie strategieën

De replicatie strategie bepaalt waar de data precies op de cluster geplaatst wordt. Hierbij zijn er twee strategieën:

SimpleStrategy

Deze SimpleStrategy wordt meestal gebruikt als er maar één datacenter aanwezig is. Wanneer men over meerdere datacenters beschikt is het beter om naar de andere strategie te kijken. Wanneer de standaard partitioner, de Murmer3Partitioner, gebruikt wordt gaat de partitioner de locatie bepalen aan de hand van de hash van de primaire sleutel. De replica's van de data worden geplaatst op de volgende nodes in de ring met de richting van de klok mee.

Op figuur 3.1 wordt een voorbeeld gegeven waarbij de replicatie factor 3 is. De zwarte namen zijn de originele data en de grijze de replica's.



Figuur 3.1: Illustratie SimpleStrategy (Strickland, 2014)

NetworkTopologyStrategy

Op productieclusters wordt vaak de NetworkTopologyStrategy toegepast. Hierbij gaat men ervan uit dat een productiecluster uit meerdere datacenters bestaat of in de toekomst meerdere datacenters zal bevatten.

Het grote voordeel van deze strategie is dat er rack awareness is en dat de snitches ingesteld kunnen worden. Rack awareness zorgt ervoor dat de replica's op verschillende racks worden opgeslagen. Dit is echter niet mogelijk bij SimpleStrategy. Ook kan men

door het gebruik van deze strategie voor ieder datacenter een andere replicatie factor instellen.

3.3 Snitches

Een snitch is verantwoordelijk om de data snel en efficiënt op te halen binnen de cluster. Een andere verantwoordelijkheid van een snitch is om mee te helpen bepalen waar de replica's fysiek opgeslagen worden. Er zijn acht soorten snitches beschikbaar op Cassandra (Strickland, 2014).

1. **SimpleSnitch:** Deze snitch gaat altijd samen met SimpleStrategy en is speciaal ontworpen voor eenvoudige datacenters.
2. **RackInferringSnitch:** Deze snitch probeert de netwerk topologie van de cluster te achterhalen. Het gebruik van deze snitch wordt echter sterk afgeraden. Het probleem is hier dat ervan uitgegaan wordt dat het ip-adres de structuur van het datacenters en de racks reflecteert.
3. **PropertyFileSnitch:** Met deze snitch kan de administrator zelf instellen welke nodes tot welke rack en datacenter behoren. Als men deze snitch gebruikt moet men dus de ganse topologie uitschrijven. Iedere node moet ook precies op dezelfde manier geconfigureerd worden. Dit alles heeft veel overhead als er nodes toegevoegd of verwijderd moeten worden.
4. **GossipingPropertyFileSnitch:** De GossipingPropertyFileSnitch lijkt zeer sterk op de PropertyFileSnitch. Het grote verschil is dat men per rack en datacenter maar één node hoeft te configureren. Daarna verspreidt de snitch de configuratie van deze node naar alle andere nodes.
5. **CloudStackSnitch:** Deze snitch de datacenters en racks op volgens CloudStack's land, locatie en beschikbaarheid zones.
6. **GoogleCloudSnitch:** De GoogleCloudSnitch is speciaal gemaakt voor Cassandra binnen Google Cloud. Hierbij zet deze snitch de locatie als datacenter en de beschikbaarheidszone als rack.
7. **EC2Snitch:** Deze snitch is zeer gelijkaardig aan de GoogleCloudSnitch. Net zoals bij de vorige snitch wordt de regio als datacenter ingesteld en de beschikbaarheidszone als rack.
8. **EC2MultiRegionSnitch:** Deze snitch werkt op identiek dezelfde manier als de EC2Snitch. Het enige verschil hier is dat deze snitch publieke ip-adressen toestaat voor communicatie tussen verschillende datacenters.

Voor productieomgevingen is de `GossipingPropertyFileSnitch` meestal de beste keuze wanneer er met fysieke datacenters wordt gewerkt. Als men in de cloud werkt kiest men best voor de bijhorende snitch.

3.4 Seed node

Sommige nodes in de Cassandra cluster zijn seed nodes. De configuratie van deze seed node wordt gebruikt bij het bootstrappingproces wanneer er nieuwe nodes aan de cluster toegevoegd worden. De nieuwe node zal eerst communiceren met de seed node om zo informatie over de andere nodes in de cluster te bekomen. Hoewel er technisch gezien maar één seed node nodig is per cluster, is het toch sterk aangeraden om meerdere seed nodes per datacenter beschikbaar te stellen (Kan, 2014).

Hoofdstuk 4

Data structuren in Cassandra

Bij Cassandra is er redelijk wat onduidelijkheid over de terminologie. Dit komt omdat er twee implementaties, namelijk CQL en opslag, zijn. Binnen deze twee implementaties worden vaak termen hergebruikt (Ahmed, 2015). De structuren die hieronder besproken worden zijn allemaal onderdeel van de CQL implementatie.

4.1 Column

Binnen Cassandra is een kolom de kleinste data structuur die er bestaat. De kolom van een relationele database kan niet vergeleken worden met een kolom in Cassandra. Hun functie is volledig anders.

Een kolom in Cassandra bevat drie of vier elementen:

1. De naam van de kolom
2. De waarde van de kolom
3. Een timestamp
4. Een time-to-live veld

Voor de gebruiker zijn hier de naam van de kolom en de waarde van het grootste belang. Met deze velden gaat er uiteindelijk gewerkt worden. De naam en waarde worden opgeslagen als byte arrays. Toch voorziet Cassandra een resem built-in types. De timestamp is er om conflicten op te lossen. De time-to-live is een optioneel veld. Hiermee kan aangegeven worden hoe lang de kolom opgeslagen moet worden.

4.2 Row

Een rij kan beschouwd worden als een geordende lijst van kolommen. Elke rij heeft een unieke sleutel. Bij Cassandra is het zo dat als één kolom van een rij op een node staat, alle andere rijen ook op deze node staan.

A.d.h.v. figuur 4 kan verklaard worden waarom Cassandra tot de Column Family Store behoort terwijl er sprake is van rijen. De kolommen worden in Cassandra namelijk in rijen opgeslagen.

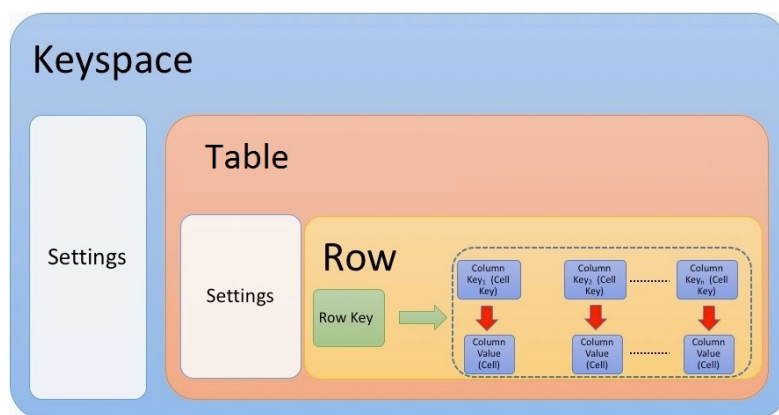
4.3 Table

Een niveau hoger dan de rij vindt men de tabel terug. Binnen Cassandra is een tabel eigenlijk beter te beschrijven als een column family. Dit was ook in de eerste versies van Cassandra het geval.

In een tabel vind men alle rijen terug die hetzelfde schema hebben. Toch kan Cassandra als een schemaloze database beschouwd worden doordat de kolommen zelf geen schema hoeven te hebben. Door het feit dat men hier alle rijen groepeerd is het belangrijk dat de rijen een unieke sleutel hebben (Hewitt, 2010).

4.4 Keyspace

Een keyspace kan vergeleken worden met een database of schema in relationele databases. De keyspace bevat de replicatie strategie en replicatie factor.



Figuur 4.1: Logische datastructuren in Cassandra

Hoofdstuk 5

Opzetten van de Cassandra cluster

5.1 Apache Cassandra

Om de cluster op te zetten werd geopteerd om gebruik te maken van virtuele machines, die geconfigureerd werden met Vagrant.

In een eerste poging om een werkende Cassandra cluster te bekomen werd op elke Vagrant machine Cassandra 3.3, op het moment van dit onderzoek de meest recente versie, geïnstalleerd. Nadat dit gebeurd was, dienden nog enkele stappen voltooid te worden om een werkende cluster te bekomen (DataStax, 2016b). Deze configuratie gaf echter veel problemen. Cassandra werd na enkele bewerkingen telkens onbruikbaar en gaf de volgende foutmelding weer: "could not access pidfile for Cassandra". Een eerste oplossing voor dit probleem was om ervoor te zorgen dat de user 'cassandra' toegang had tot de pidfile, wat namelijk niet het geval was doordat de installatie van Cassandra werd uitgevoerd door de Vagrant setup. Maar ook dit leverde weinig resultaat op. Wel moet opgemerkt worden dat de installatie van Cassandra zelf geen problemen met zich meebracht. Voor de aanpassing van de configuratie file van Cassandra werkte deze perfect op iedere node.

5.2 DataStax OpsCenter

Uiteindelijk werd er geopteerd om gebruik te maken van OpsCenter omdat dit een gemakkelijke manier is om snel een Cassandra cluster te bekomen en omdat er ook goede mogelijkheden tot observeren van de database aanwezig zijn. Er werd voor

OpsCenter community edition 5.2.4 gekozen. Bij deze versie diende Cassandra 2.1.11 gebruikt te worden (Cantoni, 2016).

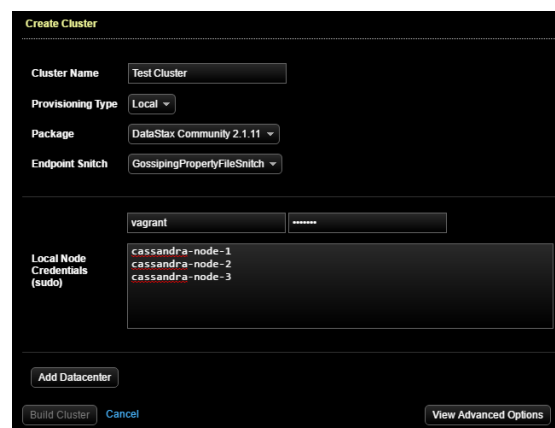
De setup bestaat uit één master node waarop het OpsCenter runt en verder uit drie slave nodes waarop de uiteindelijke Cassandra database komt te runnen. Na de NAT router van Oracle Virtual Box werd een privaat netwerk opgezet opdat deze machines met elkaar zouden kunnen communiceren. Hiervoor moest iedere machine een uniek ip-adres krijgen binnen het netwerk en moest de file `/etc/hosts` op iedere node aangepast worden. De scripts hiervoor kunnen in bijlage A teruggevonden worden.

Eenmaal de virtuele machines correct geconfigureerd waren, kon overgegaan worden tot de eigenlijke installatie van Cassandra. Zoals eerder vermeld werd, werd hiervoor gebruik gemaakt van het OpsCenter. Daarom werd op de master node naar 'localhost:8888' gesurft om de installatie te kunnen starten. Op de pagina werden verschillende opties aangeboden en hier werd voor de optie 'brand new cluster' gekozen.

In het volgende venster wordt er om verschillende zaken gevraagd. Tabel 5.1 en figuur 5.1 geven weer hoe dit venster werd ingevuld.

Property Name	Waarde
Cluster Name	BP Cluster
Type	local
Package	datastax community 2.1.11
Endpoint Snitch	GossipingPropertyFileSnitch
Username en password	vagrant/vagrant
Local Node Credentials	cassandra-node-1, cassandra-node-2, cassandra-node-3

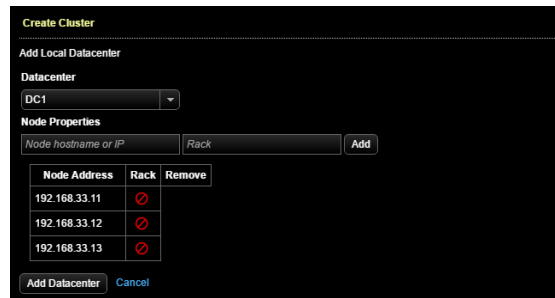
Tabel 5.1: Configuratie van de Cassandra Cluster



The screenshot shows the 'Create Cluster' window in the OpsCenter application. It contains several input fields and dropdown menus, all of which have been populated with specific values. The 'Cluster Name' field contains 'Test Cluster'. The 'Provisioning Type' dropdown is set to 'Local'. The 'Package' dropdown is set to 'DataStax Community 2.1.11'. The 'Endpoint Snitch' dropdown is set to 'GossipingPropertyFileSnitch'. Below these, there are fields for 'Username' (vagrant) and 'Password' (vagrant). A text area for 'Local Node Credentials' contains the list 'cassandra-node-1', 'cassandra-node-2', and 'cassandra-node-3'. At the bottom of the window, there are four buttons: 'Add Datacenter', 'Build Cluster', 'Cancel', and 'View Advanced Options'.

Figuur 5.1: Cassandra: Instellingen deel 1

Hier moest een datacenter toegevoegd worden. Hierbij is er een vrije keuze voor de naam van het datacenter en zijn de node properties het ip-adres van de slave nodes (Figuur: 5.2).

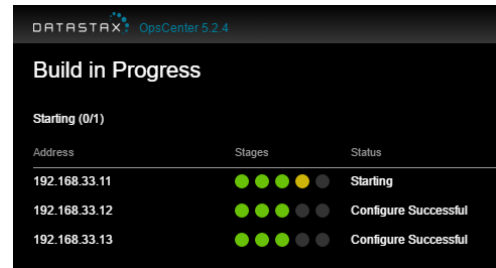


Node Address	Rack	Remove
192.168.33.11		X
192.168.33.12		X
192.168.33.13		X

Figuur 5.2: Cassandra: Instellingen deel 2

Eenmaal de datacenters toegevoegd zijn, kan er verder gegaan worden. Bij het drukken op de knop 'build cluster' wordt verder nog gevraagd om de fingerprints van de nodes te accepteren.

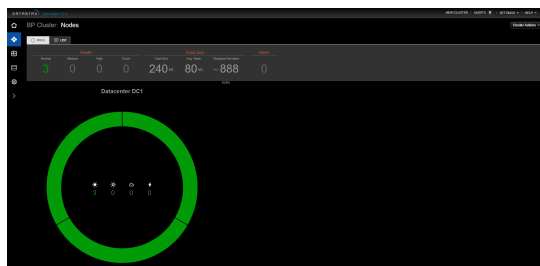
Hierna begint OpsCenter met de installatie van de Cassandra cluster (Figuur: 5.3). Deze installatie nam enige ogenblikken in beslag. De eerste keer kwam er een fout voor omdat er niet genoeg werkgeheugen was toegekend aan de slave nodes. In de setup die hier gebruikt werd, werd het minimum aanvaarde geheugen, nl 2GB, gegeven aan de slave nodes. Samen met Cassandra wordt ook de DataStax agent meegeïnstalleerd opdat het OpsCenter zou kunnen communiceren met iedere node.



(b) Deel 2



Na de installatie komt men terecht in het Dashboard van OpsCenter (Figuur 5.4a). Hierin worden een aantal zaken weergegeven zoals: de gezondheid van de cluster, het aantal write requests, de write request latency... Hier kunnen nog meer grafieken aan toegevoegd worden door de knop 'add graph' te gebruiken. In het tabblad 'nodes' kan de gezondheid van de cluster bekeken worden net zoals gezien kan worden hoe de data verdeeld zit over de cluster. Deze informatie kan in ringvorm, zoals in figuur 5.4b weergegeven wordt, of in een lijst weergegeven worden.



(b) Tabblad nodes: ring

Cassandra biedt zelf ook een tool aan die deze monitoring uitvoert: 'nodetool'. Om te bekijken of de installatie goed gelukt is, kan men via ssh inloggen op een van de nodes van de cluster en hier 'nodetool status' laten lopen (Figuur 5.5). Als men met nodetool

de status opvraagt, krijgt men eveneens alle nodes in de cluster te zien, samen met de hoeveelheid data die ze bevatten en hoeveel procent deze data voorstelt van alle data die op de cluster aanwezig is.



```
vagrant@cassandra-node-1:~$ nodetool status
Datacenter: DC1

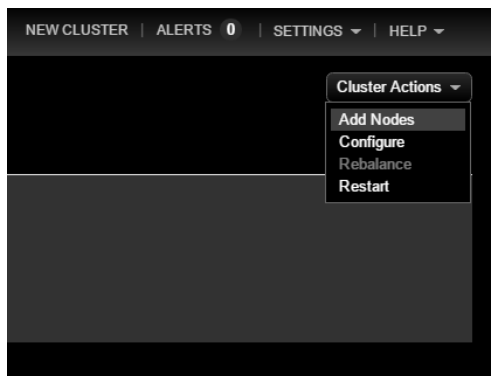
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address      Load          Tokens       Owns    Host ID
--  --
UN 192.168.33.11  80.5 KB      256         ?       74a74814-d29b-4091-b621-89599cb4b1f9
UN 192.168.33.12  80.97 KB     256         ?       0167d105-d0ca-4994-9550-0fc089c6a4db
UN 192.168.33.13  80.38 KB     256         ?       279f3eae-832f-4a1e-9d70-d68512ea39ff

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
vagrant@cassandra-node-1:~$
```

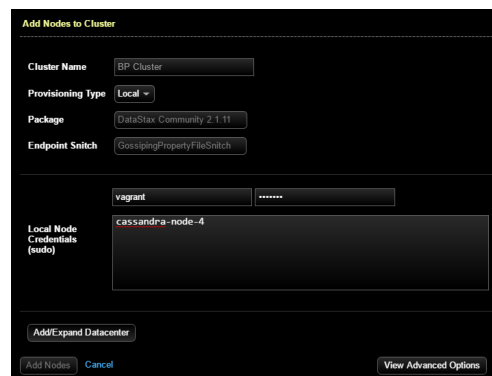
Figuur 5.5: nodetool

5.3 Toevoegen van een node

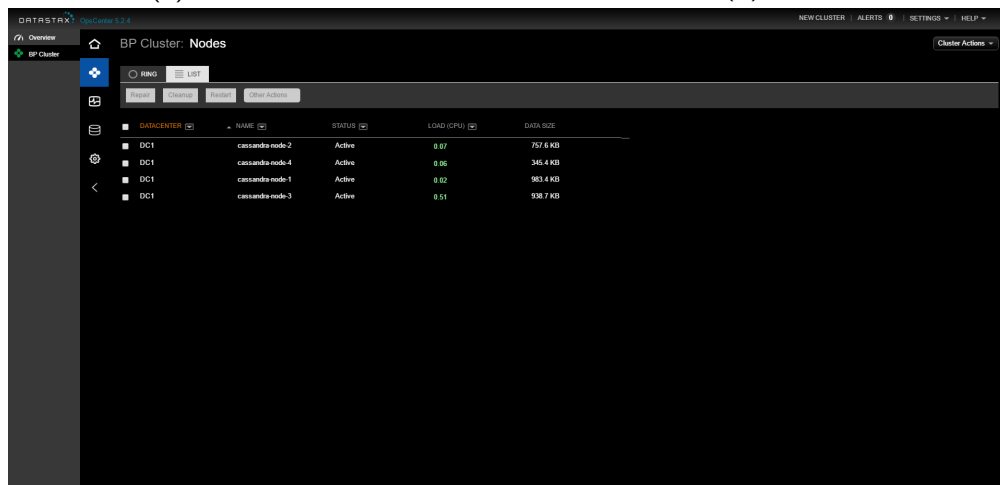
Het proces voor de toevoeging van een node is quasi gelijk aan het opzetten van de cluster. Hierbij dient men in het OpsCenter bij het menu 'cluster actions' voor de optie 'add node' te kiezen (Figuur 5.6a). Hier wordt een vierde node toegevoegd aan de cluster. Net zoals bij de installatie van de cluster wordt opnieuw het scherm getoond waarin de aanmeldgegevens voor de gegeven node gevraagd worden (Figuur 5.6b). Via de knop 'Add/Expand Datacenter' kan men op dezelfde manier nodes toevoegen aan het bestaande datacenter die tijdens de installatie werd aangemaakt. Bij het bevestigen wordt opnieuw gevraagd om de fingerprint van de node te accepteren en hierna begint de installatie van de software op de node. Waardoor de agent en Cassandra geïnstalleerd en geconfigureerd worden. Na dit alles kan men in het tabblad 'nodes', nu in een lijstvorm, zien dat de nieuwe node is toegevoegd aan de cluster (Figuur: 5.6c).



(a) Add node



(b) Deel 2

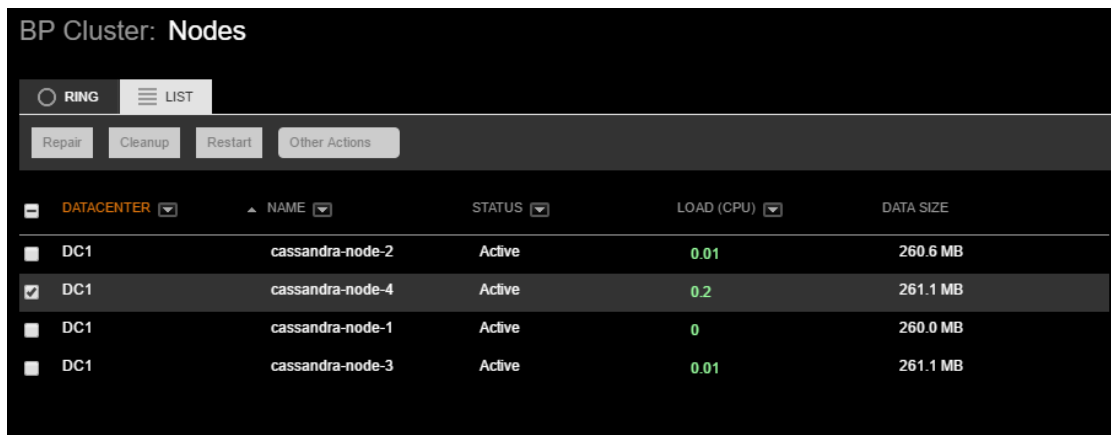


(c) Tabblad node met 4 nodes

Figuur 5.6: Toevoegen van een node via OpsCenter

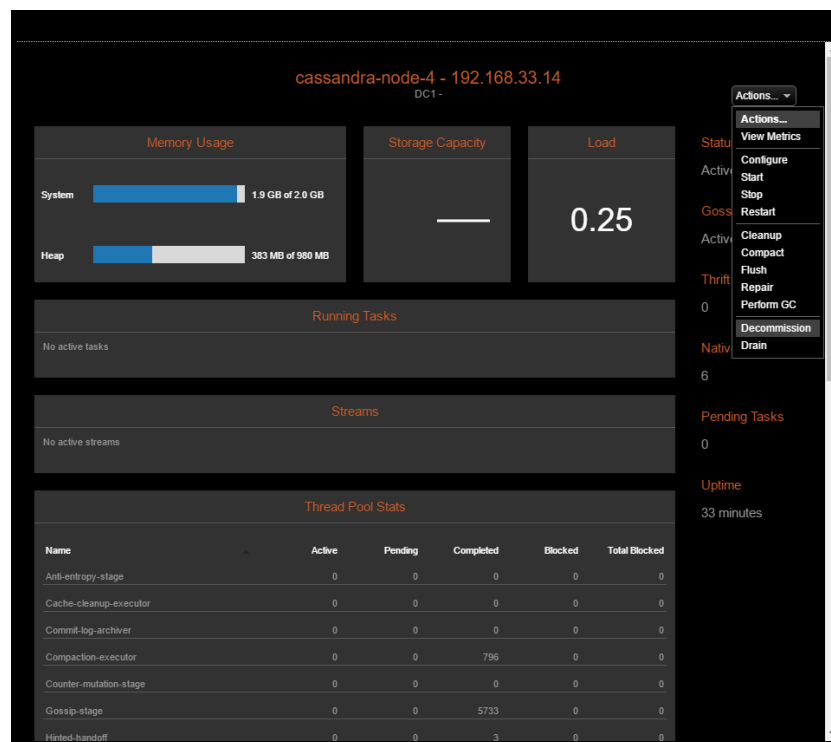
5.4 Verwijderen van een node

Het verwijderen van een node van een Cassandra cluster is net zoals het toevoegen van een node zeer eenvoudig. Binnen het OpsCenter gaat men naar het tabblad nodes en hier selecteert men de node die men wil verwijderen (figuur 5.7).



Figuur 5.7: Verwijderen van een node: Selecteer de node

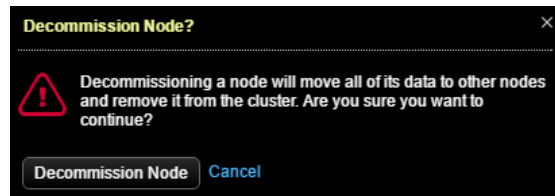
Daarna komt men op een volgend scherm terecht (figuur 5.8). Hier kiest men bij de acties voor "Decommision".



Figuur 5.8: Verwijderen van een node: Selecteer de node

Voor de node effectief verwijderd wordt, wordt er nog om bevestiging gevraagd. Hier

wordt ook nog meegedeeld dat alle data van de te verwijderen node overgeplaatst zal worden op de andere nodes (figuur 5.9)



Figuur 5.9: Verwijderen van een node: Selecteer de node

Hoofdstuk 6

Datamodellering in Cassandra

6.1 Primaire sleutel

Binnen de primaire sleutel in Cassandra worden de partitie kolommen (6.2) en clustering kolommen (6.3) vastgelegd voor een tabel. In tegenstelling tot relationele databases wordt de primaire sleutel hier niet gebruikt als unieke sleutel voor de rij, maar om snel te weten te komen waar de data zich binnen de cluster bevindt (Kan, 2014). Als een tabel slechts één kolom heeft als primaire sleutel, dan is deze kolom meteen ook de partitie kolom en zijn er geen clustering kolommen.

Door het feit dat de primaire sleutel in Cassandra niet uniek hoeft te zijn, gedragen het INSERT commando en het UPDATE commando zich op een identieke wijze. Een nadeel hiervan is dat er geen waarschuwing weergegeven wordt als een rij overschreven zou worden. Omdat dit veelal ongewenst gedrag is binnen applicaties komt men al snel terecht bij een samengestelde primaire sleutel, een primaire sleutel uit verschillende kolommen.

6.2 Partitie kolom

Het eerste deel van de primaire sleutel bestaat uit de partitie kolommen. Het doel van deze partitie kolommen is om de data gebalanceerd over alle nodes te spreiden. Op deze manier kan de fysieke locatie van de data ook snel achterhaald worden (Kan, 2014).

Bij het kiezen van de partitie kolommen dient men met een aantal zaken rekening te houden om de data evenwichtig over alle nodes te spreiden, maar er is slechts

één fysieke restrictie. Iedere rij kan slechts 2 miljard kolommen bevatten (McFadin, 2013). In de meeste gevallen is dit ruim voldoende. Tijdsreeksen vormen hier een uitzondering op. Deze reeksen kunnen al gauw miljarden entries bevatten. Hier is het dus zeer belangrijk om goede partitie kolommen te kiezen of men komt hier al snel in de problemen.

6.3 Clustering kolom

Het laatste deel van de primaire sleutel bestaat uit de clustering kolommen. Deze bepalen de volgorde van de data op de fysieke media (Strickland, 2014). De clustering kolommen hebben echter geen invloed op de fysieke locatie van de data binnen de cluster. Toch zullen ze een belangrijke invloed hebben op welke query's uitgevoerd kunnen worden.

6.4 De WHERE clause

Zoals Lerer (2015) aanhaalt is één van de grootste verschillen tussen CQL en SQL de WHERE clause. In SQL kent de WHERE clause geen restricties. Bij CQL is dit anders. Hier worden restricties opgelegd door de partitie en clustering kolommen. Eveneens kan er enkel op de partitie en clustering kolommen gefilterd worden.

6.4.1 Restricties opgelegd door de partitie kolommen

Een eerste restrictie die wordt opgelegd door de partitie kolommen is dat ofwel alle partitie kolommen worden opgenomen in de WHERE clause ofwel geen enkel. Dit is nodig omdat Cassandra anders de hash niet kan berekenen. Deze hash is echter nodig om te weten op welke node de data zich bevindt.

Een volgende restrictie die wordt opgelegd door de partitie kolommen is het feit dat enkel de '=' en IN operator rechtstreeks gebruikt kunnen worden. Tot Cassandra 2.2 kon de IN operator enkel op de laatste gedefinieerde partitie kolom toegepast worden.

De laatste restrictie die wordt opgelegd door de partitie kolommen heeft te maken met de >, >=, < en <= operators. Deze zijn enkel rechtstreeks toepasbaar als de partitioner ingesteld is op ByteOrderedPartitioner. Indien dit niet het geval is moet men een omweg maken via de token functie. Op het eerste zicht lijkt het gebruik van de token functie minder efficiënt voor het selecteren van data, maar dit weegt niet op

tegen de nadelen die ByteOrderedPartitioner heeft op het gebied van de distributie van de data, zoals eerder vermeld werd. Wel dient hiermee opgepast te worden omdat de operators dan uitgevoerd worden op de tokens en niet rechtstreeks op de data. Dit levert niet altijd het gewenste resultaat op.

6.4.2 Restricties opgelegd door de clustering kolommen

Voor de restricties op de clustering kolommen uitgelegd kunnen worden, moet eerst uitgelegd worden hoe de data in Cassandra binnen een partitie opgeslagen wordt. Dit zal nu aan de hand van een voorbeeld uitgelegd worden (Lerer, 2015). Neem de onderstaande tabel:

```
CREATE TABLE NumberOfTwitterMessages (  
    userid bigint, date text, hour int, minute int,  
    nrOfTweets int,  
    PRIMARY KEY ((userid, date), hour, minute)  
);
```

Hier wordt de data nu als volgt opgeslagen binnen de partitie:

```
{hour: 20  
  {minute: 4 {nrOfTweets: 6}}  
  {minute: 7 {nrOfTweets: 1}}  
  {minute: 21 {nrOfTweets: 16}}  
  ...  
}
```

De eerste restrictie wordt hier opgelegd door de manier waarop de data opgeslagen is. Als men een voorwaarde wil vastleggen voor een clustering kolom, dan dienen de clustering kolommen die voor deze komen in de primaire sleutel ook vastgelegd te worden. Dit is nodig omdat Cassandra de data anders niet efficiënt kan terugvinden.

Tot Cassandra 2.2 was de IN operator enkel toegelaten op de laatste clustering kolom. Sinds Cassandra 2.2 is deze restrictie vervallen en kan men zelfs multi-kolom IN restricties opleggen.

De >, >=, < en <= operators zijn ook enkel toegestaan op de laatste clustering kolom waar een restrictie aan opgelegd is. Toch is hier een oplossing voor aangezien men deze operators kan toepassen over meerdere kolommen. Bij deze multi-column slices is het echter wel belangrijk dat de restrictie van de WHERE clause met dezelfde kolom start.

6.5 Doelen bij datamodellering in Cassandra

Zoals in bovenstaande sectie duidelijk werd, dient er bij Cassandra met heel wat rekening gehouden te worden als men een datamodel creëert. Hobbs (2015) definieerde wat de doelen zijn bij het opstellen van een datamodel in Cassandra. Zoals door Hobbs aangegeven wordt, lijkt de querytaal van Cassandra CQL sterk op SQL, maar kan het gebruik ervan zeer verschillend zijn.

Een eerste doel bij het opstellen van een datamodel in Cassandra is om de data evenwichtig over alle nodes te verspreiden. Dit kan bekomen worden door de partitie kolommen goed te kiezen. Zoals eerder vermeld is, worden de partitie kolommen bepaald door het eerste deel van de primaire sleutel.

Een tweede doel is om zo weinig mogelijk partitie reads te moeten doen. Doordat iedere partitie op een andere node kan staan is dit belangrijk. Als de partities effectief op verschillende nodes staan moeten de afzonderlijke commando's naar elke node apart verstuurd worden en dit zorgt voor overhead. Het is zelfs zo dat als de data op dezelfde node staat, de partitie reads nog steeds inefficiënt zijn. Dit komt door de manier waarop Cassandra de rijen opslaat.

Zaken die bij relationele databanken belangrijk zijn zoals het aantal writes en data duplicatie minimaliseren, zijn binnen Cassandra geen doelen. In Cassandra zijn write operaties goedkoop omdat Cassandra hiervoor geoptimaliseerd is. Denormalisatie en duplicatie van data zijn ook zeer normaal binnen Cassandra. Dit komt door de architectuur van Cassandra. Hierbij gaat men ervan uit dat schijfruimte goedkoop is in vergelijking met andere resources zoals CPU, geheugen, netwerk... Verder heeft Cassandra geen JOIN waardoor het ook hoogst inefficiënt en onpraktisch zou zijn om geen duplicate data te hebben.

6.6 Datamodel opstellen voor Cassandra

Hobbs (2015) definieerde niet enkel de doelen van een datamodel in Cassandra, maar haalt ook aan hoe deze bekomen kunnen worden. Voor men begint aan het opstellen van een datamodel moet al nagedacht worden over de query's die ondersteund moeten worden. Dit staat in schril contrast met relationele databanken waar het datamodel wordt bepaald door de objecten en hun relaties.

Door na te denken over de te ondersteunen query's kan het aantal partitie reads al drastisch verminderd worden. Ook dient men rekening te houden met de restricties die de partitie en clustering kolommen opleggen aan de WHERE clause.

Voor iedere query zou er slechts één partitie read uitgevoerd mogen worden. Hiervoor is het belangrijk dat de tabellen geoptimaliseerd zijn voor de reads die uitgevoerd gaan worden.

Een goed voorbeeld waarin alle zaken meteen duidelijk worden, wordt ook gegeven door Hobbs (2015) . In dit voorbeeld is het de bedoeling om users op het halen volgens hun email of username. De oplossing voor Cassandra zijn de volgende twee tabellen:

```
CREATE TABLE users_by_username (
  username text PRIMARY KEY,
  email text ,
  age int
)
```

```
CREATE TABLE users_by_email (
  email text PRIMARY KEY,
  username text ,
  age int
)
```

Bij dit datamodel krijgt iedere user zijn eigen partitie. Cassandra kan zo de data evenwichtig verdelen over de nodes. Om een user op te zoeken via een email of username dient slechts één partitie gelezen te worden.

Stel dat men nu zou proberen om redundante data te verminderen, wat in Cassandra geen doel mag zijn, met de volgende tabellen:

```
CREATE TABLE users (
  id uuid PRIMARY KEY,
  username text ,
  email text ,
  age int
)
```

```
CREATE TABLE users_by_username (
  username text PRIMARY KEY,
  id uuid
)
```

```
CREATE TABLE users_by_email (
  email text PRIMARY KEY,
  id uuid
)
```


De data zal met deze tabellen nog steeds evenwichtig gedistribueerd zijn over de nodes, maar nu moet men meer dan één partitie read doen. Dus door een niet-doel proberen te bereiken is hier een belangrijk doel verloren gegaan.

Hoofdstuk 7

Data importeren in Cassandra

7.1 Importeren via cqlsh

Om de data via cqlsh te kunnen importeren dient eerst een keyspace aangemaakt te worden. Bij het aanmaken van deze keyspace dient de replicatie strategie en de replicatie factor meegegeven te worden. Na het aanmaken van de keyspace, dient hierin de tabel waarin de data geïmporteerd zal worden aangemaakt te worden.

Nu kunnen we via het 'COPY' commando, dat binnen cql voorzien is om data te importeren in Cassandra (Cannon, 2012). Een aantal zaken dienen hierbij opgemerkt te worden.

1. De volgorde van de kolommen kan gespecificeerd worden aangezien Cassandra de kolommen automatisch alfabetisch sorteert en dit niet noodzakelijk het geval is bij het csv bestand.
2. Het scheidingsteken van de velden in het csv bestand kan gespecificeerd worden evenals de encapsulering van de velden.
3. Er kan specifiek meegegeven worden wat Cassandra moet aanvangen met de null waarde.

Dit is een zeer eenvoudige manier om data te importeren in Cassandra. Alle inserts gebeuren asynchroon, wat uiteraard goed is. Toch wordt deze methode niet aangeraden om te gebruiken bij het importeren van grote hoeveelheden data. Een eerste reden is dat er maar met één node connectie gemaakt wordt. Hierdoor wordt de werklust niet evenwichtig verdeeld over alle nodes. Bij ca. 1 miljoen rijen, afhankelijk van het aantal kolommen, kan het zijn dat deze methode vast loopt.

De bovenstaande problemen kan men op een aantal manieren oplossen. De drie meest voorkomende zijn:

1. Het opsplitsen van één groot csv bestand in verschillende kleinere bestanden.
2. Het gebruik van de sstableloader die Cassandra voorziet.
3. Het gebruik van cassandra-loader

7.2 Importeren via sstableloader

Het importeren van data via sstableloader is eveneens een eenvoudig proces, maar hier komt wat meer werk aan te pas.

Enkele belangrijke nadelen van deze manier zijn:

- Men moet een aangepaste applicatie schrijven om dit te kunnen gebruiken.
- Om sstableloader te kunnen gebruiken dienen alle nodes van de cluster online te zijn.
- De sstable dient aangemaakt te zijn vooraleer men deze methode kan gebruiken.

De reden waarom alle nodes online moeten zijn is omdat de hash van het record meteen bepaald wordt en dan doorgestuurd wordt naar de node waar het hoort te staan.

7.3 Importeren via cassandra-loader

Dit is een Java programma van Brain Hess. Dit programma maakt gebruik van de CQL driver die beschikbaar gesteld wordt door DataStax. Het ganse principe van dit programma is om asynchroon te werken en om met iedere node in de cluster te verbinden om zo het werk evenwichtig te verdelen. Dit laatste was een van de struikelblokken bij het importeren van data via cqlsh.

7.4 Testen van de verschillende loaders

In dit stuk is het de bedoeling om de verschillende manieren van data import te vergelijken. In deze vergelijking werd altijd dezelfde data gebruikt. Enkel de hoeveelheid data die ingeladen werd was verschillend.

Eerst was de meest eenvoudige manier aan de beurt, het COPY commando via cqlsh. Hier werden verschillende bestanden met dezelfde data, maar met een ander aantal rijen ingeladen.

Aantal rijen	Tijd (s)
1 000	0.388
10 000	2.068
100 000	23.635
1 000 000	195.742
10 000 000	2 150.821

Tabel 7.1: Importeren van data met cqlsh

Als men tabel 7.1 bekijkt ziet men hier dat er een lineair verband is tussen het aantal rijen en de tijd die nodig is om alle rijen te importeren. Dit is ook wat verwacht werd.

Het testen met sstableloader werd niet gedaan omwille van de reden dat alle nodes online moeten zijn. In een systeem waar alles ingezet wordt op een hoge beschikbaarheid, is het dan ook niet aangeraden om een methode te gebruiken waar import crasht als er een node offline gaat.

Het gebruik van cassandra-loader leverde veel problemen op. Dit kwam deels door het gebruik van de virtuele machines. De cassandra-loader verwacht 8 GB werkgeheugen. Dit was oorspronkelijk niet beschikbaar op de virtuele machine. Hierdoor moest het originele script aangepast worden. Echter na de aanpassing van het virtueel geheugen dat toegewezen wordt aan de Java virtuele machine crashte het programma.

Doordat het inladen via cassandra-loader niet werkte kan er geen vergelijking gemaakt worden. Toch moet er opgemerkt worden dat de manier van cassandra-loader voor het importeren en het importeren via cqlsh niet zo veel verschilt in theorie. Cassandra-loader gaat er enkel nog voor zorgen dat het werk over alle online nodes evenwichtig verdeeld wordt.

Hoofdstuk 8

Gedrag bij uitvallen van een node

8.1 Gossip en foutdetectie

De nodes van Cassandra communiceren ongeveer iedere seconde met elkaar om informatie over hun eigen status en over de status van andere nodes waarmee ze in contact staan. Hiervoor maakt Cassandra gebruik van een peer-to-peer protocol. Op deze manier kunnen nodes snel de status van andere nodes weten. Deze informatie wordt ook lokaal opgeslagen.

Cassandra gebruikt het "Phi Accrual Failure Detection Algorithm" voor het detecteren van het uitvallen van nodes (Kan, 2014). Het idee bij dit algoritme is dat de status niet weergegeven wordt door een booleaanse waarde, dood of levend, maar door een waarde die aangeeft hoe groot de kans is dat een node dood of levend is. Als een node op deze manier dood verklaard wordt, blijven de andere nodes toch nog communiceren met deze node om ze te bepalen wanneer deze terug levend is.

8.2 Herstelmechanisme

Cassandra voorziet drie built-in herstelmechanismes:

1. Hinted handoff
2. Anti-entropy node repair
3. Read repair

8.2.1 Hinted handoff

Het doel van hinted handoff is om de tijd die nodig is om node te herstellen na het uitvallen, te minimaliseren. Hierbij wordt er wat lees consistentie opgeofferd om de schrijf beschikbaarheid te verhogen.

In het geval een node offline is houdt een andere gezonde node een hint bij als er data geschreven wordt. Wanneer in het slechtste geval alle nodes offline zijn, houdt de coördinator deze hint bij. Als de node dan online komt wordt eerst gezorgd dat deze writes uitgevoerd worden en kan in de tijd die hiervoor nodig is geen reads uitgevoerd worden. Als in deze periode toch reads uitgevoerd zouden worden kan dit leiden tot inconsistente reads.

Standaard houdt Cassandra deze hints drie uur bij. Dit limiet is om te voorkomen dat deze hint wachtlijst niet te lang wordt. Na dit limiet is het nodig om handmatig een herstel te doen om de consistentie te garanderen (Strickland, 2014).

8.2.2 Anti-entropy repair

Dit algoritme staat in voor de synchronisatie van replica's en zorgt dat deze allen up-to-date zijn op alle nodes. Dit proces gebeurt asynchroon. Binnen anti-entropy repair word gebruik gemaakt van de manuele read repair (8.2.3) (Strickland, 2014).

8.2.3 Read repair

Deze repairs worden vaak opgeroepen door de anti-entropy repair. In latere versies van Cassandra wordt de anti-entropy continue gecontroleerd. Anti-entropie betekend dat alle replica's vergeleken worden en dat deze ook geüpdatet worden naar de meest recente versie (Kunz, 2013).

Bij Cassandra staan alle nodes constant in communicatie met elkaar. Wanneer data opgehaald wordt, dan is er een kans dat deze vergeleken wordt met de andere replica's. Wanneer hier verschillen opgemerkt worden dan gaat Cassandra de data herstellen naar een consistente staat. Hiervoor zijn drie soorten read repairs (Strickland, 2014):

1. **Synchronous read repair:** Wanneer de data vergeleken wordt gebeurt dit op basis van de checksum die aan de andere nodes gevraagd wordt. Als deze checksum niet overeenkomt dan wordt de volledige replica doorgestuurd. Vervolgens wordt dan naar de timestamp gekeken van de data en wordt de oudste geüpdatet. Dit betekend dat de oude replica's geüpdatet worden als ze opgevraagd worden.

2. **Asynchronous read repair:** Cassandra heeft ook een systeem voor data die niet gecontroleerd wordt bij de reads. Hierbij wordt een kans ingesteld wanneer de data gecontroleerd wordt. Deze kans is standaard tien procent. Dit wil zeggen dat de replica's tien procent van de tijd gecontroleerd worden. Als hier verschillen opgemerkt worden dan wordt de data tijdens de read hersteld.
3. **Manuele repair:** Dit is een volledige repair en zou ook regelmatig uitgevoerd moeten worden. Men kan deze repair forceren via "nodetool repair", maar Cassandra voorziet ook een veld in de instellingen om dit op regelmatige basis uit te voeren. De standaard waarde om een manueel herstel door te voeren is tien dagen.

Bij manuele herstelling kan de opmerking gemaakt worden dat hier problemen kunnen optreden als men een record verwijderd heeft. Dit is echter niet het geval doordat Cassandra wacht om een record effectief te verwijderen. In plaats van een record meteen te verwijderen, wordt hier een tombstone marker aan mee gegeven zodat Cassandra weet dat dit record niet teruggegeven mag worden. Deze records worden uiteindelijk wel verwijderd via de garbage collection van Cassandra. Op basis van de tombstone bepaald de garbage collection dan of het record effectief effectief verwijderd mag worden (Strickland, 2014).

8.3 Ondervindingen bij het uitzetten van een node

Om het gedrag bij het uitvallen bij een node te verifiëren, werd gebruik gemaakt van de cluster die eerder opgezet was (5). In deze cluster zijn vier nodes aanwezig. Om hier het gedrag te testen werd er een keyspace aangemaakt met replicatie factor vier. Zo zou de data op iedere node aanwezig horen te zijn. De replicatie strategie is hier SimpleStrategy omdat er maar één datacenter in deze cluster aanwezig is. Vervolgens werd er data in de cluster geïmporteerd. Na het importeren werd gewacht tot de data over alle nodes verdeeld was. Dit kon men waarnemen in OpsCenter. Toen alle replica's aangemaakt waren kon het testen beginnen.

Door de uitgekozen strategie en replicatie factor zou op iedere node een replica van de data aanwezig moeten zijn. Om dit na te gaan werden alle nodes alleen opgestart en hier werd dan gekeken op een specifiek record aanwezig was. Deze manier van werken werd voor een aantal records herhaald om toevalligheden te vermijden. Hier kon telkens vastgesteld worden dat de data op de node aanwezig was. Op deze manier kan aangetoond worden dat de replicatie van Cassandra uitstekend werkt.

Om de consistentie van Cassandra na te gaan werd een gelijkaardig proces gevolgd. Hierbij werd telkens één node van de cluster opgestart terwijl de andere offline wa-

ren. Op deze ene node werden dan een aantal records geüpdatet. Dit werd herhaald voor alle nodes van de cluster. Vervolgens werden alle clusters opgestart en werd gewacht tot er geen activiteit meer gezien kon worden in het dashboard van OpsCenter. Toen dit proces voltooid was op alle clusters werd nagegaan welke versie van de records teruggegeven werd. Hier werd voor alle aangepaste records de geüpdatete versie teruggegeven net zoals de ontwikkelaars van Cassandra beloven.

Hoofdstuk 9

Back-ups in Cassandra

9.1 Snapshots

Om back-ups te maken voorziet Cassandra snapshots. Een snapshot is een verzameling van pointers die naar de data in de database verwijzen. Men kan zowel van alle keyspaces, één keyspace of één tabel een snapshot nemen zolang het Cassandra nodes heeft die online zijn. Ook van een ganse cluster kan een snapshot genomen worden, maar hier kan het wel zijn dat de data in verschillende nodes niet consistent is. De snapshots telkens in een nieuwe file opgeslagen worden, maar kunnen ook incrementeel groeien (DataStax, 2016a).

Snapshots worden via nodetool genomen. De enige voorwaarde die opgelegd wordt bij het maken van snapshots is dat er voldoende plaats aanwezig is op de node waar de snapshot geplaatst wordt. Het commando om een snapshot te nemen ziet er als volgt uit:

```
$ nodetool -h host -p JMXport snapshot mykeyspace
```

Het gebruik van snapshots heeft wel een nadeel. Het verhindert het verwijderen van oude, overbodige records (DataStax, 2016a).

Om een snapshot te herstellen moet men er eerst voor zorgen dat de tabellen die hersteld moeten worden aangemaakt zijn en ook leeg zijn. Daarna kan men de snapshot inladen via de sstableloader tool. Hierna sluit men de node die hersteld moet worden en daarna voert men het commando "nodetool drain" uit. Vervolgens moet de commitlog map leeg gemaakt worden. Een andere map die leeg gemaakt moet worden, met uitzondering van de submappen snapshots en backups, is data map van de keyspace. Hierna zoekt men de meest recente snapshot en plaatst deze in de keyspace map. Als

er incrementele back-ups zijn genomen dan moet men deze ook in deze map plaatsen. Tot slot start men de node opnieuw op en voert men het commando "nodetool repair" uit. Op deze manier is node hersteld a.d.h.v. een snapshot (DataStax, 2016a).

9.2 Nood aan back-ups in Cassandra

Door met replica's te werken en de manier waarop dit gebeurt in Cassandra zou men zich kunnen afvragen of het wel nodig is om back-ups te nemen. Voor deze vraag beantwoord kan worden moet eerst duidelijk het verschil tussen replica's en snapshots uitgelegd worden.

Een snapshot neemt als het ware een foto van de databank op een bepaald tijdstip, vandaar ook de term snapshot. In deze snapshot worden de referenties naar de data bij gehouden. Verder wordt er ook voor gezorgd dat verwijderde records niet echt verwijderd worden (8.2.3). Op die manier wordt er gegarandeerd dat de snapshot volledig blijft. Door deze manier van werken is de snapshot binnen Cassandra dan ook een volwaardige back-up.

Bij replica's wordt vaak verkeerdelijk aangenomen dat dit een synoniem is voor back-ups. Zoals eerder uitgelegd (3.2) werd, worden replica's op een andere node opgeslagen op de beschikbaarheid van de data te garanderen. Hierbij wordt de historische data, zoals aanpassingen aan de data, niet gekopieerd naar de andere nodes. Dit is al een eerste aanwijzing waarom dit niet als een volwaardige back-up beschouwd mag worden. Een andere, belangrijkere reden is dat door de manier waarop gerepliceerd wordt binnen Cassandra, de replica's binnen enkele seconden de nieuwe waarden bevatten. Stel dat hier een wijziging of verwijdering gebeurt, wordt dit onmiddellijk doorgegeven aan de andere nodes. Als deze handeling verkeerdelijk gebeurd is, dan kan men met replica's de originele data niet meer terugzetten.

Nu de verschillen tussen snapshots en replica's duidelijk zijn is het ook veilig om te stellen dat binnen Cassandra nog altijd back-ups nodig zijn. Hierbij gaat het dan vooral om verlies van data tegen te gaan die verkeerdelijk gewijzigd of verwijderd wordt. Via replica's kan deze niet meer hersteld worden.

Hoofdstuk 10

Use cases voor Cassandra

10.1 Messaging

Voor messaging systemen is het zeer belangrijk dat er 100% uptime is. Door de peer-to-peer architectuur van Cassandra is er geen single point of failure. Als hier ook nog eens multi-datacenter replicatie aan toegevoegd wordt is het bijna onmogelijk dat het systeem downtime kent (Chan, 2014).

Binnen deze use case zijn de belangrijkste gebruikers Instagram, New York Times, ComCast. . .

10.2 Fraude bestrijding

Cassandra voorziet real-time monitoring. Hiervoor kan nodetool of een ander tools zoals Datastax OpsCenter. Op deze manier kan verdacht gedrag snel opgespoord worden. Toch hoeft de fraude bestrijding niet enkel op deze manier te gebeuren.

Nguyen (2014) geeft een mooi voorbeeld om spam mail op te sporen bij Nextgate. Binnen de database wordt de data zo gemodelleerd dat alle berichten met eenzelfde inhoud in dezelfde rij terecht komen. Door een simpele count uit te voeren op de rijen kan men nagaan hoe vaak eenzelfde bericht verstuurd is. Als de count een bepaald getal overschrijdt dan wordt het bericht als spam beschouwd.

Naast Nextgate zijn er ook nog andere belangrijkste gebruikers zoals Barracuda Networks, ZoneFox, Iovation. . .

10.3 Product catalogi en afspeellijsten

Door de enorme schaalbaarheid, de hoge performantie en de hoge beschikbaarheid is Cassandra uitstekend geschikt om product catalogi en afspeellijsten bij te houden. Zowel product catalogi als afspeellijsten kunnen exponentieel groeien.

De bekendste gebruikers binnen deze use case zijn SoundCloud, Spotify, Reddit, Netflix. . .

Bij Reddit wordt Cassandra in veel zaken gebruikt maar de belangrijkste toepassingen zijn het voting systeem, gelijke pagina's en gelijke subreddits (Harvey, 2013).

10.4 Internet Of Things en sensor data

Door de hoge doorvoersnelheid is Cassandra uitermate geschikt voor het opslaan van sensor gegevens. Het is dan ook vooral deze eigenschap die vaak de doorslag geeft om voor Cassandra te kiezen als men bij deze use case terecht komt.

Bij deze use case zijn de bekendste gebruikers Aeris, i2O, CERN, NASA. . .

Keller (2013) legt uit waarom er binnen NASA met Cassandra gewerkt wordt. Cassandra laat volgens hem toe om de data op een veel natuurlijker manier in te geven. Ook is het mogelijk met Cassandra om zeer snel de resultaten van een query te zien. Hier heeft hij het voorbeeld dat zeer snel alle informatie over een specifiek ip-adres op een specifieke tijd kan opvragen. Een ander groot pluspunt voor Cassandra is de ingebouwde time-to-live. Op die manier wordt data die niet meer relevant is automatisch verwijderd. De hoge schrijfsnelheid is voor NASA ook belangrijk omdat deze organisatie dag en nacht veel verschillende feeds ontvangt.

10.5 Aanbevelingen en personalisatie

Het grote probleem bij aanbevelingen is dat de data op een persoon afgesteld moet zijn en dat een grote hoeveelheid data real time verwerkt moet worden. Dit is echter geen probleem voor Cassandra want hier is de database voor geoptimaliseerd.

Enkele bekende bedrijven zoals ebay, Yahoo, Eventbrite. . . gebruiken Cassandra om deze problemen op te lossen.

Binnen Unity wordt ook Cassandra gebruikt. Mäkinen (2015) legt uit dat hiervoor overgestapt werd van MongoDB naar Cassandra. Dit was omdat er behoefte was aan

een sneller systeem. Met meer dan 500 miljoen documenten in MongoDB was het niet meer efficiënt om hier data in op te zoeken. De hogere snelheid was vooral nodig bij Unity Ads, hiermee kunnen game developers advertenties toevoegen aan hun games. Dit slaat op wie op welke moment welke advertentie te zien krijgt. Hierdoor zijn er veel writes en wordt de time-to-live gebruikt om ervoor te zorgen dat oude records verwijderd worden. De data die hier weggeschreven wordt, wordt dan gebruikt om te bepalen welke advertenties aan welke gebruiker getoond worden.

10.6 Markten voor Cassandra

Door de geschiktheid voor aanbevelingssystemen kan Cassandra eindgebruikers persoonlijke suggesties geven. Door de hoge doorvoersnelheid van Cassandra kan dit real time gebeuren. Op dezelfde manier als een product catalogus gemaakt kan worden kan een boodschappen wagentje gecreëerd worden. Doordat Cassandra ook uitstekend geschikt is voor messaging is het mogelijk om dit in te zetten voor notificaties voor de eindgebruiker. Met al deze zaken in het gedachten kan besloten worden dat Cassandra een goede keuze is voor retailers en aanbieders van digitale media.

Een andere markt voor Cassandra is de financiële sector. Hier wordt immens veel data verstuurd en opgeslagen, hier hoeft men enkel nog maar aan transacties te denken. Cassandra biedt ook uitstekende mogelijkheden om fraude tegen te gaan wat toch ook een groot pluspunt is in deze sector. Cassandra heeft in deze markt al enkele gebruikers zoals ING, UBS, Xoom. . . Een relatief nieuwe trend is dat verzekeraars, zoals Corona en AXA, sensoren in bijvoorbeeld een auto laten plaatsen en dat op basis hiervan dan een verzekeringspolis wordt opgesteld. Hiervoor wordt veel data opgeslagen om deze polis correct te kunnen bepalen. Zoals hierboven vermeld is Cassandra ook voor deze job uitstekend geschikt.

De hierboven vermelde markten zijn markten waar Cassandra bijna kant en klare oplossingen voor aanbiedt. Natuur kan Cassandra ook in andere markten gebruikt worden, maar hier zal waarschijnlijk meer aandacht moeten besteed worden aan het datamodel en zal er samen gewerkt moeten worden met andere technologieën.

Hoofdstuk 11

Conclusie

De schaalbaarheid van Cassandra werd nagegaan door nodes toe te voegen aan de cluster en het verwijderen hiervan. Via de configuratie files van Cassandra is dit een heel karwei om te doen. Verder werd er ook niet in geslaagd om op deze manier een werkende cluster te verkrijgen binnen deze bachelorproef. Toen werd overgestapt naar het OpsCenter van DataStax was dit echter een ander verhaal. Via deze weg was het zeer eenvoudig om een cluster te beheren en hier nodes aan toe te voegen of nodes te verwijderen.

Na de schaalbaarheid werd de betrouwbaarheid van de database getest. Dit werd gedaan a.d.h.v. het uitschakelen van de nodes en zo af te toetsen of de theorie wel strookt met de werkelijkheid. Bij uitvallen van de nodes kon telkens vastgesteld worden dat de data beschikbaar bleef. Het hinting systeem van Cassandra bleek ook uitstekend te werken in deze test. Toen enkel één node online was werden hier toch updates van records op uitgevoerd en bij het opnieuw opstarten van een node kon via OpsCenter vastgesteld worden dat er data uitgewisseld werd tussen de nodes. Als men vervolgens de node, die eerst online was, uitschakelt, kan men toch de geüpdatete data terugvinden op de node die juist terug online komt. Met het onderzoek dat in deze bachelorproef gedaan is, kan besloten worden dat Cassandra geen last heeft van "single points of failure".

Bij de data modellering van Cassandra zijn er toch een aantal eigenaardigheden als men vanuit een SQL omgeving komt. Zo is het principe van modelleren naar de query's die uitgevoerd zullen worden bijzonder. Het is echter wel nodig om hierbij stil te staan aangezien de primaire sleutel, die bestaat uit de partitie en clustering sleutel, restricties oplegt aan de WHERE clause. Toch is de reden waarom Cassandra deze restricties oplegt goed te begrijpen. Want op deze manier kan de data zeer snel opgehaald worden.

Voor de back-ups kan besloten worden dat deze nog altijd nodig zijn. De replica's zijn hier geen volwaardige vervangers voor. Door het gebruik van replica's wordt de beschikbaarheid van de data gegarandeerd, maar dit is geen garantie tegen corrupte data.

Eén opmerking die bij dit alles moet gemaakt worden is dat alle test op virtuele machines zijn uitgevoerd en waarbij er slechts één datacenter beschikbaar was. Door het gebruik van virtuele machines zijn de absolute tijdsgegevens in deze bachelorproef niet representatief voor een echte cluster waar de machines voor Cassandra alleen zijn voorbehouden.

Binnen het CAP theorema was de focus bij Cassandra vooral op beschikbaarheid en partitionering. Hier slaagt Cassandra zeer goed in. Maar meer zelf op het gebied van consistentie wordt zeer goed gescoord door de goed uitgedachte herstelmechanismes. Met de woorden van Kan (2014) over wat Cassandra nu precies is kan de bachelorproef afgesloten worden, want niets in dit onderzoek kon iets van wat hij zei weerleggen.

"Cassandra can be simply described in a single phrase: a massively scalable, highly available open source NoSQL database that is based on peer-to-peer architecture."
(Kan, 2014)

Bibliografie

- Ahmed, A. S. (2015). Cassandra terminology. <http://exponential.io/blog/2015/01/08/cassandra-terminology/>. Geraadpleegd op 2016-05-18.
- Bauer, G. R. (2013). Apache cassandra: The case against the byteorderedpartitioner. <http://www.geroba.com/cassandra/apache-cassandra-byteorderedpartitioner/>. Geraadpleegd op 2016-04-25.
- Bisbee, S. (2015). Scale it to billions - what they don't tell you in the cassandra readme. <http://blog.threatstack.com/scaling-cassandra-lessons-learned>. Geraadpleegd op 2016-03-24.
- Brewer, E. (2012). Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29.
- Brewer, E. A. (2000). Towards robust distributed systems. In *PODC*, volume 7.
- Cannon, P. (2012). Simple data importing and exporting with cassandra. <http://www.datastax.com/dev/blog/simple-data-importing-and-exporting-with-cassandra>. Geraadpleegd op 2016-03-04.
- Cantoni, B. (2016). Multinode template. <https://github.com/bcantoni/vagrant-cassandra/tree/master/2.MultiNode>. Geraadpleegd op 2016-03-25.
- Chan, W. (2014). Apache cassandra + messaging applications = a match made in heaven. <http://www.datastax.com/2014/11/apache-cassandra-messaging-applications-a-match-made-in-heaven>. Geraadpleegd op 2016-05-19.
- Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4.
- DataStax (2016a). Backing up and restoring data. <http://docs.datastax.com/en/>

- cassandra/2.1/cassandra/operations/ops_backup_restore_c.html. Geraadpleegd op 2016-04-28.
- DataStax (2016b). Initializing a multiple node cluster (single data center). <https://docs.datastax.com/en/cassandra/2.1/cassandra/initialize/initializeSingleDS.html>. Geraadpleegd op 2016-02-26.
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM.
- Ellis, J. (2009). Cassandra: Open source bigtable + dynamo. <http://www.slideshare.net/jbellis/cassandra-open-source-bigtable-dynamo>. Geraadpleegd op 2016-01-11.
- Fowler, M. (2012). Nosqldefinition. <http://martinfowler.com/bliki/NosqlDefinition.html>. Geraadpleegd op 2016-01-11.
- Fowler, M. (2013). Introduction to nosql. https://www.youtube.com/watch?v=qI_g07C_Q5I. [videofile], Geraadpleegd op 2016-01-11.
- Harvey, J. (2013). Reddit upvotes apache cassandra for horizontal scaling; managing 17,000,000 votes daily. <http://planetcassandra.org/blog/interview/reddit-upvotes-apache-cassandra-for-horizontal-scaling-managing-17000000-votes-daily/>. Geraadpleegd op 2016-05-19.
- Hewitt, E. (2010). *Cassandra: the definitive guide*. "O'Reilly Media, Inc."
- Hobbs, T. (2015). Basic rules of cassandra data modeling. <http://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling>. Geraadpleegd op 2016-03-20.
- Kan, C. (2014). *Cassandra Data Modeling and Analysis*. Packt Publishing Ltd.
- Keller, C. (2013). 5 minute c* interview | nasa. <http://planetcassandra.org/blog/5-minute-c-interview-nasa/>. Geraadpleegd op 2016-05-19.
- Kunz, G. (2013). Antientropy. <https://wiki.apache.org/cassandra/AntiEntropy>. Geraadpleegd op 2016-05-18.
- Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.
- Lerer, B. (2015). A deep look at the cql where clause. <http://www.datastax.com/dev/blog/a-deep-look-to-the-cql-where-clause>. Geraadpleegd op 2016-04-25.

- McFadin, P. (2013). Cassandra 2.0 and timeseries. <http://www.slideshare.net/patrickmcfadin/cassandra-20-and-timeseries>. Geraadpleegd op 2016-04-25.
- Mäkinen, J. (2015). Gaming dev platform unity powers up with cassandra; migrates away from mongodb for a scalable low latency solution. <http://planetcassandra.org/blog/interview/gaming-dev-platform-unity-powers-up-with-cassandra-migrates-away-from-mongodb-for-a-scalable-low-latency-solution/>. Geraadpleegd op 2016-05-19.
- Nguyen, H. (2014). Nexgate: Social media security company nexgate relies on cassandra for fraud detection. <https://www.youtube.com/watch?v=ic1ez2WFd0g>. [videofile], Geraadpleegd op 2016-05-19.
- Ploetz, A. (2015). We shall have order. Geraadpleegd op 2016-03-24.
- Sadalage, P. (2014). Nosql databases: An overview. <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>. Geraadpleegd op 2016-01-11.
- Schumacher, R. (2015). A brief introduction to apache cassandra. <https://academy.datastax.com/resources/brief-introduction-apache-cassandra>. Geraadpleegd op 2016-01-11.
- Strickland, R. (2014). *Cassandra High Availability*. Packt Publishing Ltd.

Lijst van figuren

3.1	Illustratie SimpleStrategy (Strickland, 2014)	11
4.1	Logische datastructuren in Cassandra	15
5.1	Cassandra: Instellingen deel 1	17
5.2	Cassandra: Instellingen deel 2	18
5.3	Installatie van Cassandra door OpsCenter	19
5.4	Rondleiding in OpsCenter	19
5.5	nodetool	20
5.6	Toevoegen van een node via OpsCenter	21
5.7	Verwijderen van een node: Selecteer de node	22
5.8	Verwijderen van een node: Selecteer de node	22
5.9	Verwijderen van een node: Selecteer de node	23

Lijst van tabellen

5.1	Configuratie van de Cassandra Cluster	17
7.1	Importeren van data met cqlsh	32