



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 weer 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 voor 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 datamodellering 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 mijn bachelorproef, de morele ondersteuning, het eenvoudiger 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	6
1.3	Probleemstelling en onderzoeksvragen	6
2	Methodologie	8
3	Architectuur	10
3.1	Partitionering	10
3.1.1	Problemen met de ByteOrderedPartitioner	11
3.2	Replicatie	12
3.2.1	Replicatiestrategieën	12
3.3	Snitches	13
3.4	Seed node	15
4	Data structuren in Cassandra	16
4.1	Column	16
4.2	Row	17
4.3	Table	17
4.4	Keyspace	18
5	Opzetten van de Cassandra cluster	19
5.1	Apache Cassandra	19
5.2	DataStax OpsCenter	19
5.3	Toevoegen van een node	23
5.4	Verwijderen van een node	24
6	Datamodellering in Cassandra	27
6.1	Primaire sleutel	27
6.2	Partitiekolom	27
6.3	Clusteringkolom	28

6.4	De WHERE clause	28
6.4.1	Restricties opgelegd door de partitiekolommen	28
6.4.2	Restricties opgelegd door de clusteringkolommen	29
6.5	Doelen bij datamodelling in Cassandra	30
6.6	Datamodel opstellen voor Cassandra	31
6.7	Datamodel en de hoge snelheid	33
7	Data importeren in Cassandra	35
7.1	Importeren via cqlsh	35
7.2	Importeren met sstableloader	36
7.3	Importeren met cassandra-loader	36
7.4	Testen van de verschillende loaders	37
8	Gedrag bij uitvallen van een node	38
8.1	Gossip en foutdetectie	38
8.2	Herstelmechanisme	38
8.2.1	Hinted handoff	39
8.2.2	Anti-entropy repair	39
8.2.3	Read repair	39
8.3	Ondervindingen bij het uitzetten van een node	40
9	Back-ups in Cassandra	42
9.1	Snapshots	42
9.2	Nood aan back-ups in Cassandra	43
10	Use cases voor Cassandra	44
10.1	Messaging	44
10.2	Fraudebestrijding	44
10.3	Productcatalogi en afspeellijsten	45
10.4	Internet Of Things en sensor data	45
10.5	Aanbevelingen en personalisatie	45
10.6	Markten voor Cassandra	46
11	Conclusie	47

Hoofdstuk 1

Inleiding

1.1 NoSQL

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

NoSQL databanken vinden hun oorsprong in de moeilijkheid om objecten uit een object georiënteerd programma op te slaan in een relationele databank: de zogenoemde 'impedance mismatch', bovendien is er het feit dat relationele databanken vaak niet goed werken op een cluster of niet met grote hoeveelheden realtime data om kunnen gaan. . . Als gevolg hiervan maken NoSQL databanken vaak geen gebruik van een relationeel model, zijn ze gemaakt om op clusters te werken en zijn ze bovendien schema-less. . . Kortom databanken onder de noemer NoSQL zijn aangepast om de problemen, die zich nu binnen de relationele databanken manifesteren, aan te pakken (Fowler, 2012).

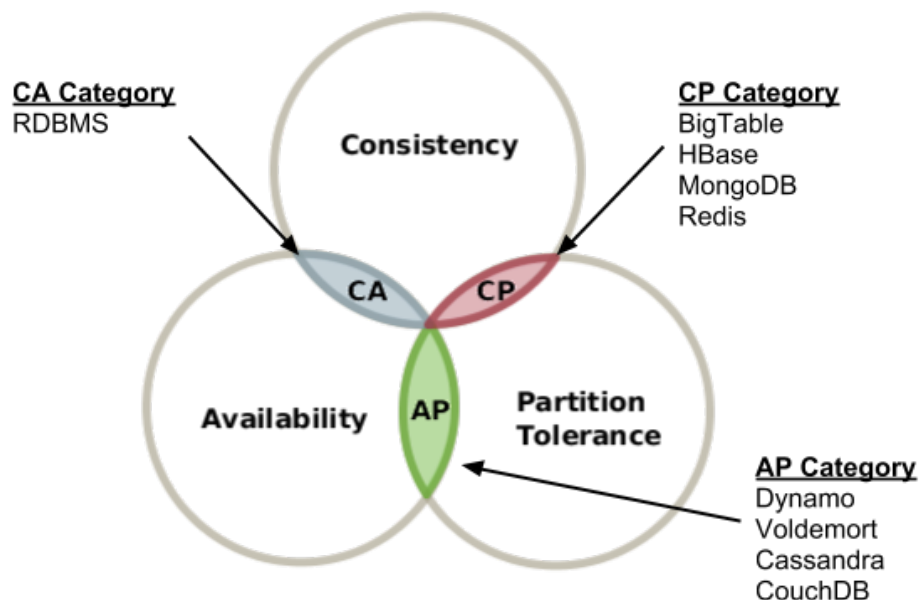
Sadalage (2014) zegt dat binnen de NoSQL databanken vier grote types naar voor geschoven kunnen worden, namelijk key-value stores (Riak, Redis. . .), document stores (MongoDB, CouchDB. . .), column family stores (Cassandra, HBase. . .) en graph databases (Neo4J, Infinite Graph. . .). Elk van deze types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

schillende specifieke use cases voor.

Bij gedistribueerde systemen zijn de consistentie, de beschikbaarheid en de partitie tolerantie enorm belangrijk. Brewer (2000) stelde hieromtrent het CAP theorema op en stelde dat het onmogelijk was om binnen een gedistribueerd systeem deze drie zaken tegelijk te realiseren. Hij gaf hiermee de aanzet tot de veronderstelling dat er slechts twee van de drie punten in het CAP theorema tegelijk verkregen kunnen worden. Men is dus genoodzaakt om één eigenschap op te offeren.

Maar wat houden deze drie punten die Brewer aanhaalt nu juist in?

- **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.



Figuur 1.1: CAP theorema (Alvarado, 2014)

Als softwareontwikkelaar heeft men keuze uit een hele reeks databanken. Voor de komst van NoSQL was er echter geen sprake van een keuze voor een al dan niet relationele databank. Lange tijd kon er enkel een relationele database gebruikt worden omdat er niets anders voor handen was. Wanneer men voor een bepaalde databank kiest is het belangrijk dat men vooraf weet hoe de data gebruikt zal worden. Wanneer men hier inzicht in heeft, kan men gericht gaan zoeken naar een geschikte databank, hetzij een relationele databank, hetzij een NoSQL databank.

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 vindt 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 eerst en vooral een hoge throughput toelaat, dat daarnaast miljarden write operaties per dag moet aankunnen en dat bovendien mee kan 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 ook realtime antwoorden toelaat (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 snelheid op te offeren voor extra 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 het nastreven van de drie punten 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? Een laatste punt waarbij er stilgestaan wordt, is 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 dat 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 aan de praktijk te kunnen toetsen 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 Gent ingeladen in deze virtuele cluster. Voor deze data kon ingeladen worden, moest er eerst stilgestaan worden bij het datamodel van deze data. 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 (één van de nodes van de databank) uitgeschakeld om te zien hoe Cassandra hierop reageert. Doordat dit in een virtuele omgeving gebeurde, was er geen risico op verlies van kritieke data.

Hoofdstuk 3

Architectuur

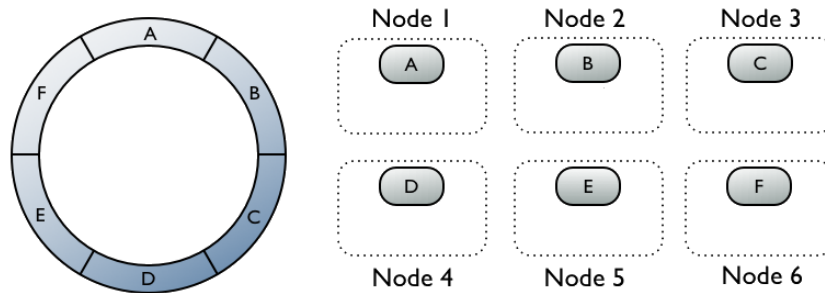
3.1 Partitionering

Eén 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 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 primaire 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 primaire 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 primaire sleutel. De tokens worden berekend aan de hand van de hexadecimale representatie van de leidende tekens van de primaire sleutel. Op deze manier kan men, net zoals men met een cursor in een SQL omgeving

zou doen, de tabel geordend overlopen, gesorteerd op de primaire sleutel.



Figuur 3.1: Illustratie partitioner (DataStax, 2016b)

De cluster op bovenstaande figuur (Figuur 3.1) bestaat uit 6 nodes. De partitioner zal er nu voor zorgen dat er zes ranges zijn voor de hashwaarden van de primaire sleutel, in dit geval A, B, C, D, E en F. Wanneer een record ingevoerd wordt, zal de partitioner hier nu de hashwaarde voor berekenen. Indien de hashwaarde zich in de range A bevindt, zal de partitioner er voor zorgen dat deze record op node 1 geplaatst wordt.

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 door 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 replicatiefactor te blijven behouden zelfs indien bepaalde nodes uitgevallen zouden zijn. Deze replicatiefactor wordt ingesteld op keyspace niveau, wat vergelijkbaar is met het database of schema niveau bij relationele databanken (4.4). De replicatiefactor bepaalt hoeveel replica's van de data bijgehouden worden. Op basis van de partitioner en de replicatiestrategie wordt bepaald waar de replica's opgeslagen worden. Om deze replicatie in goede banen te leiden zijn er een aantal strategieën uitgewerkt (Kan, 2014).

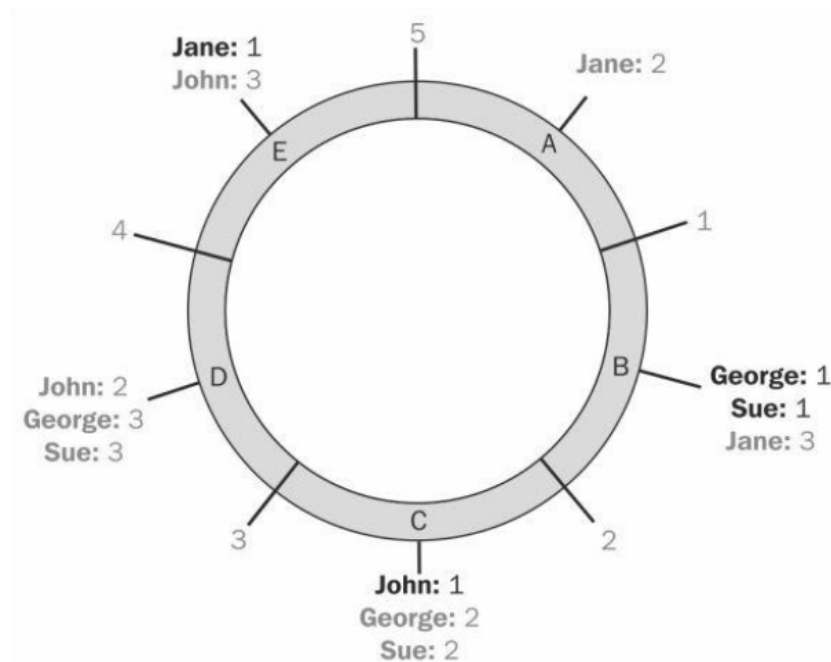
3.2.1 Replicatiestrategieën

De replicatiestrategie 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 de andere strategie te gebruiken. Wanneer de standaard partitioner, de Murmer3Partitioner, gebruikt wordt, zal 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 3.2 wordt een voorbeeld gegeven waarbij de replicatiefactor 3 is. De zwarte namen zijn de originele data en de grijze zijn replica's.



Figuur 3.2: Illustratie SimpleStrategy (Strickland, 2014)

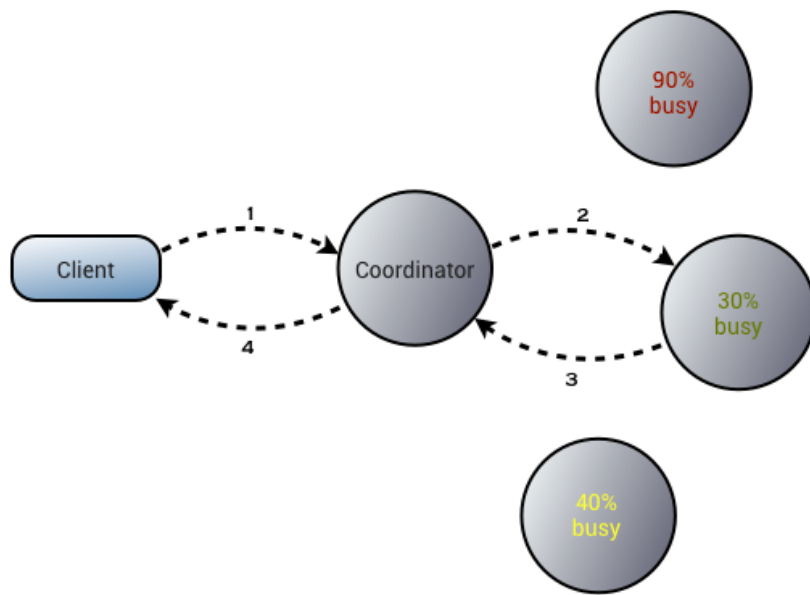
NetworkTopologyStrategy

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. Snitches worden verder toegelicht (3.3). 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 replicatiefactor instellen.

3.3 Snitches

Een snitch is ervoor verantwoordelijk om de data snel en efficiënt op te halen binnen de cluster. Hierdoor is het belangrijk dat de snitch snel de fysieke locatie van de replica's kan achterhalen.



Figuur 3.3: Illustratie werking snitch (PlanetCassandra, 2013)

Op figuur 3.3 is te zien hoe een snitch nu juist te werk gaat. Er wordt een query naar de coördinator, de node waarmee de client verbonden is, gestuurd. De snitch op de coördinator zal vervolgens proberen om informatie, zoals de huidige belasting, te verzamelen over de nodes waar de gevraagde data opgeslagen is. Op basis van de teruggekregen informatie zal de snitch dan beslissen aan welke node de query doorgegeven zal worden voor verdere verwerking.

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 tot welk datacenter behoren. Als men deze snitch gebruikt moet men dus de hele topologie uitschrijven. Iedere node moet ook op precies dezelfde manier geconfigureerd worden. Dit alles geeft 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 labelt 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 datastructuur die er bestaat. De kolom van een relationele database kan niet vergeleken worden met een kolom in Cassandra. Hun functie is volledig anders.

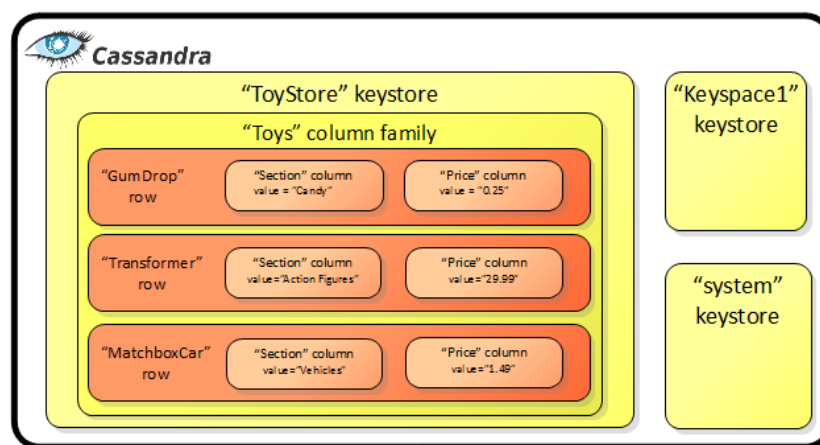


Image is Public Domain: Released July 11, 2010 by divconq.com

Figuur 4.1: Kolommen in Cassandra (Lampe, 2010)

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 zal 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 hoelang 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.

Aan de hand van figuur 4 kan verklaard worden waarom Cassandra tot de Column Family Store behoort, terwijl er toch sprake is van rijen. De kolommen worden in Cassandra namelijk in rijen opgeslagen.

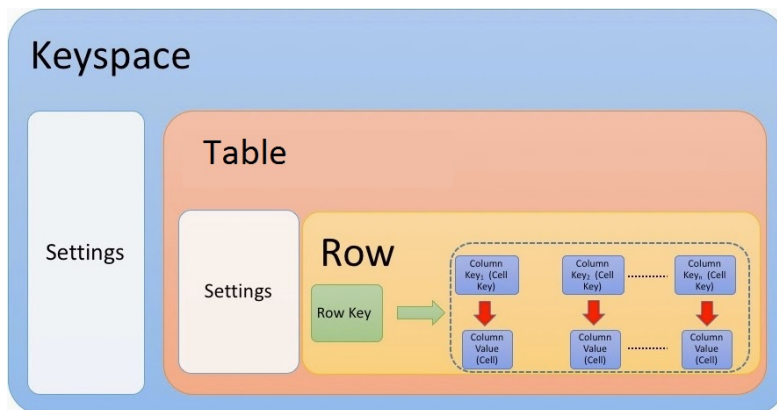
4.3 Table

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 vindt 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 groepeert 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 replicatiestrategie en replicatiefactor.



Figuur 4.2: 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, 2016c). 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 configuratiefile van Cassandra werkte deze perfect op iedere node.

5.2 DataStax OpsCenter

Uiteindelijk werd ervoor 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 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 om verschillende zaken gevraagd. Tabel 5.1 en figuur 5.1 geven weer hoe dit venster werd ingevuld.

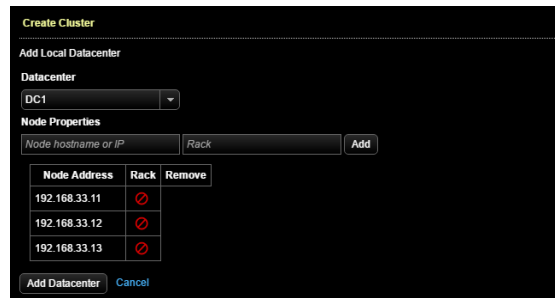
Property Name	Waarde
Cluster Name	BP Cluster
Type	local
Package	datastax community 2.1.11
Endpoint Snitch	GossipingPropertyFileSnitch
Username en password	vagrant/vagrant
Local Node Credentials	cassandra-node-1, cassandra-node-2, cassandra-node-3

Tabel 5.1: Configuratie van de Cassandra Cluster

The screenshot shows the 'Create Cluster' dialog in the OpsCenter application. The 'Cluster Name' field is set to 'Test Cluster'. The 'Provisioning Type' is set to 'Local'. The 'Package' is set to 'DataStax Community 2.1.11'. The 'Endpoint Snitch' is set to 'GossipingPropertyFileSnitch'. Below these, the 'Username' is 'vagrant' and the 'Password' is 'vagrant'. A section labeled 'Local Node Credentials (sudo)' contains a list box with three entries: 'cassandra-node-1', 'cassandra-node-2', and 'cassandra-node-3'. At the bottom of the dialog, 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).

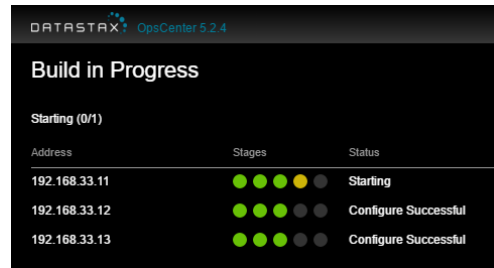


Node Address	Rack	Remove
192.168.33.11	X	
192.168.33.12	X	
192.168.33.13	X	

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.

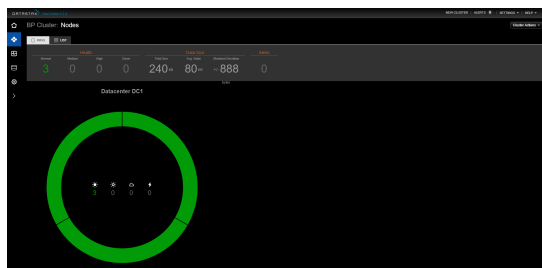
Hierna begint OpsCenter met de installatie van de Cassandra cluster (Figuur: 5.3). Deze installatie nam enige ogenblikken in beslag. De eerste keer kwam er een fout voor omdat er niet genoeg werkgeheugen was toegekend aan de slave nodes. In de setup die hier gebruikt werd, werd het minimum aanvaarde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



(b) Deel 2



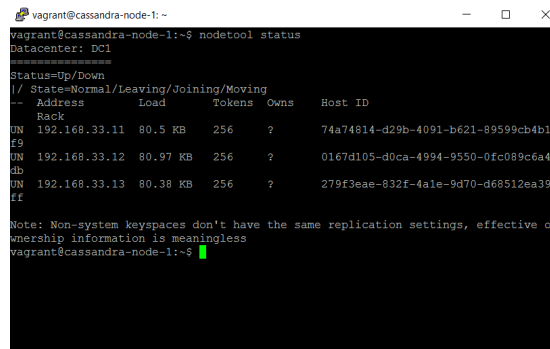
Na de installatie komt men terecht in het Dashboard van OpsCenter (Figuur 5.4a). Hierin worden een aantal zaken weergegeven zoals: de gezondheid van de cluster, het aantal write requests, de write request latency... Hier kunnen nog meer grafieken aan toegevoegd worden door de knop 'add graph' te gebruiken. 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 één 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 inneemt van alle data die op de cluster aanwezig is.



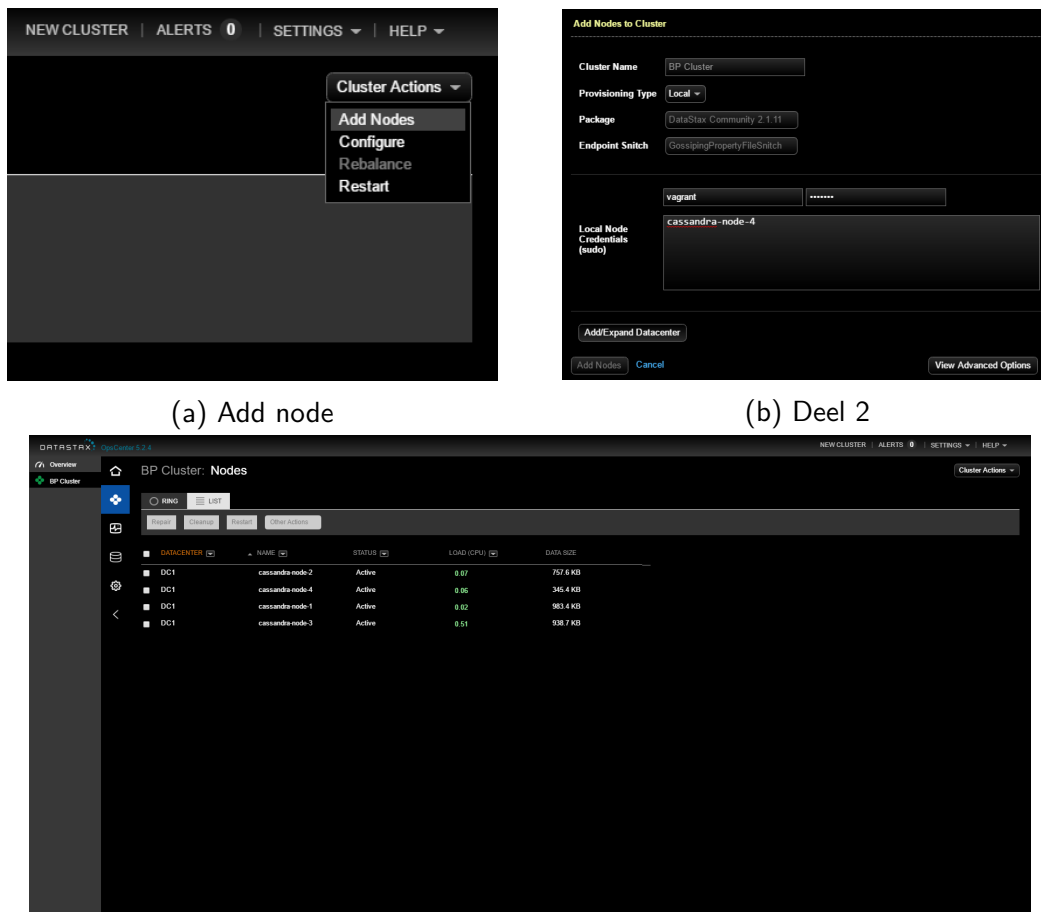
```
vagrant@cassandra-node-1:~$ nodetool status
Datacenter: DC1
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address      Load          Tokens       Owns    Host ID
--  --
UN 192.168.33.11  80.5 KB      256         ?       74a74814-d29b-4091-b621-89599cb4b1f9
UN 192.168.33.12  80.97 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 aan de cluster toegevoegd. Net zoals bij de installatie van de cluster wordt het scherm, waarin de aanmeldgegevens voor de gegeven node gevraagd worden, nog eens weergegeven (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).

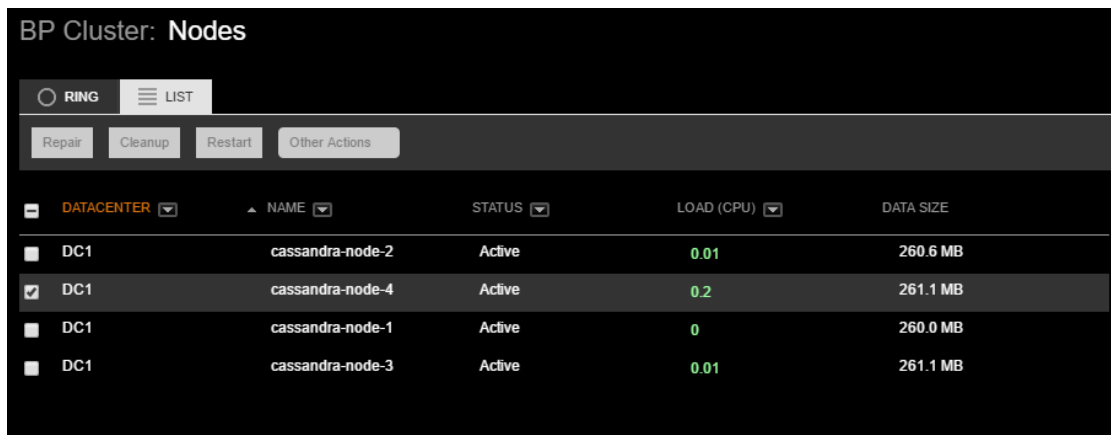


(c) Tabblad node met 4 nodes

Figuur 5.6: Toevoegen van een node via OpsCenter

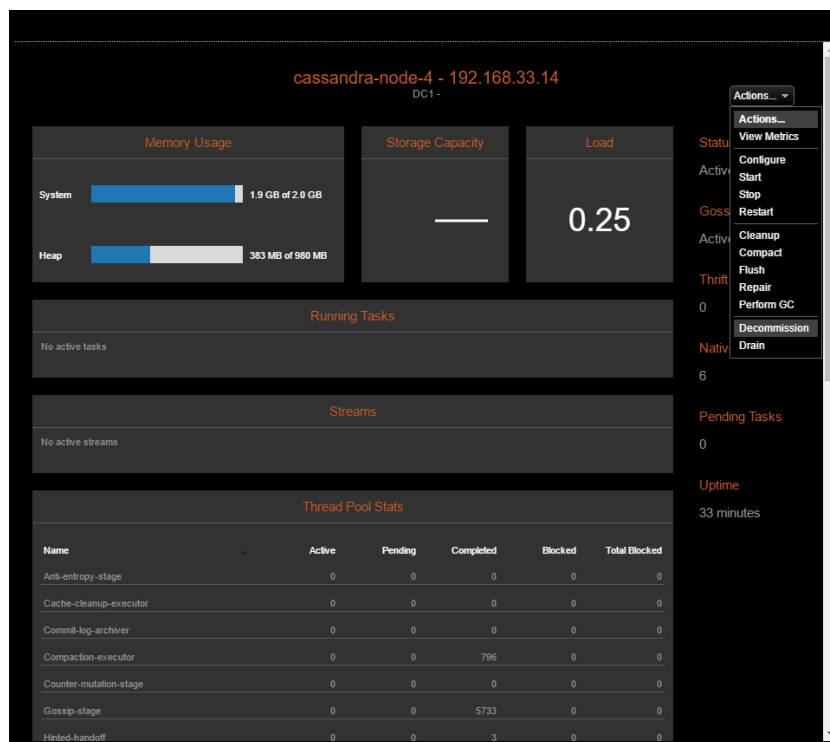
5.4 Verwijderen van een node

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

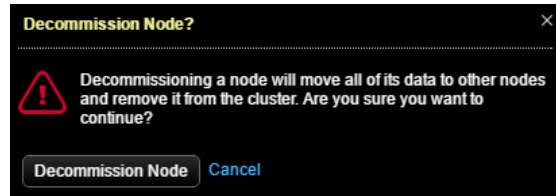
Daarna komt men op een volgend scherm terecht (figuur 5.8). Hier kiest men bij de acties voor "Decommission".



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 partitiekolommen (6.2) en clusteringkolommen (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 partitiekolom en zijn er geen clusteringkolommen.

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 Partitiekolom

Het eerste deel van de primaire sleutel bestaat uit de partitiekolommen. Het doel van deze partitiekolommen 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 partitiekolommen 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 partitiekolommen te kiezen of men komt hier al snel in de problemen.

6.3 Clusteringkolom

Het laatste deel van de primaire sleutel bestaat uit de clusteringkolommen. Deze bepalen de volgorde van de data op de fysieke media (Strickland, 2014). De clusteringkolommen 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 clusteringkolommen. Eveneens kan er enkel op de partitie- en clusteringkolommen gefilterd worden. Al deze restricties komen er door de achterliggende architectuur bij Cassandra (zie sectie 6.7).

6.4.1 Restricties opgelegd door de partitiekolommen

Een eerste restrictie die wordt opgelegd door de partitiekolommen is dat ofwel alle partitiekolommen worden opgenomen in de WHERE clause ofwel geen enkele. Dit is nodig omdat de partitioner anders de hashwaarde niet kan berekenen. Deze hashwaarde is echter nodig om te weten op welke node de data zich bevindt (zie sectie 3.1).

Een volgende restrictie die wordt opgelegd door de partitiekolommen 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 partitiekolom toegepast worden.

De laatste restrictie die wordt opgelegd door de partitiekolommen 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 clusteringkolommen

Voor de restricties op de clusteringkolommen 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:

Code fragment 6.1: Layout van de tabel NumberOfTwitterMessages

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

De eerste restrictie wordt hier opgelegd door de manier waarop de data opgeslagen is. Als men een voorwaarde wil vastleggen voor een clusteringkolom, dan dienen de clusteringkolommen die voor deze komen in de primaire sleutel ook vastgelegd te worden. Query 6.2 is dus niet mogelijk voor de tabel "NumberOfTwitterMessages" (6.1) omdat hier geen restrictie is op het veld "hour", terwijl dit voor het veld "minute" komt in de primaire sleutel. Dit is nodig omdat Cassandra de data anders niet efficiënt kan terugvinden.

Code fragment 6.2: Foutieve query bij meerdere partitiekolommen

```
SELECT nrOfTweets FROM NumberOfTwitterMessages
WHERE userid = 2222
AND date = '2016-04-25'
AND minute = 30
;
```

Om te zien waarom deze data niet efficiënt opgehaald kan worden met query 6.2, moet men kijken naar de manier waarop Cassandra de data opslaat binnen de partities (zie code fragment 6.3). Hierbij kan men zien dat er in een boomstructuur gewerkt wordt om de data op te slaan. Als men echter een top in deze boom weglaat (hier het veld "hour"), kan het pad naar het veld "nrOfTweets", het resultaat van de query, niet op een efficiënte manier bepaald worden.

Code fragment 6.3: Data opslag binnen een partitie

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

Tot Cassandra 2.2 was de IN operator enkel toegelaten op de laatste clusteringkolom. 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 clusteringkolom 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. Zo is query 6.4 perfect mogelijk. Query 6.5 is equivalent aan de vorige query, maar hier wordt ook aangetoond dat de multi-column slice met dezelfde kolom moet beginnen.

Code fragment 6.4: Multi-column slice restrictie

```
SELECT * FROM NumberOfTwitterMessages
WHERE userid = 2222
AND date = '2016-04-25'
AND (hour, minute)  $>=$  (12, 30)
AND (hour, minute)  $<=$  (14, 0)
;
```

Code fragment 6.5: De restrictie moet starten met dezelfde kolom

```
SELECT * FROM NumberOfTwitterMessages
WHERE userid = 2222
AND date = '2016-04-25'
AND (hour, minute)  $>=$  (12, 30)
AND (hour)  $<=$  (14)
;
```

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 partitiekolommen goed te kiezen. Zoals eerder vermeld is, worden de partitiekolommen 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 onbelangrijk. 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 gerealiseerd 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 clusteringkolommen 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 6.6:

Code fragment 6.6: Correcte modellering bij Cassandra

```
CREATE TABLE users_by_username (  
    username text PRIMARY KEY,  
    email text ,  
    birthday timestamp  
);
```

```
CREATE TABLE users_by_email (  
    email text PRIMARY KEY,  
    username text ,  
    birthday timestamp  
);
```

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 6.7:

Code fragment 6.7: Foutieve modellering bij Cassandra

```
CREATE TABLE users (  
    id uuid PRIMARY KEY,  
    username text ,  
    email text ,  
    birthday 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,

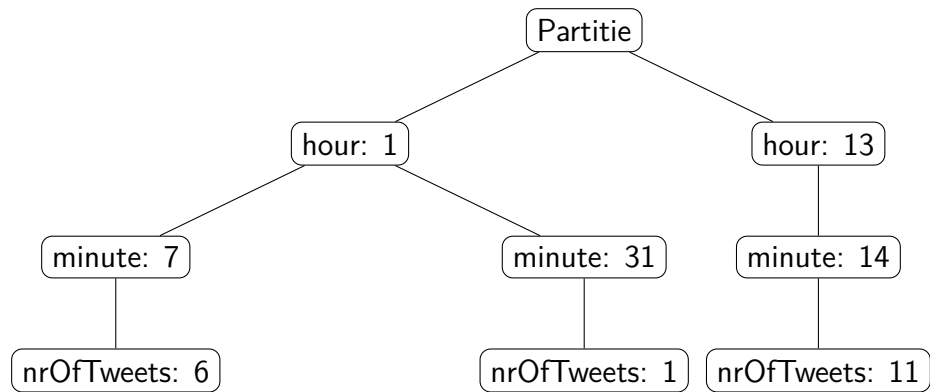
maar nu moet men meer dan één partitie read doen. Dus door een niet-doel proberen te bereiken is hier een belangrijk doel verloren gegaan.

6.7 Datamodel en de hoge snelheid

Nu alle concepten van datamodellering duidelijk zijn, kan er ook verklaard worden waarom dit kan bijdragen aan de hoge schrijf- en leessnelheid van Cassandra. Hiervoor moet men kijken hoe Cassandra de data fysiek opslaat op de nodes, voor de clusteringkolommen werd dit al even kort aangehaald (zie code fragment 6.3).

Als men nu de tabel "NumberOfTweets" (6.1) er opnieuw bijneemt, weet men dat de partitiekolommen "userid" en "date" de node zullen bepalen waarop de data fysiek aanwezig is. Men mag aannemen dat deze partitiekolommen ervoor zullen zorgen dat de data evenwichtig verspreid wordt over alle nodes. Doordat de data evenwichtig verspreid is zullen de query's ook verspreid worden over de nodes. Hoe het verdelen van de query's in zijn werk ging werd reeds uitgelegd bij de snitches (zie sectie 3.3). De hoge schrijfsnelheid wordt bekomen doordat de partitioner het werk over alle nodes verdeelt (zie sectie 3.1). Het is dus zo dat hoe beter de data verspreid is over alle nodes, hoe beter het werk verdeeld kan worden over de nodes. Door een goede verdeling van het werk kunnen veel query's tegelijk uitgevoerd worden. Het datamodel is verantwoordelijk voor het verspreiden van de data over alle nodes door middel van de partitiekolommen en zo ook rechtstreeks voor het verspreiden van alle query's over alle nodes.

De clusteringkolommen hebben ook een grote bijdrage tot de hoge leessnelheid van Cassandra. De clusteringkolommen bepalen hoe de data opgeslagen wordt binnen de partitie. Met een partitie wordt hier bedoeld op data die gegroepeerd is volgens identieke waarden voor de partitiekolommen. Binnen de partitie worden de clusteringkolommen opgeslagen in een boomstructuur, zoals al eerder vermeld werd. Een boomstructuur is een structuur die zeer geschikt is om er opzoekingen in te doen. Cassandra zorgt er ook voor dat de waarden in de lagen van de boom gerangschikt zijn per. Voor de tabel "NumberOfTwitterMessages" (6.1) kan dit er dan zoals in figuur 6.1 uitzien.



Figuur 6.1: Mogelijke fysieke structuur binnen een partitie

Door gebruik te maken van deze boomstructuur kan Cassandra dus zeer snel de resultaten voor de query's ophalen eenmaal de juiste node en partitie bereikt zijn. Opnieuw is het datamodel verantwoordelijk voor de clusteringkolommen en hiermee verantwoordelijk hoe efficiënt de data opgehaald kan worden.

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 replicatiestrategie en de replicatiefactor meegegeven te worden. Na het aanmaken van de keyspace, dient men hierin de tabel waarin de data geïmporteerd zal worden, aan te maken.

Nu kunnen we door middel van het 'COPY' commando, dat binnen cql aangeboden wordt, data importeren in Cassandra (Cannon, 2012).

```
COPY tablename(column1 , column2 ..)
FROM 'path/to/file '
(WITH option1 = 'value' (AND option2 = 'true' AND ...))
```

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 vastloopt.

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 met sstableloader

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

Enkele belangrijke nadelen van deze manier van werken 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 en daarna doorgestuurd wordt naar de node waarop de data hoort te staan.

7.3 Importeren met cassandra-loader

Dit is een Java programma geschreven door Hess (2016). 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 verbinding te maken. Op deze manier kan het werk evenwichtig verdeeld worden. Dit laatste was één 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 het aantal records dat 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 dat hier een lineair verband bestaat tussen het aantal rijen en de tijd die nodig is om alle rijen te importeren. Dit leverde de verwachte resultaten op.

Een test met sstableloader werd niet uitgevoerd omdat daarvoor alle nodes online moeten zijn. In een systeem waar alles ingezet wordt op hoge beschikbaarheid, is het dan ook niet aangeraden om een methode te gebruiken waar een import sowieso crasht als er een node offline gaat.

Het gebruik van cassandra-loader leverde veel problemen op. Dit kwam deels door virtuele machines te gebruiken. De cassandra-loader verwacht namelijk 8 GB werkgeheugen. Op de virtuele machine was een dergelijke hoeveelheid werkgeheugen echter niet voorhanden. Hierdoor moest het originele script lichtjes aangepast worden. Na de aanpassing van het virtueel geheugen dat toegewezen wordt aan de Java virtuele machine, crashte het programma.

Doordat het inladen met cassandra-loader niet werkte kan er dus ook geen vergelijking tussen beide methodes gemaakt worden. Er dient opgemerkt te worden dat beide manieren (cassandra-loader en importeren via cqlsh) in theorie niet zoveel van elkaar verschillen. Het enige pluspunt dat cassandra-loader heeft, is dat deze tool het werk evenwichtig over alle online nodes gaat verdelen.

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 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 te bepalen wanneer deze weer online is.

8.2 Herstelmechanisme

Cassandra voorziet drie ingebouwde herstelmechanismes (Strickland, 2014):

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 een node te herstellen na het uitvallen, te minimaliseren. Hierbij wordt er wat leesconsistentie opgeofferd om de schrijfbeschikbaarheid te verhogen.

In het geval een node offline is, houdt een andere gezonde node een hint bij als er data geschreven wordt. Wanneer alle nodes in het slechtste geval offline zouden zijn, houdt de coördinator deze hint bij. Als de node dan online komt, wordt er eerst voor gezorgd dat deze writes uitgevoerd worden en kunnen 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. Deze limiet bestaat om te voorkomen dat deze hint-wachtlIJst niet te lang wordt. Na deze periode van drie uur is het nodig om handmatig een herstel te doen om zo de consistentie te garanderen (Strickland, 2014).

8.2.2 Anti-entropy repair

Dit algoritme staat in voor de synchronisatie van replica's en zorgt ervoor dat deze up-to-date zijn op alle nodes. Dit proces gebeurt asynchroon. Binnen anti-entropy repair wordt 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 continu gecontroleerd. Anti-entropie betekent 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 met elkaar in verbinding. Wanneer data opgehaald wordt, bestaat er een kans dat deze vergeleken wordt met de andere replica's. Wanneer hier verschillen opgemerkt worden, 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 betekent 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 deze 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 standaardwaarde 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 meegegeven 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 bepaalt de garbage collection dan of het record effectief verwijderd mag worden (Strickland, 2014).

8.3 Ondervindingen bij het uitzetten van een node

Om het gedrag bij het uitvallen van een node te verifiëren, werd gebruik gemaakt van de cluster die eerder opgezet was (5). In deze cluster zijn vier nodes aanwezig. Om het gedrag te testen werd hier een keyspace aangemaakt met replicatiefactor vier. Zo zou de data op iedere node aanwezig horen te zijn. De replicatiestrategie 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 verspreid was, wat in OpsCenter waargenomen kon worden. Toen alle replica's aangemaakt waren, kon het testen beginnen.

Door de uitgekozen strategie en replicatiefactor zou op iedere node een replica van de data aanwezig moeten zijn. Om dit na te gaan, werden alle nodes alleen opgestart en daarna werd gekeken of er een specifieke 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. Dit toont aan dat de replicatie van de data door Cassandra uitstekend werkt.

Om de consistentie van Cassandra na te gaan werd een gelijkaardig stappenplan gevolgd. Hierbij werd telkens één node van de cluster opgestart, terwijl de andere nodes

offline waren. Op deze ene node werden dan een aantal records geüpdatet. Dit werd herhaald voor alle nodes van de cluster. Vervolgens werden alle nodes opgestart en werd er gewacht tot er geen activiteit meer te zien was in het dashboard van OpsCenter. Toen dit proces voor alle nodes voltooid was, werd nagegaan welke versie van de records opgehaald werd. Hier werd voor alle aangepaste records de geüpdatete versie teruggegeven, net zoals door de ontwikkelaars van Cassandra beloofd is.

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 enkele keyspace of één enkele tabel een snapshot nemen, zolang Cassandra nodes heeft die online zijn. Ook van een ganse cluster kan een snapshot genomen worden, maar hier kan het wel voorkomen dat de data in verschillende nodes niet consistent is. De snapshots worden telkens in een nieuwe file opgeslagen, 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 zal worden. 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 de verwijdering van oude, overbodige records (DataStax, 2016a). Snapshots verhinderen dus dat data die gemarkeerd is met een "tombstone marker" verwijderd wordt. Deze markering werd reeds in het vorige hoofdstuk (8.2.3) vermeld bij de herstelmechanismes.

Om een snapshot te gebruiken bij de herstelling van een database, moet men er eerst voor zorgen dat de tabellen die hersteld moeten worden, aangemaakt en ook leeg zijn. Daarna kan men de snapshot inladen met de sstableloader tool. Hierna sluit men de node die hersteld moet worden af en voert men vervolgens het commando "nodetool drain" uit. Vervolgens moet de commitlog map leeggemaakt worden. Een andere

map die leeggemaakt moet worden, met uitzondering van de submappen snapshots en backups, is de data-map van de keyspace. Hierna zoekt men de meest recente snapshot en plaatst deze in de keyspace map. Wanneer er incrementele back-ups genomen zijn, moet men deze ook in deze map plaatsen. Als laatste stap start men de node opnieuw op en voert men het commando "nodetool repair" uit. Op deze manier is de node hersteld aan de hand van 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. Vóór deze vraag beantwoord kan worden, moet het verschil tussen replica's en snapshots eerst duidelijk 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 bijgehouden. Verder wordt er ook voor gezorgd dat de verwijderde records niet echt verwijderd worden (8.2.3). Op deze manier wordt gegarandeerd dat de snapshots volledig blijven. Door deze manier van werken, zijn de snapshots binnen Cassandra dan ook een volwaardige back-up.

Bij replica's wordt vaak foutief aangenomen dat deze een synoniem zijn voor back-ups. Zoals eerder uitgelegd (3.2) werd, worden replica's op een andere node opgeslagen om 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. Indien hier een wijziging of verwijdering gebeurt, wordt dit onmiddellijk doorgegeven aan de andere nodes. Wanneer deze handeling foutief uitgevoerd werd, 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 er binnen Cassandra nog altijd back-ups nodig zijn. Hierbij gaat het dan vooral om verlies van data, die foutief gewijzigd of verwijderd wordt, tegen te gaan. Deze kunnen niet meer hersteld worden door middel van replica's.

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).

De belangrijkste gebruikers binnen deze use case zijn Instagram, New York Times, ComCast. . .

10.2 Fraudebestrijding

Cassandra biedt real time monitoring aan. Hiervoor kan nodetool of andere tools zoals DataStax' OpsCenter gebruikt worden. Op deze manier kan verdacht gedrag snel opgespoord worden. Toch hoeft de fraudebestrijding 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 op deze manier gemodelleerd zodat alle berichten met dezelfde inhoud in één rij terechtkomen. Door een simpele count uit te voeren op de rijen kan men nagaan hoe vaak eenzelfde bericht verstuurd geworden is. Wanneer de count een bepaald getal overschrijdt, wordt het bericht als spam beschouwd.

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

10.3 Productcatalogi en afspeellijsten

Door de enorme schaalbaarheid, de hoge performantie en de hoge beschikbaarheid is Cassandra uitstekend geschikt om productcatalogi en afspeellijsten bij te houden. Zowel productcatalogi 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 sensorgegevens. Het is dan ook vooral deze eigenschap die vaak de doorslag geeft om voor Cassandra te kiezen als men in deze use case terecht komt.

De bekendste gebruikers binnen deze use case zijn Aeris, i2O, CERN, NASA. . .

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

10.5 Aanbevelingen en personalisatie

Het grote probleem bij aanbevelingen is dat de data persoonlijk afgesteld moet worden en dat een grote hoeveelheid data tegelijkertijd verwerkt moet worden. Dit is echter geen probleem voor Cassandra want hiervoor werd de database geoptimaliseerd.

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

Ook binnen Unity wordt Cassandra gebruikt. Mäkinen (2015) legt uit dat hiervoor overgestapt werd van MongoDB naar Cassandra. Dit was omdat er nood 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, hierdoor kunnen game developers advertenties toevoegen aan hun games. Dit wil zeggen: Wie ziet welke advertentie op welk moment? 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, gebruikt men dan om te bepalen welke advertentie aan welke gebruiker getoond wordt.

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 productcatalogus gemaakt kan worden, kan een boodschappenwagentje gecreëerd worden. Doordat Cassandra ook uitstekend geschikt is voor messaging, is het mogelijk om dit in te zetten bij notificaties voor de eindgebruiker. Met al deze zaken in het achterhoofd 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 bijhouden van markttransacties te denken. Deze markttransacties hebben niets met transacties binnen een SQL omgeving te maken. Cassandra biedt ook uitstekende mogelijkheden om fraude tegen te gaan wat toch ook een groot pluspunt is in deze sector. Cassandra heeft op deze markt al enkele gebruikers zoals ING, UBS, Xoom... Een relatief nieuwe trend is dat verzekeraars, zoals Corona en AXA, bijvoorbeeld sensoren in een auto laten plaatsen en dat op basis van deze sensorgegevens een verzekeringspolis opgesteld wordt. Zoals hierboven vermeld, is Cassandra ook voor deze toepassing uitstekend geschikt.

De hierboven vermelde markten zijn diegene waarvoor Cassandra bijna kant-en-klare oplossingen aanbiedt. Natuurlijk kan Cassandra ook voor andere markten gebruikt worden, maar hier zal waarschijnlijk meer aandacht besteed moeten worden aan het datamodel en zal er met andere technologieën samengewerkt moeten worden.

Hoofdstuk 11

Conclusie

De schaalbaarheid van Cassandra werd nagegaan door nodes toe te voegen aan de cluster en vervolgens weer te verwijderen. Om dit te doen via de configuratie files van Cassandra is dit een heel karwei. 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. Op deze manier 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 aan de hand van het uitschakelen van de nodes om zo af te toetsen of de theorie wel strookt met de werkelijkheid. Bij het 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, uitschakelde, kon men toch de geüpdatete data terugvinden op de node die juist terug online kwam. Met de proof of concept die in deze bachelorproef opgezet werd, kan besloten worden dat Cassandra geen last heeft van een "single point of failure".

Bij de data modellering van Cassandra zijn er toch een aantal eigenaardigheden als men vanuit een SQL omgeving komt. Zo is men niet vertrouwd met het principe van het modelleren van de data naar de query's die uitgevoerd zullen worden. 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 hiervoor geen volwaardige vervangers. Door het gebruik van replica's wordt de beschikbaarheid van de data gegarandeerd, maar dit biedt geen garantie tegen corrupte data.

Eén opmerking die bij dit alles gemaakt moet worden is dat alle testen op virtuele machines werden uitgevoerd en daarbij 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 alleen zijn voorbehouden voor Cassandra.

Binnen het CAP theorema ligt de focus bij Cassandra vooral op beschikbaarheid en partitionering. Hier slaagt Cassandra zeer goed in. Maar zelfs 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 kan 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.
- Alvarado, G. (2014). Nosql and its use in ceilometer. <https://changeaas.com/2014/08/20/nosql-and-its-use-in-ceilometer/>. Geraadpleegd op 2016-05-22.
- 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). How data is distributed across a cluster (using virtual nodes). https://docs.datastax.com/en/cassandra/2.1/cassandra/architecture/architectureDataDistributeDistribute_c.html. Geraadpleegd op 2016-05-22.
- DataStax (2016c). 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.
- Hess, B. (2016). Cassandra-loader. <https://github.com/brianmhess/cassandra-loader>. Geraadpleegd op 2016-03-04.
- 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.
- Lampe, J. (2010). Introduction to cassandra columns, super columns and rows. <http://www.divconq.com/2010/cassandra-columns-and-supercolumns-and-rows/>. Geraadpleegd op 2016-05-24.
- 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.
- PlanetCassandra (2013). Rapid read protection in cassandra 2.0.2. <http://www.planetcassandra.org/blog/rapid-read-protection-in-cassandra-202/>. Geraadpleegd op 2016-05-24.
- 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

1.1	CAP theorema (Alvarado, 2014)	5
3.1	Illustratie partitioner (DataStax, 2016b)	11
3.2	Illustratie SimpleStrategy (Strickland, 2014)	13
3.3	Illustratie werking snitch (PlanetCassandra, 2013)	14
4.1	Kolommen in Cassandra (Lampe, 2010)	16
4.2	Logische datastructuren in Cassandra	18
5.1	Cassandra: Instellingen deel 1	20
5.2	Cassandra: Instellingen deel 2	21
5.3	Installatie van Cassandra door OpsCenter	22
5.4	Rondleiding in OpsCenter	22
5.5	nodetool	23
5.6	Toevoegen van een node via OpsCenter	24
5.7	Verwijderen van een node: Selecteer de node	25
5.8	Verwijderen van een node: Selecteer de node	25
5.9	Verwijderen van een node: Selecteer de node	26
6.1	Mogelijke fysieke structuur binnen een partitie	34

Lijst van tabellen

5.1	Configuratie van de Cassandra Cluster	20
7.1	Importeren van data met cqlsh	37

Lijst van Code fragmenten

6.1	Layout van de tabel NumberOfTwitterMessages	29
6.2	Foutieve query bij meerdere partitiekolommen	29
6.3	Data opslag binnen een partitie	30
6.4	Multi-column slice restrictie	30
6.5	De restrictie moet starten met dezelfde kolom	30
6.6	Correcte modellering bij Cassandra	32
6.7	Foutieve modellering bij Cassandra	32