KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Automatisch uitrol van database systemen en vergelijking van beschikbaarheid

Thomas Uyttendaele

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Gedistribueerde systemen

Promotor:
Prof. dr. ir. Wouter Joosen

Assessor:
Ir.

Begeleider:
Ir. B. Vanbrabant

Academiejaar 2013 – 2014

Voorwoord

Voorwoord schrijven

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Thomas Uyttendaele

Inhoudsopgave

Todo list

Samenvatting

Abstract schrijven

In dit `abstract` environment wordt een al dan niet uitgebreide samenvatting van het werk gegeven. De bedoeling is wel dat dit tot 1 bladzijde beperkt blijft. Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen en symbolen

Afkortingen

IMP	Infrastructure Management Platform

Symbolen

42	aaa

Hoofdstuk 1

Inleiding

In dit hoofdstuk wordt het werk ingeleid. Het doel wordt gedefinieerd en er wordt uitgelegd wat de te volgen weg is (beter bekend als de rode draad).

Schrijf inleiding

Hoofdstuk 2

Probleem- en doelstelling

2.1 Probleemstelling

Sinds WEB 2.0 is het gebruik van het internet veranderd, van een statisch tot een dynamische omgeving waar vele gebruikers een persoonlijke en individuele ervaring krijgen. Samen met deze evolutie is ook het type, de hoeveelheid en het gebruik van data en de opslag ervan veranderd.

Deze verandering kan toegeschreven worden aan de toename in het aantal website, daarnaast is ook de content van een webpagina veranderd, geen statische pagina's maar pagina's aangepast aan de bezoeker. Deze content wordt ook niet meer enkel door de websitebeheerder online geplaatst maar de gebruikers kunnen zelf hun data invoegen en krijgen gepersonaliseerde en dynamische pagina's te zien. Tenslotte is ook de toename in het aantal gebruikers, op 30 juni 2012 waren er meer als 2.4 miljard gebruikers van het internet, een toename van meer als 500% op 12 jaar tijd. [3]

Al deze veranderingen hebben ook hun doorslag op de achterliggende infrastructuur. Waar kleinere en statische websites nog konden gehost worden op een enkele server, is dit niet meer het geval voor vele websites. Een onderdeel van deze web applicaties is de databases waarin de vereiste data kan in worden opgeslagen worden en later opgevraagd worden. Veel van deze databases zijn ontworpen naar het systeem van de relationele databases, uitgewerkt naar Codd [2].

Voor talrijke van de huidige applicaties is de voorwaarden voor dataopslag verschillend met deze van de jaren 70. Enkele systemen hebben geen nood meer aan onmiddellijke consistentie maar eventuele consistentie volstaat. Onder andere omwille van deze reden is er een stroming van nieuwe database systemen gekomen genaamd NoSQL. Onder deze noemer vallen vele systemen, elk eigen in hun soort en aangepast aan bepaalde manieren van dataopslag en de daarvoor nodige vereisten, een belangrijk element is het horizontaal schaalbaar zijn van deze systemen. Met andere woorden, deze databases zijn ontworpen om gedistribueerd te werken zodat de belasting verdeeld kan worden.

Horizontaal schaalbaar brengt verschillende voordelen mee, maar zorgt ook voor 2 problemen die in deze thesis zullen aanbod komen, namelijke de uitrol en daarnaast wat de implicaties zijn van een gedistribueerd datasysteem.

2.1.1 Uitrollen van gedistribueerde databases

Voor geavanceerde toepassingen is het gebruik van gespecialiseerde en/of gedistribueerde databases een noodzaak. Maar het moment dat van start tot tot een werkende omgeving hebben, is niet te onderschatten en dit omwille van verschillende redenen.

Een eerste stap die doorlopen wordt, is het opzoeken van de nodige informatie over de verschillende databases, nu bestaan er al verschillende websites en papers waar informatie opgezocht kan worden, maar een consistente test bestaat niet. Een voorbeeld van een vergelijkingswebsite is bijvoorbeeld vsChart [4], ook verschillende papers brengen vergelijkingen naar voor in onder andere performantie en de verschillende data modellen. Maar in de meeste gevallen blijft het aangeraden om de informatie op te zoeken op de website van de software zelf, de software evolueert snel en informatie van een jaar oud kan al verouderd zijn en zo incorrect.

Al deze informatie zou de ontwikkelaar een mogelijkheid geven om de verschillende systemen te kunnen vergelijken en een keuze te maken welke database hij verkiest.

Na de selectie van een specifieke database, komt de keuze voor de opstelling van de database. Bepaalde databases hebben enkel ondersteuning voor Master-slave configuratie, bij andere zijn alle instanties gelijkaardig en nog andere hebben een combinatie van de twee. Deze verschillende manieren zorgt ervoor dat er geen eenduidige manier is om alle verschillende systemen consequent uit te rollen. De ontwikkelaar zal in deze stap een opstellen kiezen onder welke hij de database zal uitrollen.

Nu de databasesoftware en de opstelling is gekozen, kunnen de verschillende machines fysiek opgesteld worden. Dit wil zeggen, de opstellening wordt fysiek opgesteld: de computers kiezen, deze op de juiste locatie zetten en te verbinden. De meeste NoSQL systemen zijn ontworpen om te draaien op commodity hardware wat wil zeggen dat gewone consumenten computers volstaan.

In het volgend stadium, komt de softwarematige installatie: beginnende bij het besturingssysteem tot de database software zelf. De meeste van deze databases draaien onder verschillende Linux versies. De meeste hebben wel de nodige andere software nodig om ook te kunnen draaien, een voorbeeld is HBase dat Java en Hadoop nodig heeft.

Deze systemen worden geïnstalleerd en nadien geconfigureerd. Waar voor de installatie meestal nog standaard tools zoals yum of apt-get gebruikt kunnen worden, is de configuratie afhankelijk van systeem tot systeem. Bepaalde systemen zoals Pgpool werken volledig met configuratie bestanden, andere systemen zoals MongoDB

werken voornamelijk met een configuratie via de shell, nog andere. Deze verschillende aanpakken maken dat opzetten van verschillende systemen afhankelijk is van pakket tot pakket.

Nu het systeem draait, kan het systeem getest worden en bekeken worden of het voldoet aan de vooropgesteld benodigdheden. Indien dit niet het geval is, kan een aanpassing aan de opstelling misschien voldoen of dient een ander systeem geselecteerd worden.

Vervolgens kan de ontwerpen verder gaan met het ontwerpen van de applicatie, waar de databaselaag uiteindelijk maar een gedeelte van is.

Daarnaast zou het kunnen dat de set-up voldoet tijdens het ontwerpen maar niet tijdens de uiteindelijke ingebruikname of maakt de applicatie een groei mee waardoor er extra resources nodig zijn. Nu zijn veel van de databases en zeker de NoSQL databases geschreven om eenvoudig horizontaal schaalbaar te zijn: het toevoegen of verwijderen van systemen kan in veel gevallen zelfs dynamisch.

Deze stappen zijn allemaal niet zo onlogisch maar vragen wel tijd en kennis van de installatie en configuratie van deze systemen. Bij veel lokale systemen wordt de installatie en configuratie automatisch gedaan door onder andere gebruik te maken van yum of apt-get waarna deze systemen volledig werken, want uiteindelijk dient een ontwerper om nieuwe systemen te ontwikkelen, oa. door het samenbrengen van verschillende componenten die als een black block kunnen beschouwd worden met zo weinig mogelijk configuratie.

2.1.2 Implicatie van gedistribueerde datasystemen

Een tweede element van de thesis is het onderzoek naar de implicaties van gedistribueerde datasystemen. In dit gebied is er al veel onderzoek gedaan naar de verschillen in datamodellen en performantie maar er is nog geen gerelateerd onderzoek naar de implicaties op de beschikbaarheid van de data.

De CAP-theorie van E. Brewer[1] staat voor consistency, availability en partition tolerance. De volgende definities zoals geformuleerd in [11]:

Consistency of consistentie betekent dat na een aanpassing van een schrijver in een gedeelte databron, alle lezers deze aanpassing zien.

Availability of beschikbaarheid, meer bepaald hoge beschikbaarheid, betekent dat een systeem de mogelijkheid heeft om verdere operaties (lezen en/of schrijven) te verwerken indien een of meerdere nodes in de cluster niet beschikbaar zijn.

Partition tolerance of partitie tolerantie betekent dat het systeem verder kan blijven werken in zijn geheel indien de volledige cluster (tijdelijke of finaal) in twee of meer delen opgedeeld is. Een andere betekenis is dat er de mogelijkheid is tot het dynamisch verwijderen of toevoegen van nodes, waarbij de te verwijderen of toe te voegen nodes bezien worden als een eigen netwerk partitie.

Vervolgens stelt E. Brewer dat in een gedistribueerd systeem maximaal aan 2 van de 3 elementen van de CAP-theorie kan voldoen.

Dit heeft invloed op de beschikbaarheid van de data in gedistribueerde databases, maar verschillende database systemen maken verschillende keuzes. Bepaalde database systemen geven geen strikte consistentie maar enkel eventuele consistentie waarbij de aanpassing na verloop van tijd maar beschikbaar is voor alle gebruikers. Indien een ontwerper hier geen rekening bij houdt bij de implementatie, kunnen in reservatiesysteem bijvoorbeeld dubbele boekingen voorkomen.

2.2 Doelstelling

De doelstellingen voor deze thesis bestaat uit 2 delen, net zoals de probleemstelling. In eerste instantie de uitrol van gedistribueerde databases, in tweede instantie de implicaties op de beschikbaarheid van de data.

2.2.1 Uitrol van gedistribueerde databases

Het eerste gedeelte van de doelstellingen is gerelateerd met de uitrol van gedistribueerde databases, voornamelijk het vereenvoudigen van de uitrol.

Net zoals yum en apt-get de mogelijkheid geven om lokaal op een eenvoudige manier software te installeren, zal in deze thesis de uitrol van verschillende database systemen geautomatiseerd worden en dit door middel van IDP.

Aangezien er geen unieke opstelling is, zal er nog steeds een configuratie nodig zijn. Maar het doel is om deze configuratie eenvoudiger te maken op basis van een model.

De gebruiker zal enkel een systeem moeten selecteren, de set-up bepalen en de systemen fysiek opstellen. De softwarematige installatie en configuratie zal niet langer door gebruiker moeten gebeuren maar bepaalde componenten toewijzen aan bepaalde machines, met de nodige relaties tussen de componenten.

2.2.2 Implicatie op beschikbaarheid van gedistribueerde datasystemen

Controleren na implementatie fase 2

Naast de automatisatie van de uitrol, zal er ook gefocust worden op het testen van deze systemen. Dit niet op basis van de performantie maar dit op de beschikbaarheid van de data.

Zoals vermeldt in de probleemstelling, hebben verschillende database systemen een ander methode om om te gaan met het niet beschikbaar zijn van een node en netwerk partities.

Om deze verschillen te vergelijken zullen er testen geschreven worden om de verschillende keuzes en hun implicaties duidelijk te maken.

Tenslotte zullen deze testen ook beschikbaar gesteld worden om op een eenvoudige manier ook op andere systemen uitgerold te worden en verificatie van de data mogelijk te maken. Daarnaast is het zo ook mogelijk om deze testen opnieuw uit te voeren indien er updates beschikbaar komen, die mogelijke verbeteringen brengen.

Hoofdstuk 3

Beschrijving van verschillende databases en selectie van databases

Vele NoSQL databases zijn ontwikkeld vanuit de industrie om onmiddellijk te kunnen inzetten op hun problemen, elk met hun eigen specificaties rondom performantie, schaalbaarheid, complexiteit en functionaliteit. Waar er bij de relationele databases verschillen waren in functionaliteit in beperkt zijn ten opzicht van NoSQL databases.

Onder andere Ben Scofield[11] heeft de verschillende NoSQL databases opgesplitst in 4 categorieën , met daarnaast de traditionele relationele databases waar categorie heeft een eigen invulling van het datamodel. De opsplitsing bevindt zich in tabel 3.1.

Soort	**Performantie**	**Schaalbaarheid**	**Flexibiliteit**	**Complexiteit**	**Functionaliteit**
Key-Value	hoog	hoog	hoog	geen	variabel (geen)
Column	hoog	hoog	gematigd	laag	minimaal
Document	hoog	variabel(hoog)	hoog	laag	variabel (laag)
Graph	variabel	variabel	hoog	hoog	graph theory
Relational	variabel	variabel	laag	gematigd	relational algebra

Tabel 3.1: Classificatie en categorisatie van databases door Scofield en Popescu. [9] [6]

Voor het tweede deel van deze thesis, het vergelijken van de verschillende gedistribueerde systemen met betrekking tot het beschikbaar zijn van de data, hebben de systemen bepaalde vereisten nodig. Allereerst moet het mogelijk zijn om het systeem gedistribueerd uit te rollen waarbij data verdeeld wordt over verschillende instanties, zodat de data liefst nog bereikbaar is wanneer een bepaald instantie onbereikbaar is.

Een ander element is dat de systemen vrij te installeren moeten zijn, bepaalde databases zoals Google's Bigtable worden beschreven in papers maar zijn enkel als service op hun systemen. Tenslotte moet de data ook persistent zijn.

Andere interessante elementen, is het automatisch detecteren als een instantie niet meer bereikbaar is en daarna ook terug synchroniseren als een instantie online

komt.

Verder zijn er nog andere differentiatie mogelijkheden, een beperkte query set, al dan niet automatische conflict resolutie, load-balancing, ...

In de volgende delen zullen de verschillende categorieën aanbod komen met enkele van de systemen die dit implementeren. De eerste selectie van systemen is gebaseerd op de systemen besproken door Christof Strauch [11].

3.1 Column database

In een column-gebaseerd systeem wordt de data opgeslagen per kolom in plaats van de traditionele manier, per rij. Deze aanpak werd in eerste instantie gedaan voor analyse van business intelligence.

Google's Bigtable is een van de databases gebaseerd op deze technologie, maar dit systeem is niet vrij te installeren. Maar 2 systemen die geïnstalleerd kunnen worden zijn Cassandra en HBase.

3.1.1 Cassandra

Website: *http://cassandra.apache.org/*
Cassandra is een database die gebaseerd is op 2 verschillende systemen, Amazon's Dynamo en Google's Bigtable, wat voor een combinatie van een column- en key-value-based database zorgt.

De query taal is beperkt tot 3 operaties: get, insert en delete [5], waar de laatste waarde in geval van een conflict zal opgeslagen worden.

De database kan gedistribueerd uitgerold worden waar door middel van partitionering en een consistent hashing algoritme de data verspreid wordt over de verschillende instanties. Om beschikbaarheid van de data te hebben bij een failure, wordt deze gerepliceerd over verschillende instanties met verschillende configuratie modellen.

3.1.2 HBase

Website: *http://hbase.apache.org/*
HBase is een database die gebaseerd is op Google's BigTable en draait boven op HDFS, Hadoop Distibuted File System.

De query taal voor HBase bestaat uit 4 elementen, een get, put en delete als standaard operaties en een scan om over verschillende rijen te gaan.

Voor het gedistribueerd draaien van de database, wordt de database ingedeeld in regio's. Vervolgens is een verantwoordelijk voor regio's Regionserver om de data van regio's op te slaan. Daarnaast is er ook nog afhankelijkheid van Zookeeper voor management van regio's en name en data-instanties als opslag.

3.2 Document database

Document databases zijn volgens vele de volgende stap in key-value databases, waar deze complexere structuren toe laten, dit door middel van meerdere key/value paren per element. Een document moet geen vaste structuur hebben maar elk document op zich kan verschillende velden hebben, dit kan bijvoorbeeld gezien worden bij boken. Waar een bepaald boek een recept is, kan een ander een deel zijn van een trilogie. Bij het eerste kan de kooktijd bijvoorbeeld apart opgeslagen worden en bij de tweede de andere boeken.

Enkele systemen die deze structuur implementeren zijn CoucheDB, Elastic Search en MongoDB.

3.2.1 Apache CoucheDB

Website: *http://couchdb.apache.org/*
Apache CoucheDB is een document database waar alles wordt voorgesteld met behulp van JSON. De database kan bevraagd worden door middel van Map-Reduce, de map gebeurd door een *view*, een JavaScript-functie die de gegevens zal selecteren. Nadien kan met een reduce view de data geaggregeerd worden.

Bij het gedistribueerd uitrollen zal de data met consistent hashing over verschillende instanties verdeeld worden waar elke instantie een zelfde rol heeft. Nu zal CoucheDB enkel updates van data van instantie veranderen en niet data automatisch load-balancen. Ook is het mogelijk om een exacte replica van de ene naar de andere instantie te sturen, dit wordt bijvoorbeeld handig indien documenten naar een laptop gesynchroniseerd worden om later offline te kunnen werken.

In een gedistribueerde omgeving ziet CouchDB conflicten niet als een exceptie maar als een normale omstandigheid. Wel zullen updates atomisch per rij afgewerkt worden op een enkele instantie, zodat hier geen conflict in kan bestaan. Maar indien een conflict optreedt, is het aan de bovenliggende applicatie om deze af te handelen.

3.2.2 MongoDB

Website: *http://www.mongodb.org/*
MongoDB is een document database waar de data wordt voorgesteld aan de hand van BSON, een binaire vorm vergelijkbaar met JSON. Data kan ingegegeven worden via JSON aangezien er een eenvoudige map mogelijk is.

Er is een uitgebreide query taal, waar er naast het invoegen, verwijderen en opvragen van een document ook talrijke zoekparameters meegegeven kunnen worden: dit gaat van zoeken op een enkel veld tot conjuncties, sorteren, projecties, ...

MongoDB kan in een gedistribueerde omgeving opgezet worden met een opsplitsing tussen het redundant opslaan van data en het verdelen van data. Het redundant opslaan kan gedaan worden door het combineren van instanties in een ReplicaSet waar er een master-slave configuratie is. Daarnaast kan data ook verdeeld worden over verschillende instanties of replica sets, dit kan door middel van het

configureren van shards. Conflicts worden opgevangen door de master waar er telkens een meerderheid van de instanties nodig is voor de data.

3.3 Graph database

Graph databases slaan data op door middel van knopen, lijnen en eigenschappen op beiden.

Deze opslag structuur is significant verschillend van de andere besproken technieken. De data die opgeslagen wordt in een graaf is van een andere type en om deze reden wordt deze categorie niet verder besproken.

3.4 Key-Value database

Key-value databases hebben een heel eenvoudig data model, data kan opgeslaan, opgevraagd en verwijderd worden op basis van een key. De informatie die in de database zit, is de waarde voor die key.

Met dit eenvoudig model en functionaliteit die weinig complexiteit introduceren, kan er gestreefd worden naar een hoge performatie, schaalbaarheid en flexibiliteit.

In deze categorie zullen er 5 verschillende databases besproken worden: LightCloud, MemCach, Redis, Riak en Voldemort.

3.4.1 LightCloud (Tokyo)

Website: *http://opensource.plurk.com/LightCloud/*
LightCloud is een gedistribueerde uitbreiding van Tokyo Tyrant. Tokyo Tyrant is op zijn beurt een uitbreiding op Tokyo Cabinet en voegt de mogelijkheid tot externe connecties aan Cabinet toe. Cabinet is het basis pakket.

De query taal is gelimiteerd tot 5 operaties: get, put, delete, add en een iterator om over de keys te gaan. Met add wordt er data aan een bestaand element toegevoegd.

LightCloud levert een gedistribueerde database met master-master synchronisatie. Met behulp van een consistent hashing algoritme en 2 hash rings, wordt de data verdeeld over verschillende instanties met de nodige redundantie. De eerste ring is verantwoordelijk voor de lookups oftewel het lokaliseren van de keys, de storage ring is verantwoordelijk voor het opslaan van de verschillende waarden.

3.4.2 MemCache

Website: *http://memcached.org/*
MemCache is een veel gebruikt systeem waarin al de data in RAM geheugen wordt gehouden en alhoewel er ondersteuning is met behulp van MemCacheDB voor persistentie is deze database niet bedoeld voor persistente opslag.

Omdat één van de vereisten was dat de data persistent moest zijn, is dit systeem niet verder onderzocht in deze context.

3.4.3 Redis

Website: http://www.redis.io/
Redis is een key-value database met de mogelijkheid tot opslaan van complexe datastructuren zoals lijsten, sets en mappen. Naast de standaard instructies om een enkele waarde toe te voegen, zijn er specifieke commando's om operaties op de complexere objecten te doen, het verwijderen van een lid van een bepaalde set. Redis biedt ook ondersteuning voor transacties en heeft deze de mogelijkheid tot expire, hierdoor zal een waarde automatisch vergeten worden na een meegegeven tijd.

De database wordt volledig in geheugen geplaatst maar ondersteund 2 soorten van persistentie, oftewel door middel van RDB, oftewel met een AOF log. Bij RDB worden er over tijd snapshots gemaakt van de database en weggeschreven op harde schijf. In het geval van AOF wordt elke schrijfoperaties weggeschreven en kan de database opgebouwd worden met behulp van deze lijst.

Tenslotte heeft Redis momenteel een relatief beperkte mogelijkheid tot een gedistribueerde database. Het is mogelijk om data over verschillende instanties te distribueren met behulp van sharding welke op voorhand gedefinieerd dient te worden en is er ook de mogelijkheid tot master-slave opstelling met automatische failure detection. De laatste is nog wel in beta al is het al mogelijk om te gebruiken. Tenslotte is er naar de toekomst ook meer uitbreiding op komst met behulp van Redis Cluster waar data automatisch verspreid wordt over verschillende instanties.

3.4.4 Riak

Website: http://basho.com/riak/
Riak is een key-value database met de mogelijkheid tot opslaan van strings, JSON en XML. Daarnaast heeft deze standaard operaties maar hier enkele uitbreidingen op gemaakt. Allereerst is het mogelijk om secundaire indexen te definiëren op de elementen, MapReduce toe te passen en een full-text search.

Riak is gebouwd om gedistribueerd te draaien waar al de instanties evenwaardig zijn. Data wordt verdeeld over de verschillende instanties en elk element wordt standaard op 3 verschillende instanties opgeslagen. Indien een bepaalde instantie faalt, wordt dit met een gossiping algoritme verspreid over de verschillende instanties waarmee een naburige instantie overneemt. Daarnaast is er automatische recovery indien een instantie terug online komt.

3.4.5 Project Voldemort

Website: http://www.project-voldemort.com/
Project Voldemort is een key-value store met enkel 3 basis operaties: get, put en delete met de mogelijkheid voor als keys en values strings, serializable objecten, protocol buffers of raw byte arrays te gebruiken.

Deze database ondersteunt verschillende modes van distributie. De opbouw bestaat uit verschillende lagen, elk met hun eigen gedefinieerde functie. Met behulp van deze lagen kan de ontwikkelaar extra functionaliteit toevoegen met behulp van een extra laag om de applicatie meer te finetunen naar zijn uitwerking. Data wordt

verdeeld met behulp van consistent hashing over de verschillende servers, waarbij data verschillende keren wordt bijgehouden om ervoor te zorgen dat de data nog beschikbaar is in het geval van falen.

3.5 Relationeel database

Relationele databases zijn want men noemt de traditionele databases: verschillende tabellen waar elk data element een rij van attributen heeft. Daarnaast hebben deze databases een uitgebreide query taal: verschillende tabellen kunnen gecombineerd worden, ...

Deze systemen waren in eerste instantie niet ontworpen om gedistribueerd te draaien, sommige systemen ondersteunen al bepaalde vormen, anderen hebben hebben nog andere pakketten nodig.

3.5.1 MySQL

Website: http://www.mysql.com/
MySQL is een relationele database waarin data kan voorgesteld worden in verschillende vormen, beginnend met een bool tot een blok tekst. Daarnaast zijn de query mogelijkheden uitgebreid.

De uitbreiding van een gedistribueerd systeem is bij MySQL ingebouwd door middel van een Master-Slave configuratie. Als mysqlfailover een faal detecteert in één van de slaven, zal de database verder werken, bij het falen van de master zal een nieuwe master handmatig aangeduid moeten worden. Ook de recovery moet handmatig gestart worden, waarna indien gewenst de originele master opnieuw als master kan gezet worden (bv. omdat deze de krachtigste computer is).

3.5.2 Pgpool-II(PostgreSQL)

Website: http://www.pgpool.net/
PostgreSQL is een relationele database en heeft soortgelijke specificaties als MySQL op een enkele computer, verschillende soorten data kunnen voorgesteld worden met uitgebreide query mogelijkheden.

Enkel als de database ook gedistribueerd moet uitgerold worden, is er een verschil. Bij PostgreSQL is er hiervoor standaard geen ondersteuning hiervoor maar moet er op externe elementen vertrouwd worden. Er bestaan verschillende componenten soorten systemen, maar aangezien er load-balancing nodig is en een zo recent mogelijk pakket, is er gekozen voor Pgpool-II, een vergelijking van de systemen kan gevonden worden op de wiki van PostgreSQL [7].

Pgpool-II heeft verschillende mode, maar aangezien er de nood was voor replicatie, is er gekozen om de replicatie mode in te stellen. Hierdoor komt de database in een Master-Slave situatie waar er het falen gedetecteerd wordt op het moment een connectie actief is. Indien een instantie terug online is, moet er manueel het recovery process opgestart worden.

Andere configuraties zijn mogelijk, onder andere met het opzetten van caching en nog andere elementen.

3.6 Selectie van de databases

In deze thesis zullen er 3 databases automatisch geïnstalleerd en getest worden. Voor deze keuze is er gekeken naar een zo groot mogelijke variatie in de verschillende systemen.

Deze selectie was in samenspraak met de andere thesis rondom dit onderwerp. Zo zijn er uiteindelijk 6 systemen uitgewerkt, Riak als key-value, Cassandra en HBase als column, MongoDB als document en MySQL en Pgpool-II (PostgrSQL) uit de relationele categorie.

Riak is gekozen vanwege de relatief grote query taal in de key-value categorie.

Cassandra is gekozen vanwege zijn interessante combinatie van key-value en column categorie.

HBase is gekozen vanwege zijn grote gelijkenis tot Google's BigTable.

MongoDB is van de document categorie gekozen omdat deze een grote configuratie mogelijkheid bood en tegelijk grondige online documentatie had.

MySQL en **Pgpool-II (PostgrSQL)** zijn beide gekozen om een vergelijking te hebben ten opzichte van de meer traditionele databases.

De databases die verder in deze thesis besproken zullen worden zijn MongoDB, HBase en Pgpool-II (PostgrSQL).

Hoofdstuk 4

Automatisch uitrollen van databases

4.1 Inleiding

4.2 IMP: Framework voor installatie

Te schrijven

4.3 MongoDB

In deze sectie zal

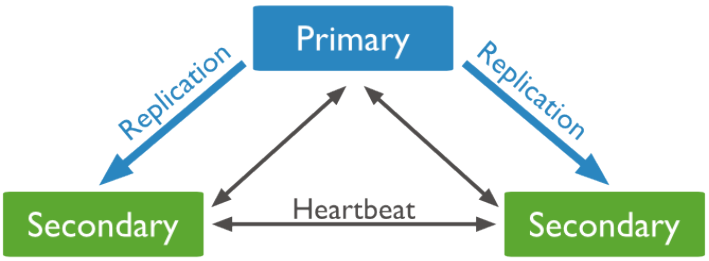
4.3.1 Werking van MongoDB

MongoDB heeft gekozen om bij gedistribueerde database op te splitsen in 2 keuzes, een eerste is door middel van *replica set*s en daarnaast door middel van *shard*ing.

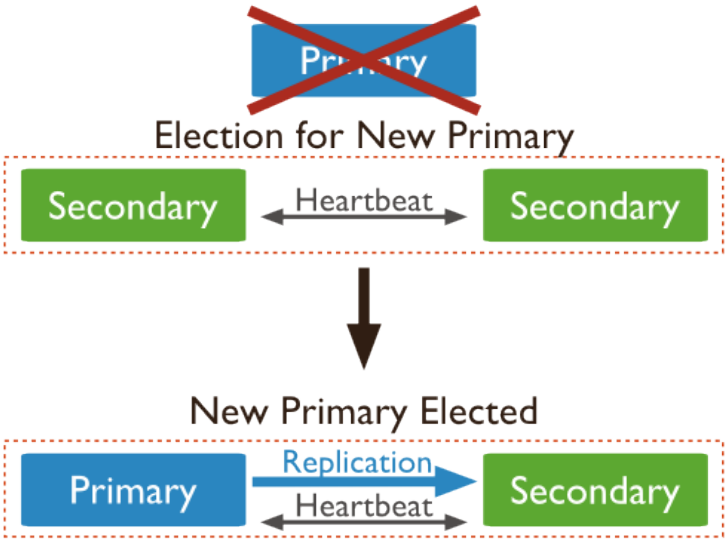
Bij *replica set* is een master-slave configuratie van de MongoDB nodes, met de master benoemd als primary en de slaves als secondary (figuur 4.1). De data is nog steeds beschikbaar zolang meer als de helft van de nodes beschikbaar zijn. De schrijfoperaties worden eerst naar de primary gestuurd en vervolgens door MongoDB naar de secondaries gesynchroniseerd. Ook leesoperaties worden standaard enkel naar de primary gestuurd maar dit kan aangepast worden in de configuratie.

Met behulp van een heartbeat houden wordt de status van de verschillende nodes opgevraagd elke 10 seconden en indien de primary niet meer beschikbaar is zal een secondary gestemd worden tot primary (figuur 4.2).

Vervolgens wordt met behulp van *sharding* de data verdeeld over verschillende shards. In productie omgeving wordt aangeraden om een replica set te nemen als shard maar het is ook mogelijk om een enkele node toe te voegen.



Figuur 4.1: MongoDB: Drie leden van een replica set met 1 master en 2 slaves. Bron: MongoDB[8]

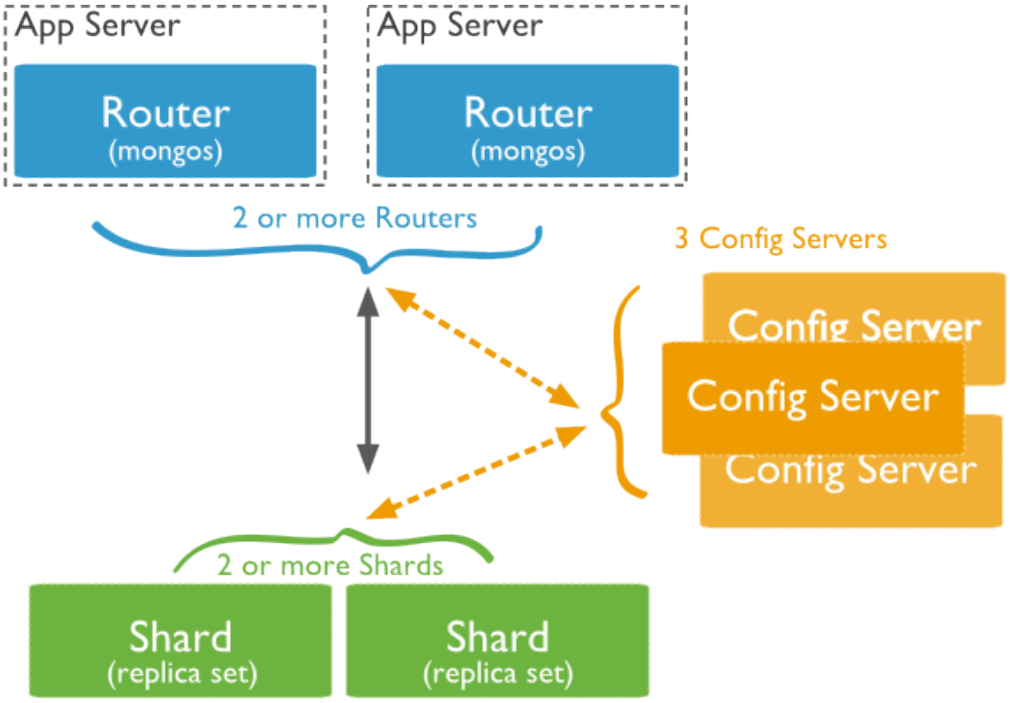


Figuur 4.2: MongoDB: Een voorbeeld van het verkiezen van een nieuwe primary bij het niet meer beschikbaar zijn van de vorige primary. Bron: MongoDB[8]

Voor lees- en schrijfoperaties wordt er verbinding gemaakt met mongos, dit zijn router nodes die de query naar de nodige shards stuurt. De configuratie van de shards wordt opgeslagen in 1 of 3 configuratie nodes (figuur 4.3).

De data wordt verdeeld over de verschillende shards met een door de gebruiker gedefinieerde formule, dit kan door middel van hashing of door het opsplitsen van een waarde in verschillende domeinen.

De configuratie van MongoDB is opgesplitst in 2 delen, een configuratiebestand en met behulp van de shell. Als eerste stelt men een configuratiebestand op die aan de instantie basisinformatie meegeeft zoals op welke poort, waar de database op het bestandssysteem op te slaan en welk type instantie het is (enkele node, deel van replica set, configuratie server of een mongos).



FIGUUR 4.3: MongoDB: Een voorbeeld cluster voor productie met 2 mongos, 3 shards en 3 configuratie servers. Bron: MongoDB[10]

Het opzetten van de replica sets en sharding verloopt via de shell, met behulp van een commando op een instantie die een deel is van de replica set kan de replica set opgezet worden. Sharding wordt opgezet door connectie te maken met een mongos en ook hier via commando shards toe te voegen, collecties aan te maken en collecties te verdelen onder shards.

Belangrijk bij sharding is dat bij het opstarten van een mongos al de verschillende configuratie servers meegegeven moeten worden, dit is de enige parameter die meegegeven moet worden in de configuratiebestanden die afhankelijk is van andere instanties.

4.3.2 Uitwerking

Voor de uitwerking van de installatie in IMP zijn er enkele aannames genomen:

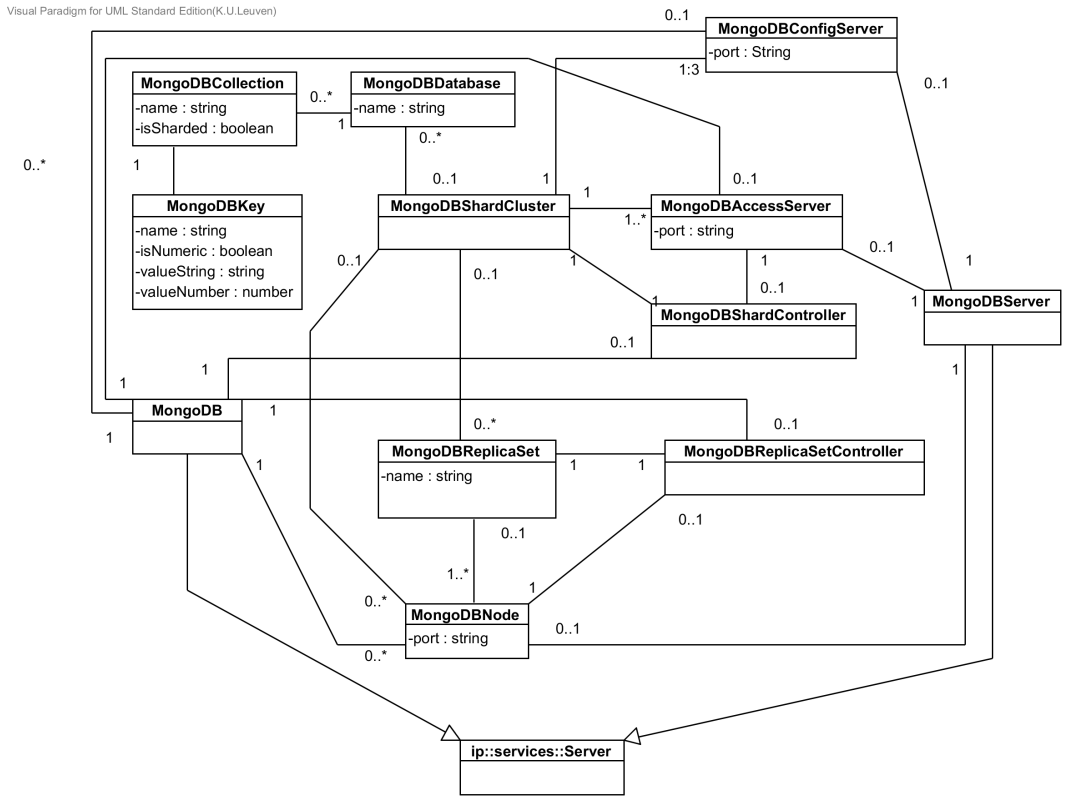
- Op elke server kan hoogstens **1** MongoDB data instantie draaien, ongeachte of deze een deel is van een replica set of een alleenstaand deel is. Dit is een gegronde aanname omdat meerdere instanties betekend dat de resources gedeeld moeten worden over meerdere servers. Een nadeel hieraan is dat elke server daardoor maar lid kan zijn van 1 replica set.

- Op elke server kan hoogstens **1** MongoDB config server instantie draaien. Indien er 2 configuratie instanties op de zelfde server zouden draaien, is er geen bescherming tegen het onbeschikbaar zijn van 1 server, aangezien 2 van de 3 instanties tegelijk uitvallen als deze server uitvalt.
- Op elke server kan hoogstens **1** mongos instantie. Meerdere instanties bieden enkel bescherming tegen het niet meer werken van het mongos proces, maar dit kan lokaal op de computer gecontroleerd worden en indien nodig kan de service herstart worden.
- Aanpassingen aan replica sets zullen afgewerkt worden met eerst het toevoegen van instanties en pas na daarna verwijderen van instantie, dit om ervoor te zorgen dat een set altijd zo veel mogelijk leden heeft.
- Uitbreidingen of aanpassingen aan de verdeling van een collectie over shards is niet mogelijk maar zal geen compilatiefout geven.

Met deze aannames is er tot het domeinmodel gekomen dat te vinden is in figuur 4.4. De verschillende entiteit hebben de volgende functie:

- MongoDB is een server in het IMP model en is verantwoordelijk voor het installeren van de basis van MongoDB. Hierna zijn basis commando's voor connectie te maken met een MongoDB instantie beschikbaar.
- MonogDBServer is een server in het IMP model en is verantwoordelijk voor het installeren van de MongoDB server.
- MongoDBNode is de implementatie van een data instantie, maximaal 1 per server. Indien gelinkt met een replica set zal deze als een deel van een replica set worden geïnitialiseerd, anders als een zelfstandige instantie.
- MongoDBReplicaSet is de voorstelling van een replica set, dit wordt niet aan een specifieke server toegewezen.
- MongoDBReplicaSetController is verantwoordelijk om de replica set te initialiseren. Belangrijk is dat indien er een uitbreiding is van de set, de node verbonden met de controller een reeds geïnitialiseerde node is.
- MongoDBConfigServer is de implementatie van een configuratie server, 1 of 3 servers zijn nodig per cluster.
- MongoDBAccessServer is de implementatie van mongos, minstens 1 is nodig maar meer kunnen gebruikt worden.
- MongoDBShardCluster is de voorstelling van een cluster van shards, er kunnen zowel alleenstaand instanties als replica sets aan toegevoegd worden.
- MongoDBShardController is verantwoordelijk om de cluster te initialiseren met de verschillende shards, databases, collecties en keys.

- MongoDBDatabase is de voorstelling van een database.
- MongoDBCollection is de voorstelling van een collectie, indien verbonden met een cluster via een database zal deze gedeeld worden over de verschillende shards.
- MongoDBKey is de wijze waarmee een collectie verdeeld wordt over de verschillende shards.



FIGUUR 4.4: MongoDB: Het domeinmodel van de implementatie in IMP.

4.3.3 Resultaat

4.4 PgPool-II (PostgreSQL)

4.5 HBase

Hoofdstuk 5

Besluit

De masterproeftekst wordt afgesloten met een hoofdstuk waarin alle besluiten nog eens samengevat worden. Dit is ook de plaats voor suggesties naar het verder gebruik van de resultaten, zowel industriële toepassingen als verder onderzoek.

Bijlagen

Bibliografie

[1] Eric A. Brewer. „Towards Robust Distributed Systems (Abstract)”. In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '00. Portland, Oregon, USA: ACM, 2000, p. 7–. ISBN: 1-58113-183-6. DOI: 10.1145/343477.343502. URL: http://doi.acm.org/10.1145/343477.343502.

[2] E. F. Codd. „A Relational Model of Data for Large Shared Data Banks”. In: *Commun. ACM* 13.6 (jun 1970), p. 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: http://doi.acm.org/10.1145/362384.362685.

[3] Miniwatts Marketing Group. *World Internet Users Statistics Usage and World Population Stats*. Dec 2013. URL: http://www.internetworldstats.com/stats.htm (bezocht op 14-12-2013).

[4] Florian Heinze. *vsChart.com – The Comparison Wiki*. Dec 2013. URL: http://vschart.com/ (bezocht op 14-12-2013).

[5] Avinash Lakshman en Prashant Malik. „Cassandra: A Decentralized Structured Storage System”. In: *SIGOPS Oper. Syst. Rev.* 44.2 (apr 2010), p. 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: http://doi.acm.org/10.1145/1773912.1773922.

[6] Alex Popescu. *Presentation: NoSQL at CodeMash – An Interesting NoSQL categoriza- tion*. Feb 2010. URL: http://nosql.mypopescu.com/post/396337069/presentation-nosql-codemash-an-interesting-nosql (bezocht op 03-02-2014).

[7] Postgresql. *PostgreSQL - Replication, Clustering, and Connection Pooling*. Okt 2013. URL: http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling (bezocht op 03-02-2013).

[8] *Replication Introduction*. URL: http://docs.mongodb.org/manual/core/replication-introduction/ (bezocht op 04-02-2014).

[9] Ben Scofield. *NoSQL – Death to Relational Databases(?)* Jan 2010. URL: http://www.slideshare.net/bscofield/nosql-codemash-2010 (bezocht op 03-02-2013).

[10] *Sharding Introduction*. URL: http://docs.mongodb.org/manual/core/sharding-introduction/ (bezocht op 04-02-2014).

[11] Christof Strauch. *NoSQL Databases*. 2010. URL: http://www.christof-strauch.de/nosqldbs.pdf.

Fiche masterproef

Student: Thomas Uyttendaele

Titel: Automatisch uitrol van database systemen en vergelijking van beschikbaarheid

Engelse titel: Automatisch uitrol van database systemen en vergelijking van beschikbaarheid

UDC: 621.3

Korte inhoud:

Hier komt een heel bondig abstract van hooguit 500 woorden. LaTeX commando's mogen hier gebruikt worden. Blanco lijnen (of het commando `\par`) zijn wel niet toegelaten!

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Gedistribueerde systemen

Promotor: Prof. dr. ir. Wouter Joosen

Assessor: Ir.

Begeleider: Ir. B. Vanbrabant