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:
Prof. dr. ir. Tias Guns,
Prof. dr. ir. Christophe Huygens

Begeleider:
Dr. ir. Bart Vanbrabant
Dr. Bert Lagaisse

Academiejaar 2013 – 2014

Voorwoord

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.

Voorwoord schrijven

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
DBMS	Databasemanagementsysteem
RDBMS	Relationeel Databasemanagementsysteem
Range Query	Het opvragen van een set van records met behulp van een enkele query
BASE	
ACID	zer

Symbolen

42	aaa

Hoofdstuk 1

Inleiding

Hoofdstuk 2

Overzicht van de technologie

Inleiding

2.1 Geschiedenis van de databasemanagementsystemen

Doorheen de geschiedenis, heeft de mens verschillende manieren gebruikt om data op te slaan en nadien terug te vinden. Doorheen de geschiedenis zijn er verschillende stappen van data management geweest, tot voor het ontstaan van de computer ging dit met pen en papier of met ponskaarten[14]. Met de opkomst van de computer, werden nieuwe methodes gebruikt die doorheen de tijd zijn mee geëvolueerd met de vooruitgang in de technologie en de veranderingen in het gebruik van de data. De hiervoor ontwikkelde software wordt gecategoriseerd onder het **databasemanagementsysteem**(DBMS's). De ontwikkeling en opkomst van de DBMS's kan in verschillende fasen opgedeeld worden.

De eerste DBMS's zijn er gekomen met de introductie van de mainframes zoals UNIVAC1 en de ontwikkeling van specifieke programmeertalen voor het werken met deze data, onder andere conferenties zoals CODASYL hebben de ontwikkeling van COBOL en andere standaarden mee ontwikkeld[14].

Mss ook het toevoegen van IBM hier + (IDS)?

De volgende grote verandering in DBMS's is er gekomen door het artikel van E. Codd over het relationele model in 1969 [7]. De sleutel concepten van het relationele model is dat de data georganiseerd is in relaties (tabellen) die gekoppeld zijn door middel van keys (constraints) waarbij redundante data wordt vermeden. Voorbeelden van populaire DBMS's (relationele DBMS's) die het model implementeren zijn Oracle, MySQL en PostgreSQL. Meer informatie komt aanbod in de volgende sectie.

De laatste nieuwe generatie zijn NoSQL databases die sinds 2000 zijn begonnen, NoSQL staat voor '*Not only SQL*'. Deze systemen zijn er gekomen als reactie op het relationele model voor een meer flexibele database, lagere complexiteit, hogere

doorvoer van data, horizontale schaalbaarheid en het draaien op commodity hardware. Verschillende voorbeelden van NoSQL systemen zijn Google BigTable, Amazon Dynamo, HBase, MongoDB, ... [34] Meer informatie en een vergelijking met relationele databases komt aanbod in de volgende sectie.

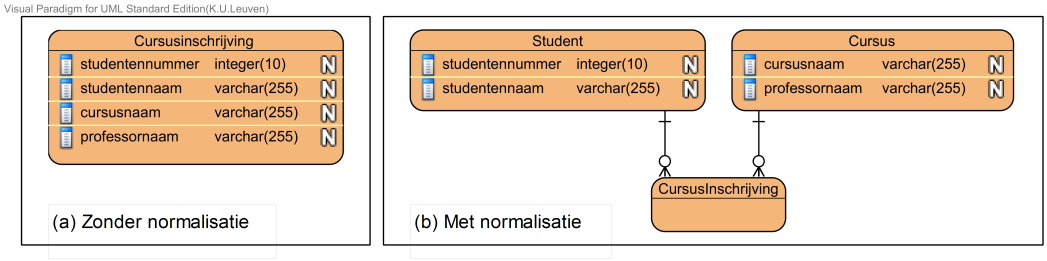
2.2 Relationele en NoSQL databases

Yeaah

2.2.1 Relationele database

Een RDBMS is een DBMS gebaseerd op relationele model voor het structuren van de database.

Het relationele model is gebaseerd op theoretische wiskundige principes zoals de set-theorie en eerste-orde predicaten logica. Het model organiseert de data in tabellen en relaties tussen de relaties. De tabel heeft kolommen die verschillende velden voorstellen en elke rij een collectie van gerelateerde datawaardes is. De relaties tussen de verschillende tabellen toont mogelijke connecties. Een belangrijke eigenschap is dat de tabellen en relaties genormaliseerd worden, hiermee wordt redundante informatie verwijderd. Dit zorgt voor een hogere data integriteit en een vermindering in data anomalieën die kunnen optreden bij een update.[10]
Dit kan geïllustreerd worden met het korte voorbeeld in figuur 2.1: de professor voor een vak zal bij elke student hetzelfde zijn, het veranderen van een professor voor een vak zou in het eerste geval een update van alle ingeschreven studenten inhouden, in het tweede geval is dit maar één enkel record, hetzelfde geldt voor de student.
Interactie met de RDBMS gebeurt op basis van SQL (Structured Query Language), een taal gebaseerd op de relationele logica geeft uitgebreide query mogelijkheden aan de gebruiker van de software.



Figuur 2.1: Relationeel datamodel (a) zonder en (b) met normalisatie

Een belangrijk concept in een relationele database is ACID tussen verschillende transacties:

Atomair (Atomicity) Een database transactie moet oftewel volledig uitgevoerd worden oftewel heeft geen enkele databasebewerking plaatsgevonden.

Consistent (Consistency) Een transactie behoudt de consistentie als de volledige uitvoering van de transactie de database van één consistente staat naar een andere brengt. Een consistente staat is een staat die ervoor zorgt dat waardes van een instantie consistent zijn met de andere waarden in dezelfde staat. Een voorbeeld is het overschrijven van €50 van persoon A naar B, op het einde moet de totale som nog steeds gelijk zijn, A €50 minder en B €50 meer.

Geïsoleerd (Isolation) Een transactie moet uitgevoerd worden alsof ze geïsoleerd is van andere transacties, die eventueel gelijktijdig uitgevoerd worden.

Duurzaam (Durability) Een voltooide transactie kan later niet ongedaan gemaakt worden. Deze verschillende concepten bieden de gebruiker van het RDBMS garanties die de programmeur van de database kan gebruiken voor zijn systeem. Daartegen over staat wel dat dit de complexiteit van de RDBMS groeit, ook indien dit voor bepaalde toepassingen misschien niet nodig is.

2.2.2 NoSQL database

NoSQL DBMS zijn ontstaan als reactie op 'one size fits all'-gedachte die RDBMS's volgen. Dit is ook zichtbaar in de verschillende NoSQL systemen, ze bestaan in verschillende variëteiten, elk voor met hun eigen eigenschappen en toepassingsgebied, toch is er een rode draad te vinden tussen de verschillende systemen vergeleken met RDBMS:

- **Lagere complexiteit**: NoSQL systemen bieden minder opties en features aan dan de RDBMS omdat veel applicaties die nu eenmaal niet nodig hebben. Bijvoorbeeld in een sociale netwerk moet een post niet onmiddellijk beschikbaar zijn voor al de vrienden van een persoon, maar dit mag even duren.
- **Hogere doorvoer**: Talrijke NoSQL systemen bieden een hogere doorvoer van data aan, meestal gecombineerd met een lagere complexiteit.
- **Horizontale schaalbaarheid en werkend op commodity hardware**: Waar grote RDBMS's draaien op dure high-end systemen, was het bedoeling van NoSQL databases om te werken met eenvoudige machines (commodity hardware).
  Horizontale schaalbaarheid staat voor het toevoegen extra machines aan een systeem voor extra resources, in tegenstelling tot verticale schaalbaarheid waar krachtigere machines worden gebruikt. In horizontale opschaling wordt de data van een enkele database verspreid over verschillende machines die elk maar een deel opslaan.

NoSQL systemen combineert deze twee elementen en biedt hierdoor een schaalbaar systeem aan met basis componenten.

- **Datamodel dichter bij objecten**: De meeste NoSQL systemen zijn zodanig ontworpen dat deze de vertaling van objecten naar opslag eenvoudiger of meer gelijkend maken dan RDBMS's.

Deze verschillende argumenten, leiden vervolgens tot een tegenreactie op ACID: BASE.

- Basis beschikbaarheid (**B**asically **Availability**): Een applicatie werkt zo goed als heel de tijd.
- **S**oft State: De data moet op een bepaald moment niet volledig consistent zijn.
- Eventuele consistentie (**E**ventual Consistency): De database zal na enige tijd in een consistente status uitkomen. Eventuele consistentie kan op zijn beurt opnieuw onderverdeeld worden in 4 categorieën [20, slide 16]:

Ga hier dieper op in

  - *Read your own writes* consistentie: Ongeachte van de server waarop een gebruiker leest, zal hij zijn update onmiddellijk correct lezen.
  - *Session* consistentie: De gebruiker zal zijn updates onmiddellijk kunnen lezen binnen dezelfde sessie, een sessie is hierdoor meestal gelimiteerd tot een enkele databaseserver.
  - *Casual* consistentie: Als een gebruiker versie X leest en vervolgens versie Y schrijft, zal elke gebruiker die versie Y leest ook versie X lezen.
  - *Monotonic Read* consistentie: Dit levert monotone tijdsgaranties dat een gebruiker enkel recentere data versies in de toekomst zal lezen.

Classificatie van NoSQL systemen

Er zijn vele NoSQL systemen ontworpen gedurende de laatste jaren, elk met hun eigen variëteit en functionaliteit. Er bestaan verschillende manieren om de systemen te classificeren, maar één van de meest gebruikte doet dit op basis de data modelering, een korte vergelijking bevindt zich in tabel 2.1.

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

Tabel 2.1: Classificatie en categorisatie van NoSQL DBMS's door Scofield en Popescu. [33] [29]

Column Model 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 intelligentie. Het systeem is geïnspireerd door de paper van Google's Bigtable [5]. [34]

Graph Model In een grafen model, wordt de data voorgesteld en opslagen volgens de grafen theorie: knopen, lijnen en eigenschappen op de knopen en lijnen. [2].

Document Model Document systemen zijn volgens vele de volgende stap in key-value systemen, waar deze complexere structuren toe laten, dit door middel van meerdere key/value paren per element. [34]
Een document moet geen vaste structuur hebben maar elk document op zich kan verschillende velden hebben, dit kan bijvoorbeeld gezien worden bij boeken. 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.

Key/Value Model Key-value systemen hebben een heel eenvoudig data model, data kan opgeslagen, 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 performantie, schaalbaarheid en flexibiliteit. [34]

extra conclusion text?

2.3 Bespreking van verschillende DBMS's

De verschillende categorieën databases zijn besproken, voor deze thesis zullen enkele relationele en NoSQL DBMS's in meer detail worden besproken. Databases uit de volgende 4 categorieën komen verder aanbod, er is gekozen om de Graph NoSQL DBMS's niet te bespreken omdat deze een heel andere dataset ondersteunen als de overige categorieën.

- Column NoSQL DBMS's: Cassandra, HBase
- Document NoSQL DBMS's: Apache CoucheDB, MongoDB
- Key-Value NoSQL DBMS's: LightCloud (Tokyo), MemCache, Redis, Riak, Project Voldemort
- Relationele NoSQL DBMS's: MySQL, Pgpool-II (PostgreSQL)

2.3.1 Column database

Nalezen

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 [19], 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.

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.

2.3.2 Document database

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.

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.

2.3.3 Key-Value database

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.

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.

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.

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.

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.

2.3.4 Relationele database

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).

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 [30].

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.

2.4 Objectieve vergelijking van de verschillende systemen

De databasemanagementsystemen zijn voor meer dan 40 jaar actief. Met de opkomst van het relationele model in 1969 door E. Codd [6], zijn gedurende vele jaren de relationele DMBS de leidende technologie geweest. Maar de afgelopen 10 jaar is er in het landschap veel veranderd met de opkomst van de NoSQL DBMS's die afstappen van het ACID naar BASE, meer gefocust op het werken op grote data in een gedistribueerde omgeving. Er worden talrijke assumpties gemaakt over NoSQL, ze zouden een hogere performantie leveren, een hogere beschikbaarheid (availability) en zouden leiden onder het principe van eventuele consistentie (eventual consistency).

2.4.1 Performantie benchmarking

Nog is bekijken binnenkort

Indien men verschillende DBMS wilt vergelijken bestaan er al enkele tools en studies om de performance te kunnen vergelijken, een blogpost van A. Popescu [28] geeft een overzicht van verschillende benchmarking tools.

Als eerste hebben vele DBMS interne benchmarking tools, waarmee de database op verschillende configuraties kan getest worden en vergeleken worden. Deze resultaten zijn nuttig indien het DBMS al gekozen is en men bezig is met het aanpassen van de parameters of om te onderzoeken wat net de bottleneck is in een bepaald systeem. Een voorbeeld hiervan is mongoperf[1] voor MongoDB.

Andere studies focussen op het testen van verschillende systemen en daarbij kunnen verschillende doelstellingen zijn: het ontwikkelen van een breed toepasbare tool, het testen van een grote verscheidenheid van DBMS's of het testen van een specifieke categorie van systemen. Elke van deze benchmarking brengt extra kennis bij maar heeft ook zijn beperkingen. Het totaal pakket van al de testen kan een gebruiker een overzicht geven.

De eerste categorie, het **ontwikkelen van een tool**, heeft als grote voordeel dat andere gebruikers nadien de testen opnieuw kunnen uitvoeren met de huidige systemen. Het is namelijk niet gegarandeerd dat het resultaat van een jaar geleden met de nieuwste versie nog te vergelijken is. Het grootste nadeel is het soort testen dat

[1] http://docs.mongodb.org/manual/reference/program/mongoperf/

kan uitgevoerd worden, er is een grote variëteit aan systemen elk met hun eigen datastructuur en query mogelijkheden. De tool moet dus een gemeenschappelijke subset zoeken en enkel dit soort queries kunnen getest worden. Een voorbeeld van zulk een tool de opensource tool YCSB[8]. In deze tool kan elk DBMS's getest worden zolang een basisset van 5 queries kan implementeren: het invoegen, updaten, verwijderen en opvragen van een enkel record met daarnaast ook de mogelijkheid tot range queries, met behulp van 1 query een verzameling van records tegelijk op te vragen.
Sommige systemen ondersteunen bepaalde queries niet onmiddellijk maar bevatten wel de functionaliteit om deze via via te implementeren, bijvoorbeeld een update kan geïmplementeerd worden door het opvragen, verwijderen en vervolgen invoegen van de geüpdatete record.

Een volgende categorie zijn de **resultaten van gerelateerde DBMS's**, in deze categorie zijn er voornamelijk resultaten te vinden van systemen met hetzelfde datamodel. Het grote voordeel hieraan is dat deze systemen in de meeste gevallen een vrij gelijkaardige set aan query mogelijkheden bevatten waardoor er meer diepgang is dan tussen meer verschillende systemen. Een voorbeeld van zulk onderzoek is gedaan door P. Pirzadeh et al[27] voor de key-value systemen, meer specifiek is er gefocust op het uitvoeren van range queries tussen Cassandra, HBase en Voldemort. Hoewel in de categorisatie van deze thesis de eerste twee onder column NoSQL vallen, zijn deze nog vrij gelijklopend.
In deze categorie vallen ook de resultaten die meestal getoond worden op de website van de DBMS's, een vergelijkende benchmark met andere soortgelijke systemen. Hoewel de resultaten niet altijd volledig objectief zijn, kan de gevolgde methode wel interessant zijn. Een voorbeeld van deze studie is de key-value benchmarking van VoltDB[15] waar Cassandra en VoltDB vergeleken worden, een belangrijke kanttekening is dat de auteur zelf al aanhaalt dat de systemen vrij verschillend zijn zoals appels en appelsienen. GLS check

Als laatste categorie, zijn er de **resultaten van verschillende DBMS's** waar zeer verscheidene systemen met elkaar getest worden. De belangrijkste voordeel is dat er resultaten zijn die verschillende soorten met elkaar vergelijken en waardoor niet alleen verschillen in het datamodel kunnen vergeleken worden in toekomstige studies maar ook performantie verschillen. Het nadeel is ook hier dat er een gemeenschappelijke subset gevonden moet worden, hierdoor kunnen bepaalde databases hun kracht net niet laten zien. Verschillende van deze onderzoeken zijn [35] en [31]. Deze laatste maakt gebruik van de YCSB tool die hierboven besproken was.

2.4.2 Consistentie testen

Zoals besproken in het vorige gedeelte, is er al relatief veel onderzoek gebeurd naar de performantie van de verschillende systemen. Maar de NoSQL systemen zijn gebouwd naar BASE, maar wat betekenen die basis beschikbaarheid, eventuele consistentie nu in de praktijk, hoe lang duurt de soft state van inconsistentie?

Een recent artikel [13] (maart 2014), begint met het stellen dat er momenteel nauwelijks gequantificeerde methodes bestaan om de eventuele consistente te meten. In hun artikel stellen zijn mogelijke methoden voor: de actieve of passieve analyse. De *actieve* analyse bestaat het wegschrijven van data in een database waarna 1 of meerdere andere gebruikers meten hoe lang het duurt vooraleer zij de nieuwe waarde lezen. De *passieve* analyse wordt er gekeken hoe de gebruiker interageert met het systeem en hoe de data updates zich gedragen. Leest deze altijd de laatste waarde (=strikte consistentie)? Is het mogelijk dat een nieuwe waarde al wordt gelezen voor de schrijfactie voltooid is?
Beide systemen hebben hun eigenschappen, de actieve analyse kan gezien worden als een systeem georiënteerde analyse en test hoe lang het duurt voor de data beschikbaar is over de verschillende systemen en heeft als uitkomst hoe DBMS's zich verschillend gedragen ten opzichte van elkaar of in verschillende netwerk- en hardwareomgevingen. Bij de passieve analyse is georiënteerd naar de gebruiker toe, hoe moet deze zijn toepassingen aanpassen, wat zijn specifieke eigenschappen van het systeem?

Voor beide analyse methodes is er al onderzoek gebeurd, maar het meeste is gebeurd naar de actieve analyse. Onder andere Duitse onderzoekers hebben op het Amazon S3 platform getest hoe lang het duurt vooraleer data geschreven in MiniStorage, een database systeem, beschikbaar is voor alle gebruikers. [1].
Daarnaast zijn er ook 2 interessante resultaten gevonden: allereerst heeft het Amazon S3 systeem geen monotone lees consistentie, daarnaast bleek het inconsistentie interval voor een bepaald record periodiek verloop te hebben.

De YCSB software van hierboven is door onderzoekers in de VS uitgebreid naar YCSB++[25] waardoor deze meer ondersteuning heeft om het meten van systeembelasting maar ook voor de eventuele consistentie. Hoewel enkele van de systemen die zij testen in principe strikt consistent zijn zoals HBase, worden deze eventueel consistent door het gebruiken van buffers bij de gebruiker. Vervolgens testen zij hoe lang het duurt voor de data ook gelezen kan worden. Een probleem met deze methode is dat deze vertraging sterk afhankelijk is van het aantal acties van de schrijvende gebruiker: indien er meer geschreven wordt, zal de buffer sneller verzonden worden en dus sneller beschikbaar zijn.
Hoewel zij stellen dat er ook zijn gedaan naar eventuele consistentie voor Cassandra en MongoDB, zijn de resultaten niet beschikbaar in het artikel of op de website.

Deze paper? [37]

Bij Netflix heeft men aan passieve analyse gedaan op hun Cassandra systeem [18] waar zij in hun testen geen consistentie problemen vonden naar de gebruiker toe. Er is geen vermelding hoeveel vertraging er zit tussen beide transacties. Volgens hun gaat het meer om de perceptie dat data verkeerd kan gelezen worden en de angst van het middle management.

2.4.3 Beschikbaarheidstesten

Een derde verschilpunt is hoe de systemen omgaan met het falen van een enkele server

Uitleggen failure handling en automatic recovery

en dit onder verschillende opties: Het is mogelijk dat deze tijdelijk uitgeschakeld wordt wegens onderhoudt, het kan om een onverwachte crash van het DBMS's gaan of zelfs een hele server, tenslotte kunnen er ook nog netwerkproblemen optreden waardaar deze (tijdelijk) niet beschikbaar is.

Nu hoe gaan deze systemen om het falen en terug online brengen van de systemen: zijn er geen acties mogelijk op de server, worden de connecties tijdelijk verbroken, is er een verhoogde of verlaagde vertraging op de transacties? En detecteert het systeem automatisch wanneer de oorspronkelijke server terug online komt of moet gemeld worden? In een NoSQL DBMS waar gewerkt wordt commodity hardware, kunnen zulke zaken regelmatig gebeuren en systemen reageren verschillend op deze acties.

Voor informatie over hoe de verschillende systemen reageren, is het momenteel uitzoeken op de website van de software verdeler en zelf uittesten van het gedrag. Naar mijn onderzoek, bestaat er nog geen vergelijkende studie tussen de verschillende systemen.

2.5 Conclusie

Na het artikel van E. Codd [6] in 1969, heeft het relationele model een dominante rol gespeeld in het database model. Met de opkomst van het internet, grotere hoeveelheden data en steeds complexere RDBMS's, is er een nieuwe stroming gekomen in de database wereld, de NoSQL DBMS's. Deze beloven betere schaalbaarheid, hogere performantie en dit op commodity hardware ten aanzien van de ACID eigenschappen naar BASE: een hogere beschikbaarheid en eventuele consistentie.

In deze thesis zal er eerst een algemene methode voorgesteld worden om verschillende systemen te testen naar de eventuele consistentie, behandelen van storingen (failure handling) en (automatisch) herstel (automatic recovery).
Daarna zal deze methode uitgevoerd worden op verschillende databases waar duidelijk verschillende resultaten en opvattingen gezien kunnen worden. Deze leiden tot enkele initiële conclusies die in de toekomst verder kunnen worden onderzocht.

Hoofdstuk 3

Methodiek van de testen

Dit hoofdstuk behandelt de wijze waarop de testen naar consistentie, failure handling en recovery worden uitgevoerd. De methodiek kan opgedeeld worden in 4 grote stappen: het opstellen, kalibreren, testen van de systemen en tenslotte het verzamelen en analyseren van de resultaten.

Opstellen van de testomgeving Deze eerste stap is voor het installeren en configureren van de DBMS en de testsoftware. Een variatie in hardware van de systemen, versienummer van de software of een verschillende netwerkinfrastructuur kunnen de uiteindelijke testresultaten beïnvloeden.

Calibratie van de testomgeving In de uiteindelijke testen wordt het gedrag onder normale belasting getest. Afhankelijk van de gekozen systemen, netwerkinfrastructuur zal dit voor elke DBMS een verschillende belasting geven. Deze stap bepaalt hoeveel gebruikers er zijn in het systeem en hoeveel bewerkingen er uitgevoerd worden per second.

Testen van de systemen In deze stap worden de testen op de verschillende systemen uitgevoerd. Voor deze methodiek is het mogelijk om te testen hoe de latency van een bewerken zich gedraagt voor, tijdens en na het falen en herstellen van een systeem. Daarnaast is er ook een testmethode voor een actieve analyse van eventuele consistentie.

Verzamelen en analyseren van de testdata In de laatste stap wordt de data van de vorige stappen verzameld en de resultaten worden visueel voorgesteld. Met behulp van de uitgebreide testdata, is het ook mogelijk om bepaalde conclusies te maken over een passieve analyse van eventuele consistentie.

In de volgende secties komen de verschillende stappen in meer detail aanbod. figuur met overzicht

3.1 Stap 1: Opstellen van de testomgeving

Het installeren en configureren een softwarepakket op een enkel systeem, is in Unix systemen veelvuldig geautomatiseerd met behulp van tools zoals apt-get en yum. Voor een systeem in een gedistribueerde omgeving, is de situatie iets ingewikkelder. Naast de lokale installatie en configuratie, is er ook een gedistribueerde configuratie stap.

In deze gedistribueerde configuratie stap, worden de verschillende systemen van elkaars bestaan op de hoogte gebracht, hiervoor bestaan twee verschillende methodes maar ook een combinatie van de configuratie methodes is mogelijk.

Configuratie bestanden Met deze methode dient er op elke lokaal systeem een configuratiebestand aangemaakt of aangepast worden met hierin één of meerdere andere leden. Nadien kunnen de verschillende lokale systemen opgestart worden of de configuratiebestanden worden opnieuw ingeladen in de al draaiende systemen. Vervolgens zullen met de configuratie elkaar vinden en samen het database systeem vormen. Deze informatie kan een ip adres zijn van één of meerdere systemen maar dit kan ook een naam zijn van het systeem die met een broadcast verdeeld wordt.

Centrale configuratie Bij een centrale configuratie, worden de systemen lokaal opgestart zonder enige lokale configuratie. Vervolgens wordt via een console, web-interface, ... connectie gemaakt met een node. Deze krijgt configuratie informatie hoe deze zich moet gedragen en volgt deze informatie op. In deze systemen is de configuratie tijdens installatie gelijk en wordt de configuratie verspreid wanneer de systemen al draaien.

Deze stap is gelijk aan het opzetten van het systeem in een productie omgeving, na het uitvoeren van deze stap, zou het DBMS immers succesvol moeten werken.

3.2 Stap 2: Calibratie van de testomgeving

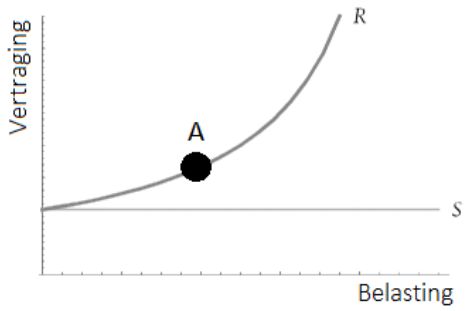
Afhankelijk van de onderliggende infrastructuur en het soort DBMS, kan het systeem een verschillend gedrag hebben onder dezelfde configuratie. Voor de eigenlijk testen is het de bedoeling om een middelmatige belasting te hebben. De queuing theory geeft de eigenschap dat $R = S + W$ waar R de totale vertraging is, S de processing time en W de tijd in de wachtrij [21]. Dit verband is visueel voorgesteld ten opzichte van de belasting in figuur 3.1(a).

Betere referentie dan Oracle?

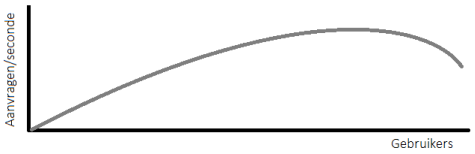
Vind er betere ;-)

Deze belasting kan afhankelijk zijn verschillende elementen, er worden 5 verschillende mogelijke parameter groepen besproken:

Hoeveelheid data per gegevensrecord Elke record in de database kan bestaan uit verschillende kolommen en per kolom een waarde. Het is belangrijk om te



(a) Verband vertraging ten opzicht van de belasting van een DBMS.



(b) Verband vertraging ten opzicht van het aantal gebruikers.

Figuur 3.1

definiëren hoe groot een gemiddeld record is, aangezien dit een invloed heeft op het schijfgebruik en het netwerkverkeer.

Type van queries De opgeslagen data kan opgevraagd worden op verschillende wijze: data kan ingevoegd, aangepast, opgevraagd of verwijderd worden. Daarnaast kan dit gebeuren voor 1 of meerdere records tegelijk. Afhankelijk van de relatieve verhouding van deze soorten, kan een ander resultaat bekomen worden: sommige DBMS's zijn meer geschikt voor veelvuldig lezen dan schrijven en vice versa.

Query specificatie Bij het opvragen of verwijderen van een record, kan er een verschil zijn naar processing tijd afhankelijk van hoe lang geleden de record geschreven is en of naburige data onlangs gelezen is. Vandaar dat ook het datadistributie gekozen moet worden. Voorbeelden van verschillende technieken zijn: voornamelijk de laatste data lezen, een uniforme kans voor alle data of bepaalde records regelmatig lezen.

Aantal connecties of gebruikers In een gedistribueerde omgeving zullen meestal meerdere gebruikers tegelijk actief zijn, maar sommige systemen hebben een voorkeur naar weinig connecties met grote hoeveelheden data, andere kunnen meer gebruikers tegelijk behandelen. Het totaal aantal queries kan berekend worden als: $\#Queries = \#Gebruikers * \#QueriesPerGebruiker$. In deze stap wordt er verondersteld dat de gebruiker het maximaal aantal queries doet, dus $1/Vertraging$. Rekening houdend met de exponentiële groei van de wachtrij vertraging (figuur 3.1(a)), betekent dit dat er een totaal maximum aantal queries per seconde bereikt wordt bij een bepaald aantal gebruikers. In deze stap wordt er gezocht naar dit aantal gebruikers, zie figuur 3.1(b).

Aantal queries per seconde In de vorige stap is er de optimale configuratie bepaald om het systeem maximaal te belasten. Maar in het begin is er gesteld dat er gezocht wordt naar een gemiddelde belasting voor dit aantal gebruikers. Er wordt gekozen om matige belasting, in figuur 3.1(a) zou dit punt A zijn.

Met de parameters afkomstig uit de calibratie, kunnen de testen opgestart en uitgevoerd worden.

3.3 Stap 3: Testen van de systemen

In deze thesis zullen er 2 verschillende soort testen uitgevoerd worden, de beschikbaarheid en consistentie testen, welke beide dezelfde algemene stappen volgen, elk met hun eigen specifieke parameters. Er zijn de 6 deelstappen:

Opstellen van de database In stap 2 was er gekozen voor een bepaalde datastructuur, deze structuur wordt zo goed mogelijk meegegeven aan de DBMS zodat deze optimale allocatie kan doen.

Inladen van de data Een bepaalde hoeveel data wordt vooraf ingeladen. Dit wordt gedaan om een basis dataset te hebben die nodig is voor sharding, het opsplitsen van de data over verschillende servers. In bepaalde DBMS's wordt dit automatisch toegepast, maar enkel bij een bepaald hoeveelheid data. Dit gebeurt op maximale snelheid.

Pauze Na het inladen van de data wordt enige tijd gewacht. Zoals aangetoond in YCSB++[25, Figuur 9], is er hogere vertraging in de DBMS's onmiddellijk na het inlezen. Dit kan onder andere te wijten zijn doordat data nog weggeschreven moet worden naar schijf of in bepaalde systemen zou het kunnen dat de sharding gebeurt op momenten met weinig belasting. Om deze piek niet te hebben, wordt er enige tijd gewacht.

Opstarten van de test (opstart kost) De test wordt opgestart en de begint te lopen. In veel gevallen is er in het begin een opwarmfase nodig omdat de vertraging net hoger of lager is als na enige tijd. Deze hogere tijd is onder andere te verklaren doordat de connectie opgezet moet worden en caches worden gevuld. Soms is deze lager doordat de schijf nog niet belast is of de er nog veel schrijfbuffers leeg zijn. Om dit gedrag te vermijden, start de data van de eigenlijke test pas na deze stap.

Uitvoeren van de test De eigenlijke test wordt uitgevoerd en de data wordt verzameld en opgeslagen. De details van de beide testen volgen achteraf.

Terugbrengen naar beginstatus Na het uitvoeren van de test, wordt het DBMS terug naar de beginstatus gebracht. Onder andere de database en de data wordt volledig verwijderd. Belangrijk in dit geval is het controleren of de data volledig verwijderd is, in bepaalde gevallen wordt er nog ergens een veiligheidskopie bijgehouden dat herstelt wordt bij een mogelijke volgende batch.

De twee verschillende testmethodes zullen nu in meer detail behandeld worden.

3.3.1 Beschikbaarheidstest

Bij de beschikbaarheidstest wordt er gekeken hoe het systeem reageert op tijdelijk (on)verachte onbeschikbaarheid van een deel van het systeem. In deze testen worden er 3 mogelijke manieren getest die de systemen onbeschikbaar terwijl er de basisbelasting die bepaald is in stap 2, op uitvoert.

Zachte stop De DMBS service wordt gevraagd om te stoppen. Op deze manier krijgt de service eerst een signaal dat deze moet stoppen en kan deze de andere waarschuwen. Achteraf wordt dezelfde service terug opgestart. Dit simuleert het gepland uitschakelen van een systeem.

Harde stop De DMBS service wordt onmiddellijk gestopt door het process te beëindigen. De service heeft geen tijd om de andere te waarschuwen. Achteraf wordt dezelfde service terug opgestart. Dit simuleert een crash van de service die pas na enige tijd opgemerkt wordt.

Netwerk onderbreken Al het netwerk verkeer wordt gedropt zonder enige waarschuwing. De service heeft geen tijd om de andere te waarschuwen én de zender krijgt geen onbereikbaar antwoord. Achteraf wordt het netwerk verkeer terug toegelaten. Dit simuleert een onderbroken internetverbinding of een onbereikbare server om eender welke andere reden.

Een zelfde systeem kan sterk verschillend reageren op de verschillende situaties: waar de eerste situatie nog eenvoudig is te behandelen doordat men op de hoogte is, is de tweede situatie al moeilijker alhoewel men wel antwoord krijgt dat de service niet beschikbaar is bij het contacteren. De derde situatie is het moeilijkste te behandelen omdat men niet weet of de berichten naar de server niet aankomen of de antwoorden verloren gaan.

3.3.2 Consistentie test

In de consistentie test wordt onderzocht welke consistentie de DBMS ondersteund. Zoals voordien besproken in deel 2.2.2, bestaan er verschillende soorten.

In deze testen is er gekozen om caching bij de gebruiker **uit te schakelen**, dit om de reden dat dit gedrag zeer onvoorspelbaar is en afhankelijk van andere acties van de lezer en schrijver. Een andere reden is dat eventueel consistentie alleen een probleem is voor data die onmiddellijk beschikbaar moet zijn, met andere woorden data die men niet mag cachen. Dit heeft als gevolg dat de belasting op de server hoger kan zijn.

Beschrijving van de test Deze test bestaat uit 3 soorten gebruikers: er is 1 gebruiker die data schrijft (=S), een aantal lezer (=L's) en tenslotte zijn er nog andere gebruikers die ervoor zorgen dat er samen met de andere gebruikers een

belasting is zo dicht mogelijk bij de basisbelasting. Het is belangrijk dat er een exacte synchronisatie in tijd is tussen de verschillende gebruikers, dit om de geregistreerde tijdstippen te kunnen vergelijken.

figuur met een periode

Taak van de schrijver De schrijver schrijft, zoals zijn naam voorspelt, voorafbepaalde data weg op vooraf vastgelegde momenten. Deze data kan een nieuw record of een update van een record zijn. Deze registreert op welk moment deze taak exact is gestart en hoe/wanneer deze is beëindigd.

Taak van de lezer De taak van een individuele lezer is om op vooraf vastgelegde momenten de data van de schrijven te gaan lezen. Dit wordt periodiek herhaald tot de data correct is gelezen of een bepaalde tijd is verstreken. Er kan ook beslist worden om ook als de data correct gelezen is, opnieuw te proberen. De lezer registreert elke keer deze gaat lezen op welk moment deze exact is gaan lezen en wat het resultaat van de actie is.

Het plannen van de lezers Zoals voordien vermeldt gaat de schrijver op bepaalde momenten schrijven en de lezer herhaalt het lezen periodiek. Het doel van de lezers is om allereerst verbonden te zijn met verschillende servers en daarnaast meer testpogingen hebben op het lezen van de data. Om deze laatste redenen worden de starttijdstippen voor de data te lezen, gelijk gespreid tussen de verschillende lezers.

Soorten eventuele consistentie Met deze uitgevoerde testen en data kan aangetoond worden dat bepaalde systemen bepaalde eventuele consistentie vereisten niet volgen. Het is in veel gevallen niet mogelijk om te bewijzen dat deze het wel uitvoeren omdat een voorbeeld niet sluitend is, maar een tegenvoorbeeld wel.

Strikte consistentie Een systeem is niet strikt consistent indien één van de lezers het nieuwe record of de update niet lezen *indien de leesactie gestart is na het voltooien van de schrijfactie*. In bepaalde gevallen is zelfs mogelijk om deze garantie strikter te maken tot *indien de leesactie voltooid is na het voltooien van de schrijfactie*.

Read your own writes consistentie Deze eventuele consistentie kan ontkracht worden indien een schrijver onmiddellijk na het voltooien zijn eigen data opvraagt en niet de nieuwe waarde leest. Dit kan enkel getest worden indien de DBMS het mogelijk maakt om met een gebruiker te verbinden naar meerdere servers. In onze methode is dit niet geïmplementeerd omdat de data van de testen zonder dit al de mogelijkheid tot het maken van een conclusie.

Session consistentie Aangezien er al conclusies getrokken kunnen worden over de vorige en deze nog een zwakkere eis heeft, is dit niet geïmplementeerd. Maar

opnieuw kan dit ontkracht worden door met de zelfde connectie als de schrijver te lezen.

Casual consistentie Deze test kan uitgevoerd worden de schrijver verschillende schrijfacties na elkaar te laten uitvoeren met tussendoor te zijn laatste actie te lezen. De lezer leest de records in dezelfde of omgekeerde volgorde. Indien deze data van een latere schrijfactie leest maar nog niet van een vroegere, is dit ongeldig. De eis kan strenger gemaakt worden door de schrijver tussendoor niet te laten lezen, dit zou andere resultaten kunnen hebben. Deze consistentie is in zijn totaal niet getest.

Monotonic Read consistentie In deze test blijft de lezer continue opnieuw proberen om dezelfde data te lezen, eenmaal deze een nieuwe versie heeft gelezen zou deze nooit meer een oudere mogen lezen.

Zoals duidelijk hierboven, biedt deze aanpak de mogelijkheid aan om naast een actieve ook een passieve analyse te doen op de data.

3.4 Stap 4: Verzamelen en analyseren van de testdata

Na het uitvoeren van de testen, dient de informatie die verschillende schrijver en lezers hebben vergaart samen gebracht te worden. Vervolgens kan de informatie verwerkt worden te bepalen hoe lang het duurt voor de data overal consistent is (actieve analyse) of om tegenvoorbeeld te zijn voor een bepaalde consistentie categorie (passieve analyse). De analyse kan gebeuren op basis van de besproken methodes hierboven.

Tenslotte worden er in de testen nog grafieken gegenereerd van de aanwezige data, dit met de voor de hand liggende reden dat een figuur veel meer duidelijkheid brengt over de data dan duizenden getallen.

3.5 Conclusie

Todo

Hoofdstuk 4

Implementatie

4.1 Selectie van de DBMS's

4.1.1 Selectiecriteria voor DBMS's

Bij de selectie van de systemen of een systeem een bepaalde eigenschap al dan niet ondersteunt. In het totaal zijn er 5 verschillende eigenschappen waarop wordt geselecteerd.

Vrije software Om testen tussen verschillende DBMS's te kunnen vergelijken op een gelijkaardige infrastructuur, is het nodig dat deze software kan geïnstalleerd worden op de eigen infrastructuur. Daarnaast is er in dit onderzoek gekozen voor gratis beschikbare software.

Persistentie Voor het testen van de beschikbaarheid van de data, is het een voordeel dat de data op harde schijf aanwezig is: bij herstel dient er minder data over het netwerk gestuurd te worden. Om deze reden hebben deze systemen een voorkeur op systemen die de data enkel in geheugen houden.

Replicatie Eén van de testen is de beschikbaarheidstest, indien de data maar op een enkele server opgeslagen is, zal de data op de uitgeschakelde server niet langer beschikbaar zijn. Met replicatie zal de data op verschillende servers opgeslagen worden en zal de data nog steeds beschikbaar zijn in het geval van een enkele uitgeschakelde server.

Data distributie Het is de bedoeling om systemen te testen die een grote hoeveelheid data kunnen opslaan. Om aan deze vereiste te voldoen, is het nodig dat elke server niet al de data opslaat bij een voldoende hoog aantal servers (bij weinig servers, is elke server nodig om aan de replicatie vereiste te voldoen)

		Cassandra	HBase	Apache CouchDB	MongoDB	Lightcloud (Tokyo)	Memcache	Riak	Voldemort	MySQL	Pgpool-II (PostgreSQL)
		Column database		Document database		Key-Value database				Relationele database	
Persistentie											
Replicatie											
Data distributie											
Query soort	Aanpassen										
	Range										

Tabel 4.1: Ondersteuning van de besproken DBMS's naar de selectie criteria met de gekozen systemen geaccentueerd.

Ondersteuning voor verschillende query methodes Bij de testen worden er 5 soorten queries uitgevoerd: invoegen, aanpassen, verwijderen en opvragen van een individueel record en het opvragen van meerdere queries. De DBMS moet ondersteuning voor deze queries. De eerste 4 kunnen in al de systemen geïmplementeerd worden met één of meerdere queries. Maar het opvragen van meerdere queries is in bepaalde systemen niet mogelijk en deze worden om deze reden niet behandeld.

Voor elk van de systemen besproken in sectie 2.3, is het eerste criterium voldaan voor al deze systemen. De overige 4 criteria zijn samengevat in tabel 4.1. Naast de 4 criteria, is er gekozen om systemen van verschillende klassen te hebben met uitzondering van Key-Value. Samen met mijn collega Arnaud Schoonjans [32], zijn er in verschillende systemen verder onderzocht. In deze thesis zijn HBase, MongoDB en Pgpool-II verder onderzocht, in de thesis van mijn collega zijn Cassandra, Apache CouchDB, Riak en MySQL verder onderzocht.

Update table :-)

4.2 Gedetailleerde bespreking van de geselecteerde DBMS's

In dit gedeelte zal elk geselecteerd systemen in meer detail uitgelegd worden, met een focus op de systeem structuur. Een gemeenschappelijk element bij al deze systemen is dat niet alle noden dezelfde functie hebben, in andere DBMS's hebben allen dezelfde taak wat de installatie kan vereenvoudigen.

Voor elk van de geselecteerde systemen zal de aangeboden API besproken worden met een blik op de datastructuur, daarna zal de systeem architectuur besproken worden.

4.2.1 HBase

Data structuur

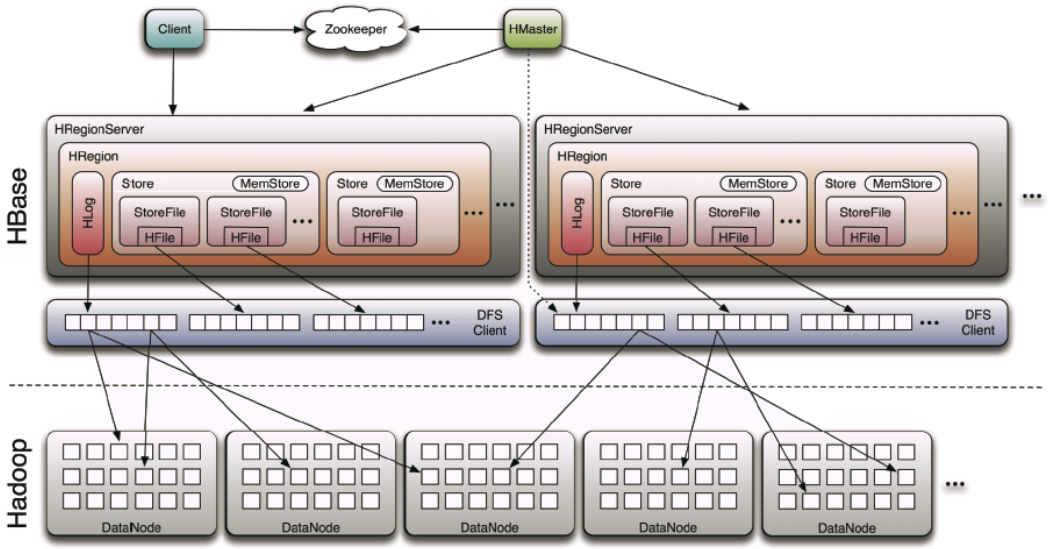
De data in HBase is gestructureerd in tabellen, die aangemaakt worden door middel van de API. Op het moment van aanmaak wordt er een schema van de tabel gemaakt. Voor elke tabel kunnen de kolommen meegegeven worden samen met een *kolom familie* voor elke kolom, maar de kolommen kunnen ook gespecificeerd worden bij het schrijven van data. De gegevens per *kolom familie* hebben dezelfde prefix en zullen fysisch samen opgeslagen worden. Indien verschillende kolommen tegelijk worden gelezen of geschreven, is het aangeraden om deze dezelfde *kolom familie* te geven.

De operaties beschikbaar in dit systeem zijn: get (verkrijgen), put (invoegen), scan (range) en delete (verwijderen). Het aanpassen van gegevens wordt uitgevoerd via een get waarbij een enkele kolom waarde van een record kan aanpast worden. Daarnaast zal er bij een scan een optimalisatie gedaan worden naar de cache grootte. Doordat er geweten is hoe groot een record ongeveer is in zijn geheel én hoeveel records er opgevraagd worden, kan de cache grootte berekend worden zodat er maar een enkele datacommunicatie nodig is.

Architectuur

De gedistribueerde versie van HBase[11] is afhankelijk van 2 andere software systemen, namelijk Zookeeper[16] en Hadoop[3], en volgt hiermee de structuur van Google's BigTable[5] die op zijn beurt afhankelijk is van Chubby[4] en Google File System[12]. Een overzicht van de architectuur bevindt zich in figuur 4.1. Deze 3 systemen zullen kort besproken worden, van Hadoop tot Zookeeper en HBase.

Hadoop[3] HBase maakt gebruik van het Hadoop Distributed File System HDFS, een gedistribueerd file systeem ontworpen om te werken op commodity hardware met een hoge fout tolerantie. HDFS heeft een master/slave architectuur en bestaat uit een enkele **namenode**, de master server, die de naamruimte en toegangscontrole onderhoudt, en **datanode**s. De data wordt opgedeeld in blokken die in een verzameling van datanodes wordt opgeslagen, op deze manier is er data distributie en



Figuur 4.1: Volledige systeemarchitectuur van HBase met Hadoop en Zookeeper. Bron [17]

asynchrone replicatie. Deze master/slave configuratie zijn verschillende soorten van services en dient door de gebruiker zelf geconfigureerd te worden.
In de configuratie die hier gebruikt wordt, is HDFS de methode de data in op te slaan in HBase met automatische replicatie en data distributie. Er is ook nog ondersteuning om dit lokaal op de harde schijf te doen bij een niet distribueerde installatie en om dit weg te schrijven naar S3.[11]

Zookeeper[16] Zookeeper is een service voor het coördineren van gedistribueerde applicatie processen, met deze service biedt primitieven aan om synchronisatie, configuratie onderhoud en groepen en benaming te doen. Zookeeper is op zijn beurt een gedistribueerd master/slave systeem dat ontworpen is om snel te zijn bij dominantie van leesoperaties. De master/slave
HBase gebruikt Zookeeper onder andere voor het bijhouden van de status van regio server en hun locatie. [11]

HBase[11] HBase is een master/slave systeem welke bestaat uit een HMaster en een HRegionServer. De **HMaster** is verbonden met Zookeeper en houdt op deze manier de status van de HRegionServers in het oog. Daarnaast is deze ook verantwoordelijk voor het opsplitsen van data (sharding) over verschillende regio's indien een tabel groeit en het toewijzen van een regio aan een HRegionServer.
De andere soort, **HRegionServer**s, is verantwoordelijke voor het dienen en beheren van regio's. Een regio is een deel van een tabel met daarin de feitelijke data die opgeslagen is in verschillende datanodes. Een HRegionServer zal de strikte consistentie afdwingen in HBase, dit kan doordat een enkele server verantwoordelijk is voor het uitvoeren van de data-updates.

Dit is de globale structuur van het HBase systeem, in het totaal zijn er 5 verschillende soorten systemen, 2 bij Hadoop, 1 bij Zookeeper en 2 bij HBase. Enkele van deze systemen worden best gegroepeerd: de namenode wordt samengesteld met HMaster en een datanode met HRegionServer. Zeker deze laatste heeft een extra performantie invloed: HBase detecteert dat er lokale opslag van de data is en de regio zal steeds deze lokale opslag hebben. Dit gecombineerd met de asynchrone replicatie, zorgt ervoor dat het schrijven van data maar naar een enkele server gestuurd moet worden.

De configuratie van de verschillende systemen gebeurt door middel van configuratiebestanden voor elke service waarna de volledige configuratie door het systeem zelf wordt gedaan, uitgezonderd het aanmaken van een tabel welke door de API wordt opgezet.

4.2.2 MongoDB[22]

Datastructuur

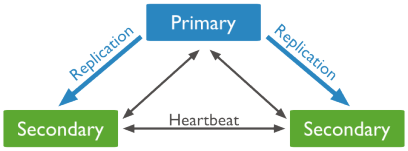
De data in MongoDB is opgeslagen in een database, die op zijn beurt een collectie bevat. Een database en collectie moeten niet aangemaakt worden op voorhand maar kunnen worden aangemaakt bij het wegschrijven van data. Een record is in un benaming een document en kan verschillende soorten velden hebben. Er zijn uitgebreide query mogelijkheden om data in te voegen, aan te passen, te verwijderen of een range aan te maken. Er is ook ondersteuning voor MapReduce[9].

Bij het schrijven van data, kunnen verschillende eisen gesteld worden voor het voltooien van de actie, startende met de actie is over het netwerk verstuurd, de primary heeft de data geschreven tot een meerderheid van de secundaries heeft de data weg geschreven.
Bij het lezen kan men kiezen om de data te lezen van de primary, secundary of de dichtstbijzijnde node. Afhankelijk van de gekozen acties, kan er verondersteld worden dat er een verschillende consistentie garantie zal zijn.

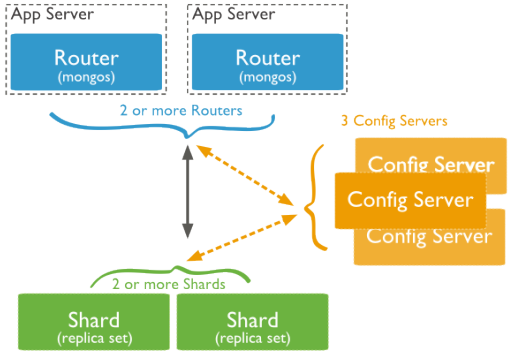
Architectuur

MongoDB is een DBMS dat de vereisten van replicatie en data distributie op een gelaagde manier garandeert. In eerste instantie zal deze de replicatie vereisten invullen, daarna zal hierop horizontale schaalbaarheid ondersteund worden.

Replicatie[23] Replicatie in MongoDB gebeurt door middel van een master/slave configuratie tussen verschillende **MongoD** instanties, of in hun termen primary/secondundaries. Deze instanties verkiezen zelf hun primary die verantwoordelijk is voor het afhandelen van de schrijfacties, de data zal vervolgens gerepliceerd worden naar de secundaries. Een verzameling van deze MongoD instanties wordt een *replicaset* genoemd. Het is slechts mogelijk om een instantie tot een enkele set toe te voegen. De data is beschikbaar zo lang er meer dan de helft van de servers beschikbaar zijn.



(a) Drie leden van een replica set met een primary en 2 secundaries.



(b) Een voorbeeld cluster voor productie met 2 mongos, 3 shards en 3 configuratie servers.

Figuur 4.2: MongoDB Architectuur voor replicatie en datadistributie. Bron figuur 4.2(a): [23], figuur 4.2(b): [24]

Data distributie[24] Horizontale schaalbaarheid wordt in MongoDB aangeboden door verschillende replicaset's of een enkele MongoD instantie te combineren tot een cluster. In het geval van de tweede keuze, zal de data niet gerepliceerd worden en wordt om deze reden niet aangeraden voor productie.

Shards Sharding gebeurt automatisch op een collectie nadat is aangegeven dat men deze wilt verdelen over de gespecificeerde delen. Voor het uitvoeren van deze sharding zijn er nog 2 extra servers types nodig: configuratie servers en toegangsserver.

Configuratie servers De configuratie servers slaan de meta data van de cluster op zoals de verschillende shardings. Deze configuratie set bestaat in productie uit exact 3 servers.

Toegangsserver De toegangsserver haalt de configuratie op uit de configuratie servers en biedt toegang voor de gebruiker aan tot de cluster. Er kunnen een onbepaald aantal toegangsservers zijn in cluster.

De configuratie van de verschillende delen gebeurt op verschillende manieren. Bij replicatie krijgt elke set een naam die in de configuratiebestanden van elke configuratie wordt gezet, nadien wordt via één instantie de verschillende instanties toegevoegd via de API. Bij de cluster worden bij het opstarten van de toegangsservers de set van configuratieservers meegegeven, het opzetten van de verschillende shards gebeurt via een toegangsserver m.b.v. de API.

4.2.3 Pgpool-II (PostgreSQL)[26]

Pgpool-II kan op 4 verschillende manieren werken, in deze opstelling is er gekozen voor de replicatie mode omdat deze zowel replicatie, belastingsverdeling, failover

en online recovery aanbiedt. Er is de mogelijkheid om ook data distributie aan te bieden maar dit is niet getest.

Vervolgens zal de datastructuur en de architectuur besproken worden.

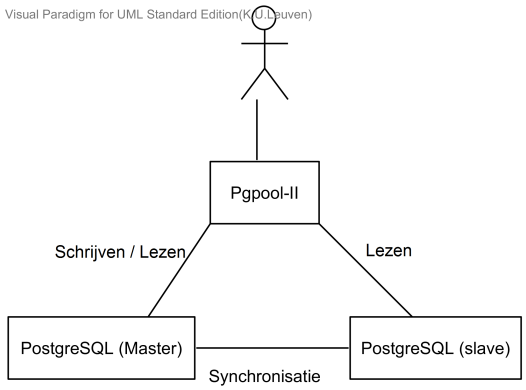
Datastructuur

De data structuur en query mogelijkheden van Pgpool-II zijn gelijklopend aan deze van PostgreSQL. Net zoals in PostgreSQL bestaat het systeem uit een schema die verschillende databases kan bevatten. In een database zijn vervolgens verschillende tabellen die de records bevatten. Voor het opslaan van de data dient de volledige tabel met al de kolommen gespecificeerd zijn.

Pgpool-II ondersteunt bijna de volledige query mogelijkheden die in de testen nodig zijn, er zijn enkele restricties die beschreven zijn op in de sectie *Restrictions* van de documentatie[26]. De query mogelijkheden zijn uitgebreid en ondersteunen naast de benodigde 5 queries, nog join acties.

Architectuur

Een Pgpool-II infrastructuur bestaat uit 2 delen, een data niveau en een routing niveau, een overzicht is gegeven in figuur 4.3.



Figuur 4.3: Systeemarchitectuur van Pgpool-II.

Op data niveau bestaat een individuele service uit een PostgreSQL installatie waarbij enkele extra functies en bestanden worden geïnstalleerd met een aanpassing aan de configuratie bestanden. Daarnaast moet er voor de online recovery ook ssh toegang voorzien worden tot al de PostgreSQL servers. De verschillende data machines hebben een master/slave structuur waar al de schrijfacties naar de master worden gestuurd en de rest verdeeld over al de machines. De master doet aan synchronisatie met behulp van de *Write-ahead-log* van PostgreSQL waar al de schrijfactie worden gelogd.

Op routing niveau draait een Pgpool-II service die als management service dient, hij bepaalt wie master en slave is, volgt de status op van de data services en doet aan online recovery. Bij het aanmaken van een database connectie naar welke service de leesoperaties zullen gaan, zo wordt de leesbelasting verdeeld.

Pgpool-II kan ook in de parallel mode werken zodat er de mogelijkheid is tot horizontale schaalbaarheid, ook is er de mogelijkheid om caching aan te zetten en een integratie met Memcache is ondersteund.

4.3 Selectie en uitwerking van de testsoftware

De testen zijn geïmplementeerd als een uitbreiding van YCSB[8] omwille van verschillende redenen. Allereerst is de broncode publiek beschikbaar onder Apache 2.0, daarnaast is dit een uitgebreid systeem voor het uitvoeren van performantie benchmarking aan de hand van de vertraging op DBMS. Hierdoor heeft deze al een uitgebreide ondersteuning voor tal van systemen, waaronder al de gekozen systemen. Wel is deze ondersteuning zo veel mogelijk geoptimaliseerd voor de systemen zodat er maximaal gebruikt wordt van de functionaliteiten van het systeem, dit houdt in dat er bijvoorbeeld bij het opstellen van de range queries rekening gehouden wordt met het aantal records dat nodig is.

De 2 testen, beschikbaarheidstest en consistentie test, worden op verschillende manieren geïmplementeerd.

Beschikbaarheidstest De beschikbaarheidstest wordt geïmplementeerd door middel van event support, hiermee kan er op vooraf gedefinieerde momenten een bepaald Unix commando uitgevoerd worden. De configuratie gebeurt met behulp van een XML bestand met de parameters van 4.2 die meegegeven wordt aan de parameter $eventFile$, de output komt in het logbestand met de elementen van tabel 4.3.

Met behulp van deze uitbreiding zullen de beschikbaarheidstesten nadien uitgevoerd kunnen worden. Er zal gekeken worden naar de verandering in vertraging op een query waarmee kan bekeken worden of het systeem nog beschikbaar is.

Naam	**eenheid**
ID	String
Starttijdstip	milliseconden
Commando	String

Tabel 4.2: Configuratie van event support

Naam	**eenheid**
ID	String
Starttijdstip	milliseconden
Duur van de actie	microseconden
Gestart?	Boolean
Beëindigd?	Boolean
Exit code	Integer

Tabel 4.3: Uitvoer van event support

Naam	eenheid	Omschrijving
consistencyTest	Boolean	Het activeren van de consistentie test
addSeparateWorkload	Boolean	Het toevoegen van een een extra belasting
starttime	Milliseconden	Het moment dat de consistentie testen beg
readThreads	Integer	Het aantal lees gebruikers
consistencyDelayMillis	Milliseconden	Het interval waarin een lees gebruiker opn
newrequestperiodMillis	Milliseconden	Het interval waarin een schrijf gebruiker o
readProportionConsistencyCheck	Float $(0 \leq x \leq 1)$	Het percentage van schrijfacties schrijven
updateProportionConsistencyCheck	Float $(0 \leq x \leq 1)$	Het percentage van schrijfacties updaten i
stopOnFirstConsistency	Boolean	Stoppen zodra de eerste keer een correct r
maxDelayConsistencyBeforeDropInMicros	Microseconden	De maximale afwijking dat de eigenlijke st
timeoutConsistencyBeforeDropInMicro	Microseconden	De maximale tijd dat een leesactie opnieu

Tabel 4.4: Configuratie van de consistentie testen

Naam	eenheid	Omschrijving
Tijd	Microseconden	Het moment dat de schrijfactie moest starten
GebruikersID	R/W-Integer	Het id van de gebruiker (W-0, R-0, R-1, ..)
Start	Microseconden	Het moment dat actie is begonnen
Vertraging	Microseconden	De tijd dat de actie heeft geduurd
Waarde	String	De gelezen of geschreven waarde

Tabel 4.5: Uitvoer van een enkel query in de consistentie testen

Consistentie testen Voor de consistentie testen is er een extra module geïmplementeerd die dit gedrag uitvoert. In deze uitwerking leest de schrijver niet zijn eigen data, al zou dit eenvoudig mee geïmplementeerd kunnen worden, dit is niet getest omdat het niet nodig was in deze testen. De testen kunnen uitgebreid geconfigureerd worden om enkel te testen wat nodig is: een overzicht van de configuratie parameters, uitgezonderd de locatie van de logbestanden, is te vinden in tabel 4.4. Voor elke uitgevoerde query, wordt een record aangemaakt met de data van tabel 4.5.

De code van deze testen is beschikbaar op GitHub in `https://github.com/thuys/ycsb` en te gebruiken onder Apache 2.0 licentie. Check

4.4 IMP: Installatie van de DBMS's en testsoftware

Voor de installatie van de verschillende systemen, is gekozen om te automatiseren met Integrated configuration Management Platform (IMP) beschreven in [36]. Met behulp

Stoppen

Wat	Commando
Zachte stop	service {{service-name}} stop
Harde stop	kill -KILL {{process Id}}
Netwerk onderbreken	iptables -A OUTPUT -d 0.0.0.0/0 -j DROP

Heropstarten

Wat	Commando
Zachte start	service {{service-name}} restart
Harde start	service {{service-name}} restart
Netwerk herstellen	iptables -D OUTPUT 1

Speciale commando's

Wat	Commando
Pgpool-II (Online recovery)	/usr/local/bin/pcp_recovery_node -d 10 {{pgpool host}} {{port}} {{gebruikersnaam}} {{wachtwoord}} {{node nummer}}

Tabel 4.6: Commando's voor het stoppen en starten in de verschillende modes van de beschikbaarheidstesten.

YCSB!

HBase

MongoDB

Pgpool-II (PostgreSQL)

4.5 Uitvoeren van de calibratie en testen

4.6 Verzamelen en analyse van de testresultaten

Hoofdstuk 5

Observaties

Hoofdstuk 6

Analyse van de resultaten

Hoofdstuk 7

Conclusie

Bijlagen

Bibliografie

[1] David Bermbach en Stefan Tai. „Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior". In: *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*. ACM. 2011, p. 1.

[2] Kurt Bollacker e.a. „Freebase: a collaboratively created graph database for structuring human knowledge". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, p. 1247–1250.

[3] Dhruba Borthakur. „The hadoop distributed file system: Architecture and design". In: *Hadoop Project Website* 11 (2007), p. 21.

[4] Mike Burrows. „The Chubby lock service for loosely-coupled distributed systems". In: *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, p. 335–350.

[5] Fay Chang e.a. „Bigtable: A distributed storage system for structured data". In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), p. 4.

[6] 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.

[7] Edgar F Codd. „A relational model of data for large shared data banks". In: *Communications of the ACM* 13.6 (1970), p. 377–387.

[8] Brian F Cooper e.a. „Benchmarking cloud serving systems with YCSB". In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010, p. 143–154.

[9] Jeffrey Dean en Sanjay Ghemawat. „MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), p. 107–113.

[10] Ramez Elmasri en Shamkant Navathe. *Fundamentals of Database Systems*. 6th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0136086209, 9780136086208.

[11] Lars George. *HBase: the definitive guide*. "O'Reilly Media, Inc.", 2011.

[12] Sanjay Ghemawat, Howard Gobioff en Shun-Tak Leung. „The Google file system". In: *ACM SIGOPS Operating Systems Review*. Deel 37. 5. ACM. 2003, p. 29–43.

[13] Wojciech Golab e.a. „Eventually consistent: not what you were expecting?" In: *Communications of the ACM* 57.3 (2014), p. 38–44.

[14] Jim Gray. „Data Management: Past, Present, and Future". In: *arXiv preprint cs/0701156* (2007).

[15] J Hugg. *Key-value benchmarking*. 2010. URL: http://voltdb.com/blog/voltdb-benchmarks/key-value-benchmarking/ (bezocht op 06-07-2014).

[16] Patrick Hunt e.a. „ZooKeeper: Wait-free Coordination for Internet-scale Systems." In: *USENIX Annual Technical Conference*. Deel 8. 2010, p. 9.

[17] ZikaiWang James Chin. *HBase: A Comprehensive Introduction*. 2011. URL: http://cs.brown.edu/courses/cs227/archives/2011/slides/mar14-hbase.pdf (bezocht op 10-07-2014).

[18] Christos Kalantzis. *A Netflix Experiment: Eventual Consistency != Hopeful Consistency*. Planet Cassandra. 2013. URL: http://planetcassandra.org/blog/post/a-netflix-experiment-eventual-consistency-hopeful-consistency-by-christos-kalantzis/ (bezocht op 06-07-2014).

[19] 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.

[20] Todd Lipcon. „Design Patterns for Distributed Non-Relational Databases". In: *Design Patterns for Distributed Non-Relational Databases* (2009).

[21] Cary Millsap. *Optimizing Oracle Performance*. "O'Reilly Media, Inc.", 2003.

[22] *MongoDB Manual*. URL: http://docs.mongodb.org/manual/ (bezocht op 10-07-2014).

[23] *MongoDB: Replication Introduction*. URL: http://docs.mongodb.org/manual/core/replication-introduction/ (bezocht op 10-07-2014).

[24] *MongoDB: Sharding Introduction*. URL: http://docs.mongodb.org/manual/core/sharding-introduction/ (bezocht op 10-07-2014).

[25] Swapnil Patil e.a. „YCSB++: benchmarking and performance debugging advanced features in scalable table stores". In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM. 2011, p. 9.

[26] *Pgpool-III: User manuel*. URL: http://www.pgpool.net/docs/latest/pgpool-en.html (bezocht op 10-07-2014).

[27] Pouria Pirzadeh, Junichi Tatemura en Hakan Hacigumus. „Performance evaluation of range queries in key value stores". In: *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. IEEE. 2011, p. 1092–1101.

[28] A. Popescu. *NoSQL benchmarks and performance evaluations*. 2010. URL: http://nosql.mypopescu.com/post/734816227/nosql-benchmarks-and-performance-evaluations (bezocht op 06-07-2014).

[29] 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).

[30] Postgresql. *PostgreSQL - Replication, Clustering, and Connection Pooling.* Okt 2013. URL: http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling (bezocht op 03-02-2014).

[31] Tilmann Rabl e.a. „Solving big data challenges for enterprise application performance management". In: *Proceedings of the VLDB Endowment* 5.12 (2012), p. 1724–1735.

[32] Arnaud Schoonjans. „Een critische evaluatie van beschikbaarheid in gedistribueerde opslag systemen". KU Leuven, 2014.

[33] Ben Scofield. *NoSQL – Death to Relational Databases(?)* Jan 2010. URL: http://www.slideshare.net/bscofield/nosql-codemash-2010 (bezocht op 03-02-2014).

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

[35] Bogdan George Tudorica en Cristian Bucur. „A comparison between several NoSQL databases with comments and notes". In: *Roedunet International Conference (RoEduNet), 2011 10th.* IEEE. 2011, p. 1–5.

[36] Bart Vanbrabant. „A Framework for Integrated Configuration Management of Distributed Systems (Een raamwerk voor geïntegreerd configuratiebeheer van gedistribueerde systemen)". Proefschrift. 2014. URL: https://lirias.kuleuven.be/handle/123456789/453199.

[37] Hiroshi Wada e.a. „Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective." In: *CIDR.* Deel 11. 2011, p. 134–143.

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: 681.3

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: Prof. dr. ir. Tias Guns,
Prof. dr. ir. Christophe Huygens

Begeleider: Dr. ir. Bart Vanbrabant
Dr. Bert Lagaisse