OND6: Toepassing van niet-traditionele databases binnen een Casus

David Diks

June 11, 2018

Contents

1 Probleemstelling

Standaard gebruiken projecten SQL databases zoals MySQL, OracleDB en PostgresQL. Deze databases hebben zijn de standaard voor een reden maar komen nog altijd met een verscheidenheid aan problemen, meest voorkomend daarvan is Scaling en Replicatie. Vanwege de inherente structuur van deze databases kan hier niet omheen gewerkt worden. [1]

De opkomst van NoSQL en MongoDB heeft vernieuwend gewerkt en zorgt voor de nieuwe generatie van applicaties en heeft het langschap ge-opend voor nieuwe databasedesigns.

Hierbij stellen wij de volgende vraag: **Wat is een gepaste database architectuur voor verschillende datasets binnen één systeem.**

Hierbij stellen wij de volgende deelvragen;

1. Hoe kunnen database architechturen gecategoriseerd worden? (Veld, Bieb)
2. Hoe scalable zijn deze gevonden architecturen? (Veld, Bieb)
3. Zijn er bruikbare implementaties van deze databases? (Bieb, Werkplaats)
4. Passen deze architecturen bij onze casus? (Werkplaats, Lab)

2 Casus: Rekeningrijden

De gebruikte casus is het Fontys-project, Rekeningrijden. Binnen deze applicatie zijn verschillende systemen ontwikkeld met behulp van Java Enterprise Edition en geassociëerde webtechnieken. Hierbij bestaat het project zelf uit meerdere kleine applicaties die samen een geheel systeem vormen.

De opdracht zelf bestaat uit het administreren van een kilometerheffingssyssteem voor voertuigen. Per auto wordt de afgelegde afstand bijgehouden en opgeslagen. Deze informatie gebruikt om de eigenaar van dit voertuig facturen te sturen. Daarnaast moet de gebruiker en overheid inzicht hebben in deze facturen.

3 Oplevering A

Dit hoofdstuk bevat de vragen van de eerste oplevering

3.1 Hoe kunnen database architechturen gecategoriseerd worden?

Databases zijn veelal te onderscheiden op basis van 2 categoriëen. De implementatie van relaties en het gebruik van schema's. Deze vormen de volgende kruistabel.

	Schema	Document
Relationeel	Relationele databases	Graph Databases
Non-relationeel	Time Series database	NoSQL database

3.1.1 Relaties

Een belangrijk onderdeel van data-opslag zijn de relaties die de stukken met elkaar hebben. Echter ziter een debat in hoe verre deze relatiesafgedowngen moeten worden. Aan de ene kant staan Relationele en Graph databases. Deze databases gebruiken relaties als harde paramaters. [2]

Hier tegenover staan de non-relationele databases zoals MongoDB en Time-series. MongoDB dwingt geen relaties af enkan hier geen garanties aan stellen. Een Time-series database bevat geen concept van relaties en slaat enkel datapunten op.

3.1.2 Schema

Een schema biedt houvast aan voor de database. Hierin worden modellen en tabellen gedefinëerd. Relationele en Time-Series databases maken hier veel gebruik van om garanties te geven. Hier tegenover staan de Schemaless databases. Deze databases hebben geen regels voor properties en laten een ontwikkelaar alles invoegen. Schemaless leidt tot een snellere ontwikkeling van software en zorgt voor hogere deliverability, terwijl Schemas datagaranties bieden.

3.2 Scalability van oplossingen

Scalen is het toevoegen van extra rekenkracht om sneller data te kunnen verwerken. Deze is op te delen in **verticaal** en **horizontaal**.

3.2.1 Verticaal

Verticaal schalen is het toevoegen van meer rekenkracht op een enkele machine doormiddel van meer RAM of een krachtigere CPU. Vanwege het feit dat dit dezelfde machine is zorgt ervoor dat de ontwikkelaar en onderhouder geen rekening hoeven te houden met clustering en replicatie. Aangezien schema databases hier al sneller problemen mee hebben is het slim om verticaal te schalen voor schema databases.

3.2.2 Horizontaal

Horizontaal schalen is het toevoegen van extra machines om rekenkracht te krijgen. Queries worden hierdoor parallel gedraaid en zijn zo sneller klaar. Het grotere nadeel hiervan is dat er een replicatie opgezet moet worden, aangezien de databases anders incorrecte queries uit gaan voeren.

Replicatie is moeilijk en intensief voor schema-databases aangezien ieder bit gekopiëerd moet worden. Schemaless databases hebben hier echter geen problemen mee en maken gebruik van sharding, waarbij iedere database een klein stuk van de complete dataset heeft. Vanwege deze reden is het makkelijker om schemaless databases horizontaal te schalen.

4 Oplevering B

Deze hoofdstuk bevat de laatste twee deelvragen en geven uiteindelijk een antwoord op de hoofdvraag.

4.1 Zijn er bruikbare implementaties van deze databases?

Met de groei van technologie is er genoeg rekenkracht om de verscheidene concepten uit te voeren. Hierbij zijn de volgende implementaties gebruikt;

1. Nosql: MongoDB - Grootste NoSQL implementatie
2. Graph Database: Neo4J - Weidverspreide Graph DB met veel documentatie
3. Time Series: OpenTSDB - Heeft een docker image en documentatie.

4.2 Is er een betere aanpak mogelijk in geval van de casus?

Om antwoord op deze vraag te geven, zijn er 3 casussen gedefiniëerd en uitgewerkt. Per onderdeel is er een iPython notebook beschikbaar met de resultaten.

4.2.1 Casus 1

Stel dat

1. 1000 auto's
2. 5000 tot 10000 locaties per auto
3. 500 personen met iederen een random hoeveelheid auto's

Hoe lang duurt het om voor een enkele auto de locatiegeschiedenis van de eigenaar op te halen?

MongoDB blinkt uit in snelheid, met gemiddeld 0.14 seconden per query. Echter is de query zelf zeer complex en bevat meerdere joins en branches en is conceptueel harder te begrijpen. De Neo4J query is een enkele regel en makkelijker te begrijpen, maar is 0.3 seconden in totaal.

4.2.2 Casus 2

Stel dat

1. 1000 auto's
2. 100 personen
3. Iedere gebruiker heeft 20 verschillende auto's gehad

Haal voor een random voertuig de eigenaarsgeschiedenis op.
MongoDB is alweer een stuk sneller met 0.05 seconden gemiddeld. De query zelf is ook begrijpelijk. Dit tegenover Neo4J, die 0.44 seconden bezig is met de query. De query zelf heeft echter een vergelijkbare complexiteit.

4.2.3 Casus 3

Stel dat

1. 100 auto's
2. 10.000 locaties voor deze auto's

Hoe lang duurt het om deze locaties op te slaan?

Neo4J loopt behoorlijk achter met 71 seconden en valt dus buiten de race voor snel data te ingesten. OpenTSDB volgt met 4 seconden en Mongo met 0.27 seconden. Voor het querien en inserten is OpenTSDB echter wel koning, aangezien dit hier met een enkele regel gedaan kan worden.

5 Conclusies en Evaluatie

Er is een verscheidenheid aan databaseoplossing, waarbij iedere soort zelf de problemen oplost. Binnen de casus van ons is er ook een mogelijkheid om data over meerdere soorten database te verspreiden. Zo kan de relationele data weggewerkt worden met Neo4J en kunnen de locaties het beste via OpenTSDB verwerkt worden.

5.1 Evaluatie

Terugkijkend op het onderzoek is dit een logische conclusie. Ieder stuk software wordt ontworpen voor problemen op te lossen en de problemen in deze casus zijn zeer nichè. Echter is de software besproken in dit onderzoek speciaal ontwikkeld voor deze use-cases.

Zelf had ik liever een uitgebreidere casus gehad waar de complexiteit echt naar voren komt. Echter is tijd een grote limitatie. Ik ben wel blij dat ik heb mogen experimenteren met Time-Series en Graph databases, aangezien deze nooit bruikbaar zijn bij Fontys-projecten.

Bibliography

[1] Andrew Bust. Rdbms vs. nosql: How do you pick?, juni 2018.

[2] Jim Webber Ian Robinson and Emil Eifrem. *Graph Databases*.

A Onderzoeksbestanden

Om meer uit te vinden over de casussen en de uitvoering hiervan, is de code beschikbaar gesteld. Kijk hiervoor op https://github.com/s61-austria/medienlaboratorium