Distributed Databases Summary

Dylan Cluyse

Contents

1. Een distributed database.

Wat?

- Verschillende componenten op een netwerk die met elkaar communiceren.
- Een systeem met het doel om data beschikbaar te maken. Data dat later kan worden gelezen of geschreven.

– De mate van beschikbaarheid doet er niet toe.

Waarom?

Horizontale schaalbaarheid	Verticale schaalbaarheid
Meer onderdelen toevoegen.	Meer capaciteit toevoegen aan een onderdeel.

- Moore's law: Het aantal transistoren verdubbelt per achttien maanden. Met andere woorden is er een snelle nood aan nieuwe hardwarematerialen.
- Fouttolerantie toelaten –> Je moet de taken onderling verdelen.
- Latency verminderen –> Kies voor een systeem dat dichter bij de client ligt. Bijvoorbeeld online gameservers zijn opgedeeld per regio's.

Kenmerken

_-*-_	_-*-_
Geen gedeeld geheugen. Iedere verwerkingseenheid heeft een eigen geheugen.	Onderling worden er berichten naar elkaar gestuurd.
Componenten zijn niet bewust van wat de andere componenten nu aan het doen zijn.	Fouttolerant zijn.

Soorten systemen

Parallel systeem	Gedistribueerd systeem
Verschillende verwerkingseenheden met een **gedeeld geheugen**. Makkelijker te ontwikkelen is.	Verschillende verwerkingseenheden met **elk een eigen geheugen**. De andere componenten zijn **onbewust** van wat de andere onderdelen aan het doen zijn. Er wordt **onderling berichten** met elkaar verstuurd.
Het probleem is dat het systeem **geen redundantie** biedt.	**Shared-nothing architecture:** Niets wordt onderling gedeeld. De enige manier van communicatie is door middel van boodschappen.

CAP-Theorem:

- Een theorie dat zegt wanneer een partitioneringsfout voorkomt, het systeem ofwel beschikbaar is ofwel zich in een consistente staat bevindt.

  - Consistency van een relationele databank is véél groter dan de consistency van een gedistribueerde databank.
  - Het is een blijvende afweging tussen consistency en beschikbaarheid.

Struikelblokken:

- Split-brain scenario: de ene helft denkt het ene en de andere helft denkt het andere. Bijvoorbeeld: Het ene systeem denkt dat een bestand verwijderd is, terwijl het andere systeem denkt dat het nog bestaat.
- Consistency en structuur raken snel verloren.
- Testen wordt moeilijker.
- De oorzaak van traagheid achterhalen wordt complexer: zowel hardware als software kunnen een rol spelen.

Fabels over gedistribueerde systemen:

Fabel	Beredenering
Er is geen latency.	Latency is wel aanwezig. Enkel is de deze sterk minder naargelang de locatie van het systeem. het duurt een tijd vooraleer een bericht toekomt op een systeem.
De bandbreedte is oneindig.	De bandbreedte op zowel client als het distribueerd systeem is beperkt.
Het netwerk is veilig.	Toegang tot het netwerk blijft iets waar je rekening mee moet houden.
De netwerktopologie blijft hetzelfde.	Computers en hardware kan worden toegevoegd. Zo verandert alles binnen een netwerk op een dynamische manier.
De transportkost van data is nul.	Data transporteren van begin- naar eindpunt vergt een inspanning qua energie en rekenkracht.
Het netwerk is homogeen.	Alle onderdelen binnen een netwerk kunnen variëren van eigenschappen. Sommige delen van het netwerk kunnen snel zijn, sommige traag.

Vier algemene problemen:

1. Partial failures
2. Niet-betrouwbare netwerken
3. Niet-betrouwbare tijdsindicaties
4. Onderlinge onzekerheid

Partial failure:

Het wegvallen van systemen.	Onopmerkbaarheid
Sommige onderdelen van een netwerk kunnen werken, terwijl andere onderdelen down zijn of niet meer in gebruik. Hoe meer computers, hoe groter de kans dat één systeem (heel even) wegvalt.	De andere systemen zien niet wanneer een systeem wegvalt. Het probleem wordt pas opgemerkt wanneer er geen antwoord is. De oorzaak kennen we niet. Dit kan liggen aan: overbelasting, defunct, te traag vergeleken met andere systemen, etc.

Niet-betrouwbare netwerken

Asynchrone verbinding	Oorzaak achterhalen	Exponential back-off	Sneeuwbaleffect
Boodschappen worden verstuurd zonder tijdsaannames. Het maakt de systemen niet uit hoe lang ze moeten uitvoern of wanneer het bericht zal arriveren.	Het is moeilijk om de oorzaak te achterhalen. Er zijn drie mogelijke problemen: een probleem tussen zender en ontvanger, de ontvanger kan niets ontvangen of de ontvanger kan niets versturen.	De tijd waarop je wacht op een antwoord moet je exponentieel vergroten. Begin met twee seconden wachten, daarna vijf seconden, daarna tien seconden, … tot maximaal vijf minuten. Vermijd te snel opnieuw opstarten. Zo maak je het enkel erger.	Eenmaal de capaciteit van de wachtrij wordt behaald, dan zal de vertraging (in seconden) exponentieel verhogen. IRL-voorbeeld: files.

Niet-betrouwbare tijd

Real-time tijd.	Monotonische tijd	Causality & Consensus
Real-time tijd zijn klokken die gesynchroniseerd worden met het gebruik van een gecentraliseerde server.	Monotonische klokken zijn klokken die op een vast moment starten en enkel vooruit gaan. Er is geen synchronisatie. Leap-seconden: een minuut is niet altijd 60 seconden. Soms kan dit 59 of 61 seconden zijn.	Causality is achterhalen wanneer een event werd uitgevoerd. Concensus is wanneer alle knopen (of nodes) met elkaar overeenkomen bij een beslissing.

5. Kafka

Check out Learn Apache Kafka for Beginners.

Transporting data

Source: Maakt data. Target: Verbruikt data.

w/	wo/
Kafka functioneert hier als tussenpersoon. De tussenpersoon zal het verkeer naar de targets regelen. Zo moeten de sources niet verbonden zijn met alle targets. Dit zorgt voor een fouttolerant en veerkrachtig systeem. Kafka staat ook sterk bij horizontale schaalbaarheid.	Iedere source is verbonden met ieder target. Dit is de meest verbruikende manier van de twee. Hier moet iedere source rekening houden met protocollen, doorvoer, etc.
m + n	m x n

Topics:

Topics: * Stream van data. Meerdere mogelijk. * Naamgeving. * Wordt opgedeeld in een **vast aantal partities**:

Partities:

- Doorvoer verbeteren
- Een bestand op een lokaal FS.
- Append-only. Je kan enkel messages op het einde toevoegen.
- Offset = 0 : Allereerste bericht.
- Offset van de laatste partitie = n
- Bij vergissing: pech!

    - Je kan het niet verwijderen of aanpassen.
    - Een dubbele actie (bv.: twee messages rond een aankoop): Je moet een derde message sturen om de dubbel ongedaan te maken.
- Ordening is niet gesorteerd!
    - min -> max
- Je kan het aanpassen (?), maar er hangen hier nadelen aan.

Kenmerken

- Immutable
- Limited: Volgens de default policy worden berichten ouder dan een week verwijderd.
- Je specifieert de topic waar het bericht naartoe moet, niet de partition.
    - Partitie is willekeurig –> Load-balancing

Broker

= Computer

- Elke broker een ID geven.
- Elk bestaat uit partities
- –> weinig controle over de brokers: geen master/slave verhouding.
- Als je één broker kent, dan kan je verbinding maken met alles binnen de cluster.
- Per standaard: drie brokers.
- één broker ook mogelijk: geen schaalbaarheid.

Partitie toekennen aan broker(s): * Algoritme

Data replication

Het repliceren van data: * Fouttolerantie: voorkomen dat data verloren raakt als het systeem van een partitie defunct gaat. * Partitioneren verhoogt de schade bij een fout.

Replication factor: * Factor hoger dan één, maar niet te hoog! * Broker kapot –> andere broker bezit de data

Leader/followers: * Leaders hebben volledige toegang tot de data. * Moet worden aangesproken als er iets in de partitie moet worden veranderd. * Volgers hebben geen toegang. * Gedrag kan worden beïnvloed. * Een applicatie binnen dezelfde rack als een follower, van de data dat die nodig heeft, zal de volger aanspreken i.p.v. leader.

Out-of-sync: De volger beschikt niet meer over de meeste recente data.

Fetch requests: Geef mij alles dat begint vanaf deze offset. * De replica weet hoeveel offsets die achterloopt op de leider. * **In-sync**: er is geen verschil tussen de replica en de leider. Enkel zij komen in aanmerking om leider te worden. *

Als de leider geen fetch request ziet voor meer dan 10 seconden == Out-of-Sync (Mortis)

Producer * Schrijft/verstuurt data naar de topic(s). * Zal automatisch opnieuw proberen.

Hoe achterhalen of een bericht is toegekomen:	–	
Bericht versturen + schietgebedje	ack=0	Unreliable, maar snel.
Bericht versturen + wachten op bevestiging	acks=1	Deels geruststellend.
Bericht versturen en wachten tot de leider + in-sync replicas het bericht hebben ontvangen.	acks=all	Volledige geruststelling, maar gevaarlijk als er géén enkele replica in-sync is. Geen in-sync replicas: enkel de leader wordt geüpdatet. Als er géén in-sync replica's zijn is alles *'goed'* verlopen. Dit voorkom je door *min in-sync replica's* op twee te plaatsen. Nooit hetzelfde getal als je replication factor (vb.: 2 & 2), want dan verwacht je dat je geen trage volgers hebt.

Consumers:

- Data lezen van de topic(s)
- Toekennen aan partitie:
  - Speciale topic binnen Kafka: Consumer offset.

Consumer offset: * De staat van topics. * Logboek: "ik heb de messages t.e.m. 50 gelezen". * Achterhalen vanaf waar de consumer berichten moet verwerken.

Delivery Semantics

- "Wat kan een consumer doen om het bericht te verwerken?"
- "Wanneer vertel je Kafka dat je klaar bent met het verwerken van een bericht?"

Zookeeper

- leader-follower architecture

Labo

We moeten hier het poortnummer 19092 gebruiken voor Kafka1.

```
kafka-topics --bootstrap-server kafka1:19092 --list
kafka-topics --bootstrap-server kafka2:19093 --list
kafka-topics --bootstrap-server kafka3:19094 --list
```

Het maakt niet uit bij welke broker. De actie zal altijd werken.

Aanmaken:

```
kafka-topics --create --topic lecture --partitions 3 --replication-factor 3
```

Omschrijven:

```
kafka-topics --bootstrap-server kafka1:19092 --describe --topic lecture
```

Nieuwe messages toevoegen:

```
kafka-console-producer --bootstrap-server kafka1:19092 --topic-lecture
```