Gedistribueerde Systemen
iHome

Tim Leys
Lisette van Leeuwen

3 januari 2017

In dit verslag wordt kort besproken welke architectuur, algoritmes en libraries wij hebben gebruikt. Ook wordt beschreven hoe het programma gebruikt moet worden.
Devices die zelf een controller kunnen starten, zoals een user of een fridge, worden in dit verslag smart devices genoemd.

1 Communicatie

Tussen een client en een server zijn twee types communicatie mogelijk, namelijk synchroon en asynchroon. Bij synchroon moet de communicatie eerst voltooid zijn, voordat de code verder kan gaan. Bij asynchrone communicatie is dit niet het geval. Als de ontvanger niet actief is, zal bij asynchrone communicatie de zender wachten totdat de ontvanger terug actief is.
Wij hebben voornamelijk synchrone communicatie gebruikt. Enkel bij leader election gebruiken we asynchrone communicatie. Bij leader election wordt steeds het hoogste ID naar de volgende client in de ring verstuurt. Indien we synchrone communicatie zouden gebruiken, kan de functie die de election start pas verder gaan als een leader is verkozen. Met asynchrone communicatie zetten we meerdere threats op, waardoor de startende functie verder kan zodra het een bericht heeft verstuurd naar de volgende client in de ring.

2 Replicatie

Elk smart device heeft een instantie van de klasse Controller. Deze wordt gebruikt om de rol van controller over te nemen als de originele server uitvalt. Wanneer dit gebeurt, moet de controller van de nieuwe leader wel de correcte data bevatten. Met behulp van replicatie zorgen wij hiervoor.
We zijn ervan uitgegaan dat AVRO inkomende berichten sequentieel en FIFO gewijs verwerkt.
Zodra de data van de server wijzigt, moeten de smart devices hiervan op de hoogte gebracht worden, zodat zij de data van hun controllers kunnen aanpassen. Direct na een wijziging stuurt de server zijn data naar de smart devices. Dit is push-based. Alle databanken van de smart devices worden geüpdatet voordat de functie returned en de remote call eindigt.
Enkel de werkende server zal zijn data versturen naar alle geconnecteerde smart devices.

We kunnen hier spreken van sequentiële consistentie. Alle operaties worden in dezelfde volgorde verwerkt door de server als waarin ze aankomen. Doordat de controllers in de smart devices worden geüpdatet vooraleer er volgende berichten verwerkt worden, geldt dat de controllers in de juiste volgorde worden geüpdatet mits een kleine delay.
Ook als een nieuw device aan het systeem wordt geconnecteerd, stuurt de server zijn data door naar de smart devices. Deze update kan echter niet naar het nieuw geconnecteerde device gestuurd worden, omdat die nog niet zijn SaslSocketServer heeft gestart. Zodra het device klaar is om berichten te ontvangen, stuurt het een pull-based bericht naar de server. De server stuurt dan alsnog de huidige data naar het device.

3 Fault tolerance

We willen dat de server ten allen tijde weet welk device nog online is, zodat er geen verbindingen worden gemaakt met gecrashte devices. Om dit te realiseren hebben we de failure detection methode heartbeat geïmplementeerd. Alle devices sturen drie keer per seconde een bericht naar de server, zodat de server weet dat het device nog online is.
Indien de server crasht, willen we dat een fridge of user de rol van controller overneemt. De user of fridge met de hoogste ID moet verkozen worden. Om te bepalen welke actieve client de hoogste ID heeft, wordt het ring algoritme van Chang-Roberts gebruikt. Zoals hiervoor beschreven, gebruiken we hier asynchrone communicatie voor.
Bij het ring algoritme worden alle devices in het systeem gebruikt. Een light of temperature sensor kan echter nooit leader worden, omdat zij geen smart devices zijn. Deze devices sturen het binnenkomende bericht door naar de volgende device in de ring. Aangezien zij niet een instantie van de controller bij houden, maar wel moeten weten welk device de volgende in de ring is, bewaren zij een lijst van geconnecteerde devices. Met behulp van replicatie, zoals hierboven beschreven, wordt deze lijst regelmatig geüpdatet.
Tegengesteld aan wat er in de opgave staat, kan een smart device dat uitverkozen is tot leader nog steeds gebruikt worden als smart device. Een user kan bijvoorbeeld nog steeds een connectie met een fridge maken. De controller draait dan op de achtergrond.
Als de originele server gecrasht is, maar later terug online komt, willen we dat de originele server de rol van controller terug overneemt. Om te bepalen of de originele server weer online is, stuurt de leader drie keer per seconde een bericht richting de server volgens de ping methode. Indien het bericht aankomt, weten we dat de server weer online is. Direct wordt dan de data van de leader naar de server gestuurt en de server stuurt een bericht naar alle devices dat het terug online is. De communicatie verloopt dan weer hetzelfde als de server niet was uitgevallen.

4 Code uitvoeren

Naast de benodigde libraries voor AVRO hebben wij ook een JSON library gebruikt. In plaats van `slf4j-simple.jar` gebruiken wij `slf4j-nop.jar`.

Deze library maakt geen gebruik van logging, waardoor AVRO geen berichten in de terminal print als er een connectie wordt geopend of gesloten.

In de bijgeleverde zip zitten vijf jar files. Dit zijn runnable jars en kunnen uitgevoerd worden met het commando `java -jar filename.jar`. Eerst moet `Controller.jar` gestart worden. Deze vraagt om een IP adres. Vervolgens krijg je de mogelijkheid tussen silent mode en interface mode. In interface mode kan direct data van de server opgevraagt worden. Dit is niet mogelijk in silent mode.
Elk device vraagt om zijn IP adres en het IP adres van de huidige controller. Als de originele controller crasht, wordt leader election gestart. Zodra een leader is gekozen, wordt diens IP adres in de terminal geprint. Dit IP adres moet vervolgens gebruikt worden als een nieuw device wordt toegevoegd.