KU LEUVEN

FACULTEIT INDUSTRIËLE
INGENIEURSWETENSCHAPPEN
TECHNOLOGIECAMPUS GENT

Labo: Communicatietechnologieën

Gedistribueerde systemen 1

Robbe De Groeve & Jerry Xiong

4ELICTsa

november 2020

Inhoud

Inleiding

In dit verslag zullen er drie communicatietechnologieën voor gedistribueerde systemen overlopen en geïmplementeerd worden aan de hand van een chatsysteem.

1 Technologieën

Voordat ze geïmplementeerd kunnen worden zullen de drie communicatiemethoden eerst overlopen en vergeleken worden. De volgende technologieën worden hieronder overlopen: sockets, RMI en gRPC.

1.1 Sockets

Dit is de basis manier van communicatie tussen gedistribueerde systemen. Sockets maken gebruik van het “message-oriented” model: ze zijn zowel een eindpunt waar een applicatie data op kan plaatsen om deze door te sturen op het onderliggende netwerk als een ingang waar de inkomende data ingelezen kan worden. Een socket wordt geïdentificeerd door een combinatie van het IP-adres en een poortnummer.

1.2 RMI

Het nadeel aan sockets is dat ze enkel data kunnen versturen en ontvangen. Hierbij zal de programmeur dus zelf altijd moeten implementeren wat er zal gebeuren met die data en in welke vorm ze verstuurd wordt. Om een grotere graad van transparantie te bekomen bij de communicatie, werd er een nieuw communicatiemodel bovenop de sockets ontwikkeld, namelijk Remote Procedure Calls. Conceptueel betekent dit dat een proces van een systeem een procedure of methode van een ander proces op een ander systeem oproept. Hierbij is het de bedoeling dat dit lijkt op een gewone oproep van een methode zonder dat de programmeur zich bezig hoeft te houden met de communicatie over het onderliggende netwerk.

Een voorbeeld van dit communicatiemodel is Remote Method Invocation of RMI. RMI wordt gebruikt om methoden van objecten in een andere Java Virtual Machine aan te roepen. Deze JVM kan zich hierbij zowel op de lokale host als op een andere host bevinden. Het nadeel van dit protocol is dat het vooral voor communicatie tussen Java-programma’s ontwikkeld is en dus niet een universeel protocol is.

1.3 gRPC

Een ander voorbeeld van een implementatie van het Remote Procedure Call communicatiemodel is gRPC. gRPC of google Remote Procedure Calls is een open source RPC-systeem ontwikkeld door Google. Het maakt gebruik van Protocol Buffers als Interface Definition Language en als het formaat voor het serialiseren/deserialiseren van berichten. Net zoals RMI biedt het de mogelijkheid om procedures op te roepen op de server.

Toch zijn er een aantal verschillen tegenover RMI. Het eerste grote verschil is dat de gRPC wel een universeel protocol is. De verschillende procedures en berichten die tussen de client en server uitgewisseld worden, kunnen automatisch in verschillende ondersteunde programmeertalen gegenereerd worden uit de protocol buffers, in tegenstelling tot enkel een Java interface bij RMI. Een tweede voordeel is dat gRPC streaming langs één of beide kanten ondersteunt waardoor eenvoudig zowel client naar server als server naar client asynchroon data gestuurd kan worden.

2 Chatsysteem

Voor de implementatie van deze communicatiesystemen zal er een chatsysteem gebouwd worden waar verschillende gebruikers met elkaar kunnen chatten in een groepschat of een privégesprek kunnen starten tussen één of meerdere personen.

2.1 Algemene opbouw

Omdat alle drie de implementaties een gelijke werking moeten leveren is ervoor gekozen om het programma modulair op te bouwen om zoveel mogelijk code te kunnen hergebruiken. Bij zowel de client als de server is er een controller geïmplementeerd die het grootste deel van de logica bevat van de chatroom. Deze chatroom voorziet de volgende methoden die via een client naar de server verstuurd kunnen worden:

- Toevoegen van een gebruiker
- Verwijderen van een gebruiker
- Online gebruikers opvragen
- Sturen van een bericht
- Nieuwe chat aanmaken

Ook zal er communicatie van de server naar de verschillende clients plaatsvinden. Bij aanpassingen van een bepaalde chat of de creatie van een nieuwe wordt deze geüpdate chat naar elke geabonneerde gebruiker verstuurd.

Onder beide controllers bevindt zich dan de connection package. De code in deze package zorgt ervoor dat de calls tussen de client en server geserialiseerd/gedeserialiseerd worden volgens het gekozen protocol en aan de correcte methoden gekoppeld worden. In volgende punten zullen deze verschillende connectie implementaties overlopen worden.

2.2 Connecties

2.2.1 Sockets

Zoals reeds besproken is dit de meer basis communicatiemethode, de programmeur zal dus zelf de structuur van de data die over het netwerk gestuurd wordt moeten bepalen. Hierbij werd er gekozen om een bericht als String te versturen die als eerste element de oproepmethode bevat waarbij het volgende element een HashMap is die verschillende parameters bevat. Om dit bericht te encoderen tot een String is een aparte encoder klasse aangemaakt die de verschillende elementen in een lange string plaatst en ze van

elkaar scheidt door chars als bijvoorbeeld “,” of “/” tussen te plaatsen. Het decoderen hiervan gebeurt dan door deze chars er terug van tussen te filteren.

Een voordeel van sockets is dat ze zowel data kunnen schrijven als lezen, dus communicatie is mogelijk langs twee kanten, dit is handig voor het sturen van chat updates van de server naar de verschillende clients. Een nadeel is wel dat sockets enkel punt tot punt werken en er dus door de programmeur nieuwe sockets opgestart moeten worden bij elke gebruiker en ook nieuwe threads die de sockets gelijktijdig kunnen lezen.

2.2.2 RMI

Dit is de simpelste methode om te implementeren. Hierbij wordt er een remote interface gemaakt met de methoden van de chatroom. Deze wordt dan aan de serverkant geïmplementeerd om de calls van de clients door te geven aan de server controller. RMI zorgt hier ook zelf voor het beheer van de communicatiethreads.

De moeilijkheid van dit protocol is dat via deze interface enkel de client, methoden van de server kan oproepen. Daarom wordt er ook op elke client ook een andere remote interface geïmplementeerd die een functie bevat die de server dan kan oproepen bij een chat update.

2.2.3 gRPC

De implementatie van gRPC lijkt goed op die van RMI. Het verschil hierbij is dat de interface gegenereerd wordt uit een protocol buffer, die de server dan implementeert. Een ander verschil is dat de parameters hier niet rechtstreeks in de opgeroepen methode geschreven worden maar wel in een request bericht geplaatst worden. Ook het antwoord van de server komt toe in een reply bericht. De structuur van deze berichten is ook gedefinieerd in de protocol buffer.

Een voordeel bij deze implementatie was de ondersteuning van serverside streaming. Hierdoor kunnen chatupdates asynchroon naar de clients gestuurd worden, zonder hiervoor aparte interfaces of threads voor aan te moeten maken.

3 Besluit

In dit verslag werden drie veelvoorkomende communicatiemethoden tussen gedistribueerde systemen besproken en geïmplementeerd. Hierbij kwamen een aantal voor- en nadelen aan bod die de uiteindelijke keuze van het correcte protocol voor een bepaald doel beïnvloeden.

Sockets kunnen bijvoorbeeld gebruikt worden als men een totale controle over de berichten over het netwerk wil hebben maar dit is ten koste van de transparantie van communicatie.

RMI en gRPC zullen voor het grootste deel van de gevallen een betere keuze zijn omdat hierbij de transparantie hoger is en er dus minder aandacht aan de communicatie tussen systemen besteed moet worden. De uiteindelijke keuze tussen deze twee protocollen zal vooral bepaald worden door de gewenste programmeertaal aangezien RMI vooral voor Java bedoeld is en gRPC door veel talen ondersteund wordt.

De code en de instructies voor de uitvoering van de drie implementaties zijn te vinden op:
https://github.com/RobbeRDG/DistributedSystems1

TECHNOLOGIECAMPUS GENT
Gebroeders de Smetstraat 1
9000 GENT, BELGIË
TEL. 09 265 86 10
WWW.KULEUVEN.BE