8 JUNI 2018

PROJECT DISTRIBUTIE

VOLUNTEER COMPUTING

ASMA OUALMAKRAN
0507834
BA Computerwetenschappen

Inhoud

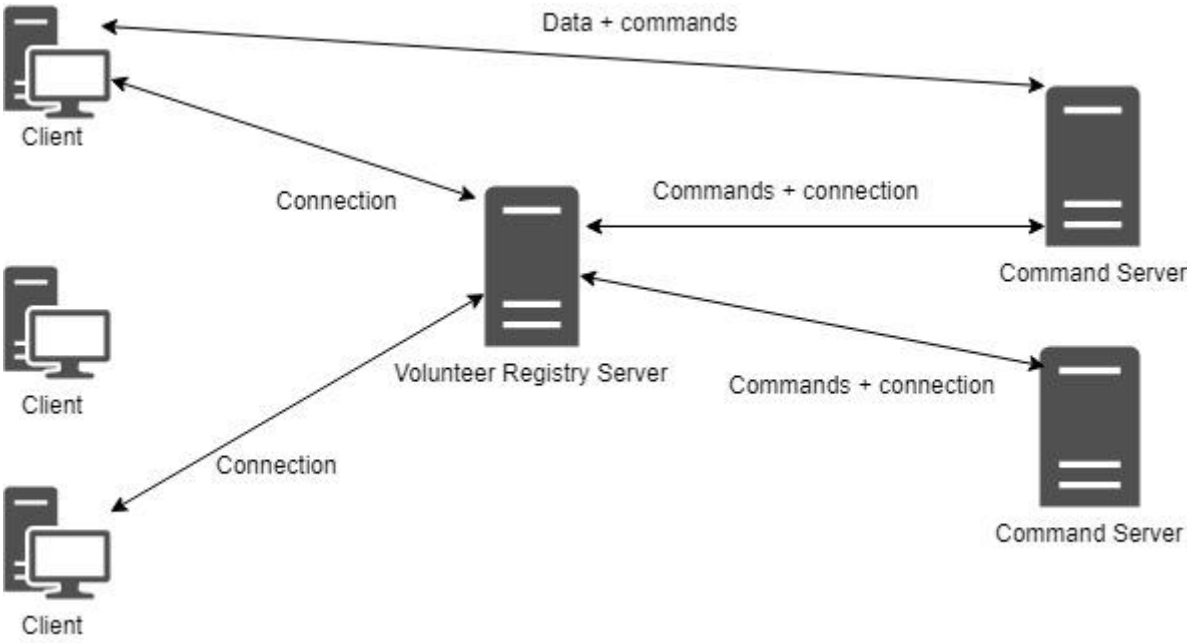
Inleiding

De opdracht bestaat uit het creëren van een volunteer computing systeem. Deze zou dan gebruikt moeten worden om analyses uit te voeren op Reddit comments. Hierbij kan er dan het algoritme gebruikt worden van het parallelisme gedeelte van het project.
Elk onderdeel van de het systeem biedt bepaalde services aan die dit mogelijk maken.

Implementatie

Er is gekozen voor een Remote Method Invocation (RMI) oplossing, dit heeft het voordeel dat alle methode oproepen makkelijk kunnen gebeuren. Dit vanwege de abstractie die het doet lijken dat het lokaal gebeurd. Bovendien maakt RMI het heel simpel om meerdere instanties van Clients en Command Servers toe te laten. Om te identificeren van welke instantie er een opdracht of data wordt gestuurd, kon dit opgelost worden door een extra parameter door te sturen.
Ook is de abstractie voor het serializen en deserializen reeds aanwezig, en moest deze dan ook niet zelf worden geschreven worden. Wel moet er rekening worden gehouden met het feit dat zelf gedefinieerde klasses niet altijd serializable zijn en deze dan aangepast moeten worden of er zelf een serialzier schrijven om dit toe te laten. Dit was het geval voor de klasse Comments. Hier is het keyword Serializable toegevoegd om dit op te lossen.
Bij de encryptie en het digitaal handtekenen wordt de data wel zelf geserialized en gedeserialized voor elk specifiek geval dat kan voorkomen. Dit gebeurd gebruikmakende van ByteArray-InputStreams of OutputStreams en Object- InputStreams of OutputStreams. Dit is gebruikt om Floats en List<Comment> om te zetten naar byte[] die nodig zijn voor encryptie. Ook gebeurt dit in de omgekeerde richting voor decryptie waarbij de byte [] in ofwel Float of in List<Comment> wordt omgezet.
Elke service van elk onderdeel is geïmplementeerd en er is gekozen om de aaneenschakeling van opdrachten te implementeren als verplichte uitbreiding. Wat nog niet werkt omdat RMI niet assynchroon is. Dit zou echter nog kunnen worden opgelost door de “start” methode een thread te laten starten die de berekening uitvoert zodat de CommandServer niet blockt op de uitvoering van de *START().*

Algemeen design



Het systeem bestaat uit drie onderdelen:

- **Client**: Dit zijn de computers die hun rekenkracht aanbieden om analyses uit te voeren. Deze ontvangen de te analyseren data van de Command Server en stuurt het resultaat van de analyser terug naar de Command Server.
- **Volunteer Registry Server**: Deze server staat in voor het registreren en de connecties open te houden tussen zichzelf en verschillende Clients, en ook tussen zichzelf en verschillende Command Severs. Ook houdt het bij welke Clients bezig zijn met opdrachten en dewelke niet.
- **Command Server**: Dit zijn de servers die de opdrachten verdelen over de beschikbare Clients en ook het volledige resultaat berekenen aan de hand van de resultaten verkregen van de Clients.

Volunteer Server

Deze server houdt de “boekhouding” bij. Het houdt bij welke Clients zijn geconnecteerd en dewelke idle zijn en dewelke niet. Ook zorgt het ervoor dat de Command Servers idle Clients kunnen verkrijgen om de analyses uit te voeren.

Command Server

De Command Server maakt gebruik van een greedy algoritme om aan Clients te geraken voor het uitvoeren van analyses. Dit betekend dat hij steeds nieuwe Clients gaat opvragen, en deze opdrachten gaat sturen zolang er Clients beschikbaar zijn. Ook houdt hij deze Clients zo lang mogelijk bezet zolang niet alle data is geanalyseerd, dit wordt besproken in het gedeelte Aaneenschakeling van opdrachten.

Dit is echter niet de beste manier om dit te doen. Dit zou in het geval wanneer er meerdere Command Servers aanwezig zijn een tekort van Clients kunnen veroorzaken. Het is bovendien ook geen optimale verdelingen van de rekenkracht.

Client

Dit zijn de elementen in het systeem die de eigenlijke analyse uitvoeren. Deze stellen een bepaalde hoeveelheid geheugen beschikbaar en laten ook toe dat er een beperkte hoeveelheid data wordt doorgestuurd per opdracht. De grootte kan worden opgevraagd met *SIZE()*. Er is geen abstractie voorzien dat het mogelijk maakt om verschillende Clients te instantiëren met verschillende geheugengroottes, opdracht groottes en gebruikte algoritmes. Waardoor dat elke Client in het systeem identiek zijn.

Encryptie

De Command Server initialiseert een beveiligde verbinding. Dit wordt gedaan door data te encrypteren door middel van RSA. Hierbij wordt de public key van de Client opgevraagd via *GET_KEY().* De Command Server gaat een symmetrische key genereren gebruikmakende van de methode *handshake(ClientProtocol client).* De key wordt gegenereerd voor AES encryptie.
Na het generen van de symmetrische key wordt deze door de Command Server opgeslagen in een dictionary waarbij de key de client is en de value de symmetrische key. Dit laat toe om van meerdere Clients hun bijbehorende keys bij te houden.

Digitaal handtekenen

Voor de digitale handtekening wordt er gebruik gemaakt van zowel de te doorsturen data als de symmetrische key. Hierbij wordt een hash gemaakt van de te verzenden data gebruikmakende van het HMAC-SHA256 algoritme met als key voor het hashing algoritme, de symmetrische key.
De data en de hash worden dan in een Pair<k,v> gestopt en vervolgens geëncrypteert voor het wordt doorgestuurd.
Bij ontvangst worden de gegevens gedecrypteerd door de ontvanger. De gegevens worden dan in twee opgesplitst in de data en de hash. Vervolgens gaat de client de data hashen en vergelijkt hij de berekende hash met de ontvangen hash. Wanneer deze met elkaar overeenkomen wordt de data opgeslagen, anders wordt deze genegeerd.

Aaneenschakeling van opdrachten

De Command Server houdt de Clients bij waarnaar hij al reeds een opdracht heeft gestuurd en nog niet idle zijn. Gebruik makende van *STORE_SIZE(ClientProtocol client)* en *TASK_SIZE(ClientProtocol client)* zoekt hij geschikte kandidaten om extra opdrachten naar op te sturen. Er wordt een maximum *TASK_SIZE* bepaald zodat clients met een te grote opdrachten queue niet worden gebruikt. *STORE_SIZE()* moet ook groter zijn dan nul. Als de client voldoet aan beide voorwaarden, dan krijgt deze extra opdrachten toegestuurd.

Dit is niet de meest efficiënte manier van handelen. Er was beter geopteerd om een best-fit algoritme te gebruiken, zodat het gebruik van de clients optimaal is.
Bovendien wordt de abstractie gebroken doordat de Command Server eigen instanties bijhoudt van welke clients niet idle zijn. Dit zou moeten gebeuren via de Volunteer Registery Server.

Result

De service RESULT() van de Command Server, gaat enkel een echt resultaat berekenen en terug geven wanneer de volledige dataset is geanalyseerd en elke Client ook zijn resultaat heeft doorgestuurd. Als dit niet het geval is, wordt er een bericht geprint en wordt het standaard resultaat van 0.0 teruggeven als resultaat.
Dit maakt het niet duidelijk voor andere mogelijke systemen of dat 0.0 het eindresultaat is of niet.