SDCC: Comunicazione Multicast in Go

1st Christian Santapaola
Facoltà di Ingegneria Informatica - laurea magistrale
Università di Roma Tor Vergata
Roma, Italia
9669chris@gmail.com

***Abstract*—Questa è la relazione per il progetto B1: Multicast applicativo in Golang per il corso di Sistemi distribuiti e cloud computing dell'università di Roma Tor Vergata.**
***Index Terms*—Multicast, Golang, SDCC**

I. Introduzione

Questa relazione volge a spiegare le scelte architetturali e implementative del progetto svolto, il progetto implementa:

- Multicast applicativo con Sequencer
- Multicast applicativo con Clock logico scalare.
- Multicast applicativo con Clock logico vettoriale.

II. Descrizione dell'architettura

Prima di discutere della struttura della rete parliamo dei componenti presenti nel sistema:

- un componente Name Service, il quale tiene traccia degli utenti, dei gruppi, degli utenti e della loro appartenenza al gruppo.
- Gli utenti effettivi, che creano, partecipano a gruppi e si inviano messaggi tra loro.

La comunicazione tra i vari utenti è di tipo P2P, ovvero gli utenti parlano direttamente tra loro e non tramite un intermediario e la semantica di comunicazione è at-most-once, ordinata tra due peer ed affidabile. La principale motivazione per le architetture scelte è stata la semplicità di sviluppo e debug; esistono architetture più efficienti ma più complicate sia da implementare che da debuggare. Per motivi simili la comunicazione tra nodi e con il name service avviene univocamente tramite grpc e protobuf, ciò perché posso non pensare ai dettagli della comunicazione di rete (serializzazione,deserializzazione,keep-alive...) e concentrarmi solo sull'implementazione degli algoritmi di multicast.

A. Architettura rete di overlay sequencer

La rete di overlay per la comunicazione multicast con sequencer è una rete strutturata di tipo centralizzato, la comunicazione avviene tramite il sequencer, il quale riceve i messaggi dagli altri peer e li trasmette al gruppo intero. I vantaggi di tale architettura sono:

1) La semplicità; infatti questa architettura non richiede algoritmi sofisticati, il che la rende facile da implementare e da debuggare in caso di problemi.
2) Il numero di messaggi è particolarmente efficiente; per ogni messaggio infatti, il sequencer dovrà mandare tanti messaggi quanti sono gli utenti del gruppo multicast, quindi il numero di messaggi da inviare scalano linearmente con la grandezza del gruppo ($O(n)$).
3) Ogni messaggio arriverà ad un peer in un massimo di due 'salti', quindi rendendo non necessario implementare meccanismi Time-To-Live.

I suoi svantaggi invece sono:

1) La centralizzazione che rende il sequencer il maggiore collo di bottiglia del sistema. Se il sequencer non riesce a processare tutti i messaggi in arrivo, il sistema va veloce al più quanto il sequencer.
2) La centralizzazione che rende sequencer il punto critico di fallimento del sistema. Se il sequencer subisce un crash la rete crasha con lui.
3) Il sequencer è un peer speciale, quindi in una rete P2P come abbiamo supposto, nonostante i nodi della rete siano omogenei tra loro, il carico di lavoro non lo è. Il sequencer compie la maggior parte del lavoro e gli altri nodi compiono un lavoro minimo a confronto, creando un carico di lavoro non uniforme nella rete, questo comporta che in un cluster di N nodi omogenei, molti nodi non fanno nulla mentre il sequencer lavora, creando uno spreco di risorse, che sono presenti ma non utilizzato.

B. Architettura di rete di overlay clock logico e clock vettoriale

La rete di overlay per la comunicazione multicast sia con clock logico che con clock vettoriale utilizzano la stessa architettura di rete applicativa. La rete è strutturata e P2P dove i nodi parlano direttamente tra loro, ogni gruppo forma la sua rete applicativa, la cui forma è quella di un grafo completo, ovvero ogni peer della rete mantiene una e attiva (nel grafo un arco) con ogni altro peer della rete. I vantaggi di tale architettura sono:

- Semplicità, sia di sviluppo che di debug, la rete essendo omogenea per ogni nodo, risulta più facile da visualizzare.
- Robustezza, ogni nodo sa tutto della rete, quindi se un nodo smette di funzionare, ogni nodo è capace di accorgersene da solo. In particolare questo tipo di rete è capace di riconoscere, quindi di gestire, il caso in cui è presente una partizione di rete molto semplicemente.
- Utilizzo della rete fisico sottostante, l'utilizzo di una connessione apposita permette di usufruire efficacemente della rete sottostante.

- Ogni Nodo è a distanza di un Hop da ogni altro nodo, ciò permette di poter evitare di implementare meccanismi di Time-To-Live.
- Omogeneità della rete, ogni nodo compie lo stesso carico di lavoro degli altri.

Gli svantaggi di tale architettura sono:

- Tempi di risposta: Il tempo per mandare un messaggio è lineare con il numero di utenti nel gruppo.
- Dimensione massima del gruppo: Utilizzando uno stack IP/TCP Il massimo di connessioni che un nodo può avere è limitato dal numero di porte disponibili, dando un numero massimo al numero di dispositivi in un gruppo.
- Problemi di consistenza: avendo dato come pro che questa architettura permette di riconoscere una partizione di rete e che la comunicazione è affidabile, per il Teorema CAP abbiamo che ogni nodo vedrà l'ordine di arrivo dei messaggi in maniera differente, essendo l'obbiettivo di questo progetto quello di ordinare i messaggi multicast, questo rende le cose più difficili.
- Numero di messaggi: il numero di messaggi richiesti per un singolo messaggio è dell'ordine del quadrato degli utenti del gruppo ($O(n^2)$), questo perché ogni messaggio richiede un messaggio di acknowledgement da ogni membro del gruppo.

III. Descrizione dell'implementazione

A. Name Service

Iniziamo dal name service, che nonostante sia un componente che sembri stare al bordo della comunicazione multicast, è il componente fondamentale per il suo funzionamento. Ogni peer si registra al naming server dandogli un indirizzo IP ed una porta in cui si mette in ascolto; il naming server fornisce inoltre un ID univoco al peer, costruito applicando la funzione hash sha256 sulla stringa 'ip:porta' fornita. Questo name server assume che la combinazione (ip, porta) identifichi univocamente un nodo. Le informazioni vengono salvate in database nosql basato su DHT. Il name service quindi è un client leggere pensato per essere facilmente riproducibile e scalabile sopra un DHT.

B. Comunicazione P2P

Ogni nodo è un peer all'interno del gruppo, ovvero si comporta sia da Server che da client, nell'implementazione il primo problema riscontrato è stato:

"Tra due peer chi inizia la connessione?"

La risposta a questa domanda è stata che per ogni peer la comunicazione è vista come una PIPE Unix, ovvero ogni peer ha un server attivo da cui riceve solamente messaggi, ma non manda niente, ed una seconda connessione di tipo client da cui manda pacchetti ma non ne riceve.

Quindi ogni peer, riceve messaggi sull'interfaccia server, quest'ultima sarà quella da comunicare al name server per permettere la comunicazione e manda messaggi su un'altra connessione di tipo client.

C. Ricezione dei messaggi

Una scelta implementativa che segue la' ultima è che il server espone all'indirizzo di ascolto un servizio di coda di messaggi, ovvero un messaggio in arrivo viene preso e inserito in una coda di messaggi di tipo FIFO, da cui verrà recuperato in seguito e processato. La coda di messaggi viene esposta tramite un servizio rpc, il client chiama un metodo remoto 'Enqueue()', mentre lato server la coda ha un metodo locale 'Pop()' che si occupa di rimuovere i messaggi dalla coda.

D. Sequencer

La scelta del sequencer è un problema critico nella formazione della rete di overlay, prima di tutto avviene prima che la rete sia pronta, secondo la scelta del sequencer deve essere univoca e condivisa da tutti i nodi. La soluzione scelta si basa sull'ipotesi che i membri di un gruppo rimangano stabili dopo l'inizio della comunicazione, quindi presi gli ID di tutti i nodi appartenenti al gruppo, vengono ordinati per ordine lessicografico, il più piccolo viene scelto come sequencer.

E. Struttura dei messaggi

L'implementazione dei messaggi è stata tale che ogni messaggio abbia un id univoco; questo ID è strutturato come 'sourceID:clock' ovvero l'ID univoco dato dal name server ed il clock (counter, scalare, vettoriale) per il messaggio, questo identifica univocamente ogni messaggio, la necessità di portarsi dietro il clock totale e non solo un 'semplice counter' è dovuto ad un problema sorto nella gestione degli ack.

F. servizio di Ack

L'implementazione del servizio di ack è stata più difficile di quanto mi sono aspettato. Ricordando quanto detto sopra un ack viene mandato solo quando un messaggio viene rimosso dalla coda FIFO del server per essere processato, tuttavia l'ack deve essere mandato in multicast tramite la comunicazione client. Uno dei problemi più difficili è stato tenere traccia degli ack arrivati per ogni messaggio, infatti dato che non c'è consistenza tra quello che vedono i nodi, può succedere che un nodo riceva l'ack di un messaggio prima di ricevere il messaggio stesso, quindi se quel messaggio ha clock inferiore a tutti i messaggi che stiamo processando, dobbiamo aspettare che arrivi prima di poter rilasciare altri messaggi. Per fare ciò dobbiamo sapere il clock del messaggio, ciò comporta la necessità che ogni ack porti con se il clock del messaggio a cui si riferisce.

G. Tenere traccia dei messaggi

Viene usato l'univocità degli ID dei messaggi per tenere delle strutture dati che tiene traccia della informazioni conosciute su un messaggio, per ogni messaggio ho necessità di sapere il numero di ack arrivati, se il messaggio è in arrivo, il clock dei messaggi arrivati ed in arrivo. Queste informazioni sono necessarie a sapere se un messaggio è 'pronto' ad essere rilasciato all'utente.

H. Comunicazione

Infine a livello utente, tutta l'astrazione viene nascosta da due chiamate che fanno Send() e Recv() dei messaggi.

La comunicazione è at-most-once e manda i messaggi in ordine a tutti gli altri membri, se un messaggio fallisce ad arrivare correttamente, il processo continua a riprovare finché non riesce, quindi in caso di partizione di rete se un gruppo di nodi diventassero irraggiungibili, gli altri membri dovranno aspettare che tornino raggiungibili per continuare la comunicazione.

IV. Descrizione delle eventuali limitazioni

Le limitazioni di questo progetto sono i seguenti:

1) La sicurezza della rete non è stata considerata, la rete non ha alcun tipo di sicurezza e si aspetta che ogni nodo si comporti bene. Essendo questo progetto più centrato sull'implementazione della comunicazione multicast, lì è dove è andata la maggior parte del lavoro.
2) Un nodo necessita di registrarsi alla rete prima di iniziare la comunicazione perché una volta iniziata non avrà più modo di unirsi.
3) Ogni nodo deve sapere a quale indirizzo IP e porta mettersi in ascolto e deve garantire che sia raggiungibile da ogni membro del gruppo.
4) Ogni nodo deve sapere a priori l'indirizzo dove raggiungere il name service.

V. Descrizione delle tecnologie utilizzate.

1) La piattaforma di sviluppo è linux
2) Il linguaggio di programmazione GO per la scrittura del progetto.
3) Il protocollo grpc e protobuf per la comunicazione tra peer, i nodi comunicano tra loro utilizzando servizi grpc.
4) Il DHT utilizzato è ETCD
5) Il name service è basato anche lui su grpc e protobuf.
6) Uso una libreria grpc per comunicare con il cluster etcd.
7) una libreria per interagire con il formato yaml

References