Project B4: Gossip-based Distance Estimation and Failure Detection

Luigi Ciuffreda
Mat. 0351898
Sistemi Distribuiti e Cloud Computing
Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Roma Tor Vergata
Luigi.ciuffreda@students.uniroma2.eu

***Abstract*—Questo documento descrive le scelte progettuali e strutturali adottate nello sviluppo di un sistema basato su algoritmi di gossip per stimare le distanze tra nodi e rilevare guasti. Il sistema utilizza coordinate di rete in uno spazio euclideo per prevedere la latenza della rete tra nodi senza misurazioni dirette. La soluzione è stata testata utilizzando Docker Compose e distribuita su istanze EC2 di Amazon.**

I. INTRODUZIONE

L'obiettivo del progetto è implementare algoritmi di gossip per stimare la distanza tra nodi (ad esempio, l'algoritmo Vivaldi) e rilevare guasti. Il sistema assegna coordinate ai nodi in uno spazio euclideo in modo che la distanza tra due nodi predica accuratamente la latenza della rete tra di essi. Questo approccio elimina la necessità di misurazioni di latenza diretta tra ogni coppia di nodi.

II. TECNOLOGIE UTILIZZATE

Il sistema distribuito realizzato per il progetto è stato implementato utilizzando il linguaggio di programmazione Go, scelto per la sua efficienza nella gestione delle operazioni concorrenti, e Docker per la virtualizzazione degli ambienti di esecuzione dei nodi. Il deployment è stato gestito tramite Docker Compose e distribuito su Amazon EC2 per garantire scalabilità e flessibilità in un ambiente cloud. Inoltre, è stata integrata una configurazione dinamica attraverso un file .json, che permette di definire in maniera flessibile il numero di nodi da inizializzare nel sistema. Questo approccio consente una facile scalabilità e adattamento alle diverse esigenze di test e di deployment in ambienti variabili. Il file **.json** viene letto all'avvio del sistema, e il numero di nodi specificato viene utilizzato per generare automaticamente il file **docker-compose.yml,** facilitando l'orchestrazione dei container Docker.

III. ARCHITETTURA

Il Sistema distribuito implementato nel Progetto utilizza una combinazione di tecnlogie modern per garantire efficienza, scalabilità, affidablità. L'architettura si basa su tre componenenti principali: i nodi, il registro e il Sistema di comunicazione.

A. Nodi

I nodi rappresenetano l'unità fondamentale del sistema. ogni nodo è un container Docker, che esegue un'istanza dell'applicazione scritta in Go. Questa scelta tecnologica permette di isolare le applicazioni, garantendo che ogni nodo possa operare indipendentemente dagli altri, pur mantenendo la capacità ci comunicare efficacemente. Ogni nodo si occupa della:

- **Gestione delle coordinate**: ogni nodo mantiene un seti coordinate che rappresentano la sua posizione in uno spazio euclideo. Queste coordinate sono utilizzate per stimare la latenza di rete tra i nodi. Le coordinate vengono continuamente aggiornate utilizzando l'algoritmo Vivaldi, basato sugli scambi di informazioni tra i nodi.
- **Comunicazione Peer-to-Peer**: i nodi comunicano tra loro attraverso protocolli http, scambiandosi periodicamente informazioni sulle proprie coordinate e sulla latenza simata. Questo meccanismo di gossiping è fondamentale per mantenere aggiornate le coordinate di ciascun nodo per rilevare eventuali guasti all'interno della rete.

B. Registry

Il registro agisce come punto di riferimento centrale per tutti I nodi del Sistema. Quando un nodo sia vvia, si registra presso al registry inviando il proprio ID, indirizzo IP e le coordinate iniziali. Il registro mantiene una lista di nodi attivi con le loro coordinate facilitando la comunicazione e la sincronizzazione all'intenro della rete.

- **Funzionalità di monitoraggio**: il registro monitora conitnuamente l'attività dei nodi attravero ping periodici. Questo permette di rilevare rapidamente nodi non più attivi e di aggiornare la rete di conseguenza.

C. Sistema di comunicazione

La comunicazione all'interno del sistema avviene principalmente attraverso HTTP, facilitando l'implementazione di un sistema di gossiping tra i nodi. I nodi inviano richieste HTTP per scambiare le proprie coordinate con altri nodi e per aggiornare il registro centrale. Questo modello di comunicazione si adatta bene all'ambiente distribuito e garantisce che i nodi possano operare indipendentemente l'uno dall'altro.

IV. OPERAZIONI FORNITE

Il sistma distribuito realizzato offer una serie di operazioni essenziali per garantire il corretto funzionamento della rete, la gestione dei nodi e la rilevazione dei Guasti. Queste operazioni sono state progettate per essere semplice ma efficaci, garantendo l'affidabilitò e la scalabilità del Sistema

A. Registrazione del Nodo

Ogni nodo, al momento dell'avvio, deve registrarsi presso il registry. Questa operazione è fondamentale per inserire il nodo all'interno della rete e per consentire al registro di tenere traccia di tutti I nodi attivi.

- Il nodo invia una richiesta HTTP al registro con I propri dettagli, inclusi l'ID, l'indirizzo IP e le coordinate iniziali.
- Il registro risponde confermando l'avvenuta registraione e fornendo informazioni utili per la successiva comunicazione con altri nodi.

B. *Ping Periodico*

Una volta registrato, il nodo invia periodicamente un ping al registro per segnalare la propria presenza.questo meccaniscmo permette al registro di monitorare l'attivita della rete e di rilevare nodi non più attivi, ad esempio a causa di guasti.

- Il ping contiene l'ID del nodo e le coordinate aggiornate.
- Se un nodo non invia ping per un determinato intervallo di tempo il registro lo considera inattivo e lo rimuove dalla lista dei nodi attivi.

C. *Aggiornamento delle coordinate*

Ogni nodo deve mantenere le proprie coordinate aggiornate per fidlriflettere correttamente la altenza della rete rispetto agli altri nodi. Questo avviene tramite l'algoritmo di Vivaldi, che sfrutta le informaioni raccolte durante le operazioni di gossiping tra i nodi.

- Il nodo seleziona casualmente un altro nodo con cui comunicare e scambia informazizoni sulle proprie coordinate e il tempo di risposta (RTT).
- Sulla base di queste informazioni, il nodo aggiorna le proprie cooridnte, cercando di minimizzare l'errore tra la distanza stimata e quella effettiamente Misurata.

D. *Rilevamento dei guasti*

Il Sistema deve essere in grado di rilevare nodi Guasti o non più attivi, in modo da mantenere l'integrità della rete e garantire la continuità del servizio.

- Il registro tiene traccia dei ping ricevut dai nodi. Se un nodo non invia ping entro un intervallo di tempo specific, viene considerate inattivo.
- Gli altri nodi nella rete sono informati del guasto, permettendo loro di ricalcolare le proprie coordinate in base ai nodi rimantenti.

E. *Scambio di informazioni (Gossiping)*

La comunicazione tra I nodi avviene tramite gossiping. Tuttavia , a differenza di una approccio distribuito puro, in cui ogninodo comunica liberamente con altri nodi senza un punto entrale di controllo, il Sistema adottato in questo Progetto utilizza un registro centrale per coordinare e facilitare queste operazioni.

- I nodi scelgono casualmente altri nodi a cui inviare le proprie informazioni e ricevere aggiornamenti sulle coordinate.
- Questo processo continua indefinitivamente, assicurando che tutti I nodi mantengano una visione aggiornata della rete.

V. LOGICA DEL SISTEMA

La logica del Sistema sviluppato si basa un'architettura centralizzata in cui un registtro centrale coordina la comunicazione e il funzionamento dei nodi che partecipano alla rete. Ecco I passaggi per garantire il corretto funzionamento della rete, la gestione dei nodi e la rilevazione dei Guasti:

1. **Avvio e registrazione del Nodo**
   - *Avvio del Nodo*: ogni nodo, al momento dellavvio, inizializza le sue coordinate e recupera le informazioni di configurazione del file **".json"**.
   - *Registrazione presso il registry*: il nodo invia una richiesta HTTP al registro con le proprie informazioni e il registro conferma l'avvenuta registrazione e aggiorna la lista dei nodi attivi nella rete.
2. **Ping Periodico e Monitoraggio**
   - *Invio periodico di Ping*: dopo la registrazione, il nodo invia periodicamente un ping al registro pr segnalare la propria presenza il ping contiene l'Id del nodo e le sue cooridnate aggiornate
   - *Monitoraggio da parte del registro*: il registro monitora l'attività ei nodi attraverso i ping ricevuti e se un nodo non invia ping entro un intervallo di tempo prestabilito viene considerato inattivo o guasto.
3. **Comunicazione tra Nodi**
   - *Scelte del Nodo per il Gossiping*: ogni nodo seleziona casualmente un altro nodo con cui comunicare e scambiare informazioni sulle coordinate ed il tempo di risposta.
   - *Scambio di informazioni*: i nodi scambiano le loro coordinae e misurao il tempo di risposta reciproco ed inoltre le informazioni scambiate veogno utilzzate per aggiornare le coordinate dei nodi secondo l'algortimo di Vivaldi
4. **Aggiornamento delle coordinate**
   - *Calcolo della latenza stimata*: il nodo utilizza l'algoritmo di Vivialdi per calcolare la latenza stimata rispetto al nodo con cui ha effettuato il gossiping,
   - *Aggiornamento delle coordinate*: le coordinate vengono aggiornate per minimizzare l'errore tra la latenza stimata e quella effettivamente misurata, migliorando la precizione della posizione del nodo nella rete.
5. **Rilevamento e gestione dei guasti**
   - *Rilevamento dei nodi inattivi:* il registro centrale rileva i nodi che non inviano ping entro il tempo prestabilito e li considera inattivi o guasti.

- *Aggiornamento della topologia di rete:* quando un nodo viene segnalato come guasto, il registro aggiorna la lista ei nodi attivi, gli altri nodi rimanenti ricalcolano le loro cooridnate per adattarsi alla nuova topologia di rete.

6. **Configurazione dinamica dei nodi**

- *Lettura del file di configurazione (.json):* il sitema legge un file ".json" che specifica il numero di nodi da inizializzare.
- *Generazione automatica di container:* in base al numero di nodi specificati nel file il sistema genera automaticamente il "docker-compose,yml" per orchestrare i container Docker.

7. Emulazione ritardi di rete

Nel sistema implementato, l'emulazione dei ritardi di rete viene simulata utilizzando una funzione chiamata `simulateNetworkDelay`. Questa introduce una latenza artificiale attraverso l'uso della funzione `time.Sleep` di Go, che mette in pausa l'esecuzione del programma per un determinato intervallo di tempo. La durata del ritardo è calcolata utilizzando una distribuzione normale con una media e una deviazione standard in modo tale da avere una variabilità naturale dei ritardi di rete.

VI. Tolleranza ai guasti

Il Sistema è progettato per garantire continuità operative anche in caso di Guasti. Ogni nodo invia periodicamente un ping al registro centrale per confermare la propria operatività. Se un nodo non invia ping entro un tempo stabilito, il registro lo considera inattivo e lo rimuove dalla lista dei nodi attivi. In caso di guasto, gli altri nodi aggiornananto automaticamente le loro coordinate utilizzando l'algortimo di Vivaldi adattandosi alla nuova configurazione della rete. Questo meccanismo assicura che la rete rimanga stabile e che le comunicazioni tra i nodi continuino senza interruzioi significative. Il Sistema è quindi robusto e in grado di mantenere la propria funzionalità anche in presenza di malfunzionamenti o perdita di nodi

VII. Limiti del Sistema

Uno dei principal limiti del Sistema è la dipendenza dal registro centrale che funge da punto di controllo per la gestione dei noid. Questa centralizzazione introduce un potenziale **single point of failure** : se il registro dovesse fallire , l'intero Sistema potrebbe interrompere la sua operatività, poichè I nodi non sarebbero più in grado di registrarsi o aggiornare le loro infomazioni. Inoltre può diventare un **collo di bottiglia** per le comunicazioni, specialmente in reti con un numero elevato di nodi, andando a rallentare il Sistema e limitandone la scalabilità