Progetto B1: Chord System

*Note: Sub-titles are not captured in Xplore and should not be used

Matteo Federico
matricola 0321569
Corso di Sistemi distribuiti e cloud computing
Laurea magistrale in Ingegneria Informatica, Università di Roma - Tor Vergata
matteo.federico.98@students.uniroma2.eu

***Abstract*—Questo documento ha l'obiettivo di descrivere il funzionamento e le scelte progettuali effettuate durante la realizzazione del progetto assegnato per il corso di sistemi distribuiti e cloud computing. Verrà presentato l'architettura del sistema e gli algoritmi usati nell'implementazione. evidenziando anche i possibili punti critici del sistema.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduzione

Il progetto scelto e realizzato per il corso di Sistemi Distribuiti e Cloud Computing è il B1. La richiesta era quella di implementare nel linguaggio GoLang un sistema basato sul protocollo "Chord" che potesse memorizzare in maniera distribuita stringhe utilizzando più server e di poter interagire con il sistema tramite un client. Inoltre, l'intero sistema doveva essere simulato utilizzando la virtualizzazione a livello sistema operativo tramite Docker e caricare il sistema in cloud tramite il servizio EC2 di Amazon.

II. Tecnologie adottate

In questa sezione si andranno ad elencare i vari linguaggi e framework utilizzati per implementare il sistema:

- **Go**: Il linguaggio principale con cui è stato sviluppato il sistema. La scelta è stata dovuta dalle specifiche di progetto.
- **Bash**: utilizzato per creare il file di avvio, permette la configurazione e l'avvio in maniera facile e configurabile.
- **Python**: Utilizzato per creare il file di docker compose all'avvio secondo le specifiche date dall'utente
- **Docker**: Utilizzato come supporto per la virtualizzazione a livello sistema operativo permette di nascondere a livello sottostante il reale sistema utilizzato.
- **Docker Compose**: Utilizzato per poter istanziare e gestire tutti conteiner all'interno della stessa macchina. Ha permesso anche di creare la rete per far comunicare i vari nodi tra loro
- **GoRPC**: per la comunicazione tra i vari nodi è stato utilizzato come meccanismo di comunicazione le RPC (remote procedure call) che permettono una comunicazione transiente, bloccante e sincrona tra i vari nodi. La libreria utilizzata è quella nativa di go del package "net/rpc"

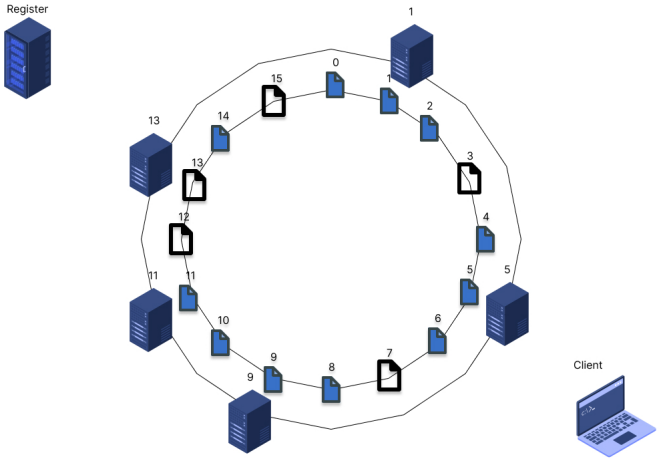
Identify applicable funding agency here. If none, delete this.

- **MD5**: Si è utilizzata come funzione di Hash Crittografica per poter ottenere per ogni oggetto un hash della stessa dimensione e poter quindi identificare per poter associare ad ogni elemento del sistema un ID univoco. La funzione utilizzata è quella implementata nativamente in Go
- **EC2**: il service cloud di amazon è stato utilizzato per effettuare il deploy dell'applicazione su una macchina virtuale nel cloud. La macchina virtuale ha un istanza t2.micro con 1 vCPU con 2.5 GHZ e 1GB di ram, e 8 GB dedicati allo storage su SSD.

III. L'Architettura

A. Il Sistema

Il sistema è composto da un **overlay network strutturata** formata da un insieme di **nodi** la cui dimensione può variare dinamicamente nel tempo, un unico **server** register e un applicazione client che permette di comunicare con il sistema. La struttura che si cerca di creare è quella di un anello.



I nodi sono gli effettivi server che memorizzano e gestiscono le risorse e che tramite il protocollo RPC permettono di esporre le operazioni di Put, Get e Remote verso l'esterno del sistema.

Ogni nodo tiene in memoria una conoscenza parziale della rete. Infatti, mantiene informazioni del suo nodo precedente e del suo nodo successore, inoltre tiene traccia di una tabella chiamata Finger Table composta da **NBit** righe dove ogni riga mantiene traccia di un nodo, e

le righe sono ordinati in base alla distanza dal nodo.

FingerTable	
0	succ(ID)
1	succ(ID+2)
2	succ(ID+4)
3	succ(ID+8)

Ogni nodo calcola la sua posizione nell'anello tramite la stessa funzione per calcolare la chiave delle risorse, utilizza una variabile di ambiente chiamata **CODICE-HASH** per ottenere la sua chiave. Infine, ogni nodo conosce per il suo insieme di risorse quali altri nodi fungono da repliche.

Il register è un server singolo, con IP fisso e conosciuto da tutti i nodi, che svolge due ruoli principali: il primo quello di funzionare da punto di inizializzazione per i nodi, infatti, ogni nodo che vuole entrare nell'anello deve prima comunicare al register tramite la chiamata RPC "Register" la volontà di entrare a far parte del sistema e lui gli comunicherà i nodi che già fanno parte dell'anello.

Il secondo compito è quello di fornire al client un entry-point per comunicare con il sistema ovvero quello di comunicare ai client un nodo che fa parte della rete. Il nodo viene scelto dal register con una politica Round-Robin.

Il client implementato con un'interfaccia CLI permette di comunicare con il sistema esponendo le operazioni fondamentali, Put, Get, Remove.

B. Le Operazioni

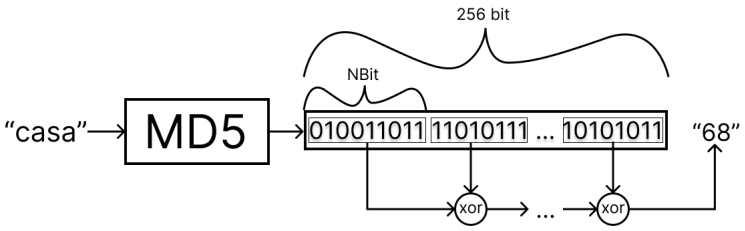
Il sistema permette di memorizzare risorse, che vengono effettivamente rappresentate da stringhe. Ad ogni risorsa memorizzata viene assegnata una chiave che rappresenta la locazione di quella risorsa all'interno del sistema. La chiave di ogni risorsa viene calcolata tramite una combinazione della funzione di hash **MD5** e l'operazione logica di **XOR**, precisamente la funzione permette di prendere una stringa, la converte in un hash composto da 256 bit, divide l'hash in parti uguali della stessa dimensione di **NBit** (è un parametro configurabile all'avvio del sistema che può assumere solo i valori 8/16/ 32 e permette di identificare la dimensione massima dell'anello)

- **key Put (Resource)**: data una risorsa permette di memorizzarla nel sistema e restituisce la chiave che la identifica. Se esiste già una risorsa memorizzata con la stessa chiave, la risorsa viene sostituita.
- **Resource Get (Key)**: permette di ottenere la risorsa collegata alla chiave che si è passata come parametro. Se la chiave rappresenta una risorsa vuota viene ritornata la stringa vuota come valore di ritorno.
- **Remove (Key)**: permette di rimuovere una risorsa all'interno del sistema

C. Funzione di Hash

Tutto il protocollo di Chord si basa sul creare, tramite hash distribuito, un anello dove ogni nodo possa essere identificato su una specifica posizione e che due nodi diversi difficilmente possano mapparsi sullo stesso identificativo, per questo la funzione di hash è fondamentale per ridurre le collisioni per questo motivo ho scelto di utilizzare un algoritmo di hash crittografico per poter ridurre al minimo la possibilità di collisione. La chiave di ogni risorsa e viene calcolata tramite una combinazione della funzione di hash crittografica MD5 e l'operazione logica di XOR, precisamente la funzione permette di prendere una stringa, la converte in un hash composto da 256 bit (16 bytes), scompone l'hash in parti uguali della stessa dimensione di NBit (dove è un parametro configurabile all'avvio del sistema che può assumere solo i valori 8/16/32 e permette di identificare la dimensione massima dell'anello) e infine effettuata l'operatore XOR a tutti i frammenti uscenti. Ogni nodo per identificare se stesso nell'anello utilizza, un proprio parametro, chiamato CODICE-HASH che viene sottoposto alla stessa procedura della stringa per poter calcolare la posizione nell'anello.

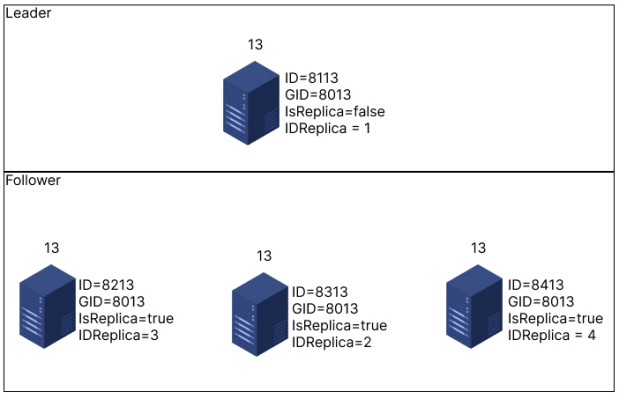
Per affrontare la possibilità se pur bassa di avere delle collisioni, ovvero, più nodi abbiano lo stesso identificativo, si é deciso di imporre che i nodi con lo stesso identificativo formino un cluster che gestiscano le medesime risorse.



D. Tolleranza ai guasti

Per aumentare la tolleranza ai guasti e aumentare la consistenza in caso di errori nei vari nodi, si è adottata la tecnica del **Leader-Follower**.Consiste nel avere un cluster di nodi che hanno medesima chiave e quindi dovranno gestire il medesimo insieme di risorse. Questo insieme di nodi formerà un cluster per quell'insieme di risorse. Il cluster avrà un solo Leader che fungerà da primario e il resto dei nodi come Follower (o repliche) che fungeranno da duplicati in caso il Leader dovesse avere un guasto.

E' importante notare come questa tecnica di duplicazione non serve ad aumentare la disponibilità del sistema poiché i nodi replica del cluster non sono esposti e non vengono utilizzati dagli altri nodi dell'anello.



Le operazioni di lettura e scrittura da parte del client possono avvenire solo sul Leader. Ogni volta che viene scritto una risorsa il leader invia la scrittura sui nodi Replica in modo tale da tenerli aggiornati.

I nodi Replica oltre ad aggiornare le proprie risorse quando vengono contattati dal Leader, svolgono il compito di monitoraggio di quest'ultimo e in caso di guasto o di non risposta del Leader, avviano un algoritmo di elezione per eleggere il nuovo Leader. L'algoritmo di elezione scelto è **l'algoritmo Bully**

Ogni volta che un nodo Replica si aggiunge al cluster contatta il Leader per ottenere tutte le risorse che gestisce.

1) Operazione di Put: Quando un nodo viene contattato per l'inserimento di una nuova entry, calcolerà l'ID della risorsa e controllerà se quella chiave apparterrà alle risorse a lui assegnate, se non appartiene alla sua lista di risorse assegnate dovrà cercare nella sua FingerTable

2) Algoritmo bully: quando il nodo Leader non risponde i nodi Replica avviano un algoritmo di elezione, nel particolare, una versione inversa dell'algoritmo bully dove la modifica apportata é che il nodo che viene eletto é il nodo replica con valore di identificazione minore.
Per tanto quando un nodo replica avvia l'algoritmo contatta tutti i nodi replica con id minore del suo e se ve ne é almeno uno che risponde annulla l'algoritmo è si rimette in attesa.
Fino a quando la replica con ID minore viva manderà l'aggiornamento al server register e segnalerà che lui é il nuovo Leader.Infine, invierà l'aggiornamento a tutti i nodi replica.

3) Tolleranza agli errori: Un nodo che subisca un fallimento nel contattare un altro nodo, avvierà una routine di gestione degli errori, dove per prima cosa andrà a contattare il server register per ri-ottenere i suo predecessore e il suo successore, inseguito eventuali cambiamenti dei nodi nella sua FingerTable e annullerà l'operazione che ha avviato la comunicazione.

E. Problemi del sistema

1) server register: Il problema principale dell'architettura, sicuramente é il server register centralizzato, che si deve occupare di fornire l'entrypoint a eventuali client che vogliono comunicare con il sistema, il che significa che se il server register subisce un guasto la rete non sarebbe più raggiungibile.
Per allegierire il più possibile, il suo compito e distribuire nel modo più uniforme possibile le richieste ai vari nodi, il server register utilizza una politica round-robin per scegliere il nodo da restituire al client in caso voglia comunicare con l'anello.
Deve monitorare tramite heartbit lo stato di vita dei possibili server che partecipano alla rete inoltre deve mantenere aggiornata la lista dei possibili server che partecipano al sistema.

2) Partizione di rete: in caso si verificasse una partizione di rete, verrebbero create due partizioni dove solo in una delle due sarà presente il server register. Per tanto solo i nodi nella partizione del server register saranno in grado di ricreare l'anello e potranno riportare solo le loro informazioni.
Quindi in caso di possibile partizioni di rete, impedisce ad una delle due partizioni di poter formare un anello e quindi impedisce che i nodi formino due sistemi separati, in questo caso però il sistema non riesce a garantire la disponibilità di tutti i dati dell'anello che sono precedentemente salvati.