Progetto B1: Sistema Chord

Michele Tosi  
mat. 0327862  
Sistemi distribuiti e cloud computingLaurea magistrale in Ingegneria Informatica Università degli Studi di Roma Tor Vergata  
michele.tosi@students.uniroma2.eu

*Abstract*—Questo documento fornisce una descrizione delle scelte progettuali e strutturali adottate nello sviluppo del sistema Chord, un sistema costituito da nodi collegati ad anello che gestiscono delle risorse. La sua architettura basata su distributed hash table consente la ricerca rapida e scalabile delle informazioni.

# Introduzione

Il Progetto ha come scopo la realizzazione di un sistema Chord che permetta l’inserimento e la rimozione di nodi nella rete configurata a forma di anello. I nodi devono essere in grado di comunicare tra loro per poter eseguire le operazioni di ricerca, inserimento e rimozione di una risorsa. La gestione delle risorse avviene tramite finger table in modo da rendere gli accessi alle risorse più veloci e, di conseguenza, il sistema più performante. Per quanto riguarda la simulazione del modello si è utilizzata la virtualizzazione a livello di sistema operativo tramite Docker e si è caricato il sistema in cloud tramite il servizio EC2 offerto da Amazon.

# Tecnologie utilizzate

Per la realizzazione del progetto è stato utilizzato il linguaggio di programmazione **Go** per lo sviluppo della logica del sistema chord e per lo sviluppo automatizzato del file docker-compose. È stato inoltre utilizzato il linguaggio di scripting **Bash** per l’aggiunta di un nodo isolato e per la configurazione automatizzata della macchina virtuale avviata tramite il servizio **EC2** utilizzato per effettuare il deploy dell’applicazione su una macchina virtuale nel cloud.

Altre tecnologie utilizzate sono **Docker** per realizzare la virtualizzazione a livello di sistema operativo permettendo l’isolamento tra i vari nodi, **Docker Compose** per istanziare e gestire i container all’interno della stessa macchina oltre che per creare la rete attraverso la quale i nodi sui diversi container riescono a comunicare.

Per la comunicazione tra i vari nodi è stata utilizzata la libreria **goRPC** per la gestione di chiamate remote.

Infine, per quanto riguarda la configurazione dei parametri che caratterizzano il sistema (numero di bit per gli identificatori e numero di nodi da creare al momento dell’inizializzazione) è stato utilizzato un file **.json**, mentre per la gestione degli id dei nodi e delle risorse è stato utilizzato il meccanismo di hash **SHA-1** modificato opportunamente in modo da ottenere uno spazio continuo in cui mappare sia nodi che risorse (consistent hashing).

# Architettura

## Struttura

Il sistema è composto da una overlay network strutturata la cui dimensione può variare in modo dinamico nel tempo formata da un insieme di nodi che si occupano della gestione delle risorse, da un client che comunica con il sistema ed un server registry.

I **nodi** memorizzano e gestiscono le risorse esponendo tramite il protocollo RPC le funzioni per aggiungere, rimuovere o cercare le risorse. Ad ogni nodo viene assegnato un ID che lo identifica all’interno della rete, il nodo si occupa di gestire tutte le risorse che si trovano tra lui (incluso) e il suo predecessore (escluso).

Il **server registry** è un server il cui IP è fisso e conosciuto sia dai nodi che dal client.

Si occupa di gestire l’ingresso dei nuovi nodi nella rete, infatti, non appena un nuovo nodo fa richiesta per entrare questo gli risponde con l’IP del suo successore.

Si occupa inoltre di comunicare al client un nodo della rete da contattare per interfacciarsi al sistema secondo una strategia Round-Robin in modo da mantenere un carico bilanciato tra i vari nodi.

Il **client** una volta ricevuto l’IP di un nodo appartenente alla rete permette di eseguire le operazioni di inserimento ricerca e rimozione delle risorse, in più permette di fare uscire nodi dalla rete.

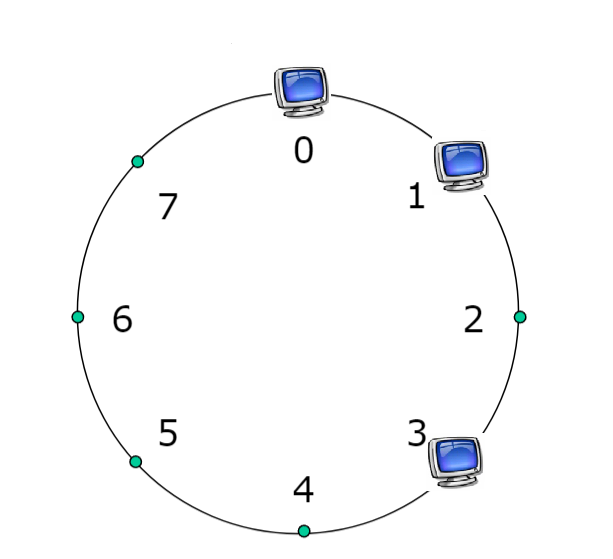


Fig 1 Rappresentazione anello sistema Chord, ogni nodo gestisce tutte le risorse tra sé (incluso) e predecessore (escluso).

## Funzione di Hash

Il sistema Chord sfrutta la tecnica del **consistent hashing** per gestire la distribuzione delle chiavi e il bilanciamento del carico tra i nodi della rete. Questa prevede che sia le risorse che i nodi vengano mappati uniformemente sullo stesso spazio di indirizzamento mediante una funzione di hash e ciascun nodo gestisca un intervallo di chiavi hash consecutive.

Nel contesto del progetto, è stata apportata una modifica alla funzione di hash basata su SHA-1 in modo che l’output generato costituisse un ID appartenente all’anello. Questo è stato ottenuto suddividendo l’output di 160 bit in parti di lunghezza pari al numero di bit utilizzati per la creazione dell’anello e successivamente eseguendo l’operazione di XOR tra esse.

## Finger Table

Per ottimizzare le operazioni viene utilizzata una Finger Table, ossia una tabella di routing in cui il nodo mantiene informazioni sulle posizioni degli altri nodi nell’anello logico distribuito. In questo modo si evita che ogni nodo debba mantenere informazioni su tutti gli altri, con quest’ultima soluzione che comporterebbe un costo di ricerca ottimale O(1), ma una quantità elevata di informazioni da mantenere in memoria oltre al fatto che in caso di join o leave ogni nodo dovrebbe contattare tutti gli altri. L’opzione opposta, ossia mantenere solo l’informazione sul successore comporterebbe contattare tutti i nodi dell’anello nel caso peggiore per eseguire la ricerca con costo quindi O(N).

La Finger Table è costituita da una lista di nodi formata da m righe, dove m è il numero di bit che costituiscono gli ID.

Se un nodo p vuole costruire la sua Finger Table per ogni riga i esegue ricerca il successore della risorsa:

In questo modo ogni nodo conosce bene i nodi vicini e la conoscenza decresce con l’aumentare della distanza. Sfruttando la Finger Table il costo dell’operazione di ricerca risulta essere O(log N).

L’aggiornamento della Finger Table non viene eseguito ogni volta che c’è un nuovo ingresso nella rete perché è un’operazione onerosa, ma viene eseguita periodicamente. In questo modo si ottiene un trade-off tra prestazioni e consistenza.

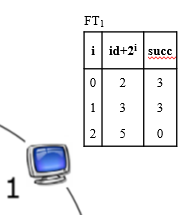


Fig 2 Finger Table relativa al nodo 1.

# Operazioni fornite

Il sistema permette la memorizzazione di risorse (stringhe) a ognuna delle quali viene assegnato un ID che rappresenta la loro posizione all’interno dell’anello e viene utilizzata per la gestione da parte dei nodi.

Il client per interagire con il sistema avrà a disposizione le seguenti operazioni:

* **Inserimento di una risorsa:** memorizza una risorsa all’interno del sistema in modo che possa essere possibile recuperarla in un secondo momento. Questa operazione restituisce l’identificativo della risorsa utile per la sua gestione.
* **Ricerca di una risorsa:** restituisce la risorsa con ID pari a quello inserito dall’utente se questa è presente all’interno del sistema.
* **Rimozione di una risorsa:** rimuove la risorsa con ID pari a quello inserito dall’utente se questa è presente nel sistema.
* **Rimozione di un nodo:** il nodo con ID inserito se presente nel sistema effettua la leave, tutte le risorse che gestisce passano al suo successore.

È possibile infine aggiungere un nuovo nodo. Una volta creato tramite script bash, questo contatta il server registry per ottenere gli IP dei suoi vicini in modo da poter entrare nella rete e iniziare a gestire le risorse acquisite dal successore.

# Logica del sistema

## Join di un nodo

Il sistema gestisce l’aggiunta di nuovi nodi all’interno di esso. L’ingresso di un nuovo nodo nel sistema segue quattro fasi:

### **Contatto del registry**:il nodo per entrare nella rete contatta il registry per avere informazione riguardo ai suoi vicini in modo da collocarsi correttamente all’interno dell’anello.

### **Collocazione nel sistema:** il nodo comunica ai vicini ricevuti nel passaggio precedente la sua intenzione di entrare nella rete. In risposta, il successore aggiorna il proprio predecessore. Il predecessore a sua volta aggiorna il proprio successore e la propria Finger Table.

### **Trasferimento delle risorse:** il nodo appena entrato contatta il successore per ottenere le risorse che dovrà gestire.

### **Creazione della Finger Table:** il nodo crea la propria Finger Table secondo la logica specificata precedentemente.

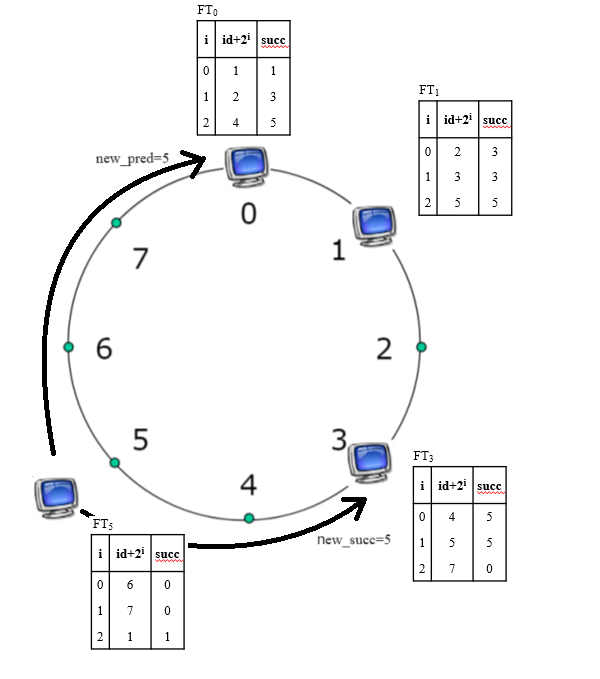


Fig 3 join di un nodo, il nodo 5 entra, lo comunica ai suoi vicini, le Finger Table vengono aggiornate.

## Leave di un nodo

Il sistema permette la leave dei nodi. Una volta che l’utente specifica l’ID del nodo da rimuovere avvengono quattro passaggi:

### **Contatto del registry**:il client contatta il registry fornendo l’ID del nodo da rimuovere, se questo è presente nell’anello viene contattato.

### **Trasferimento delle chiavi al successore:** il nodo prima di uscire trasferisce le proprie risorse al successore che si occuperà di gestirle.

### **Contatto del successore:** il nodo comunica al successore il suo nuovo predecessore.

### **Contatto del predecessore:** il nodo comunica al predecessore il suo nuovo successore. Il predecessore a questo punto aggiorna anche la sua Finger Table.

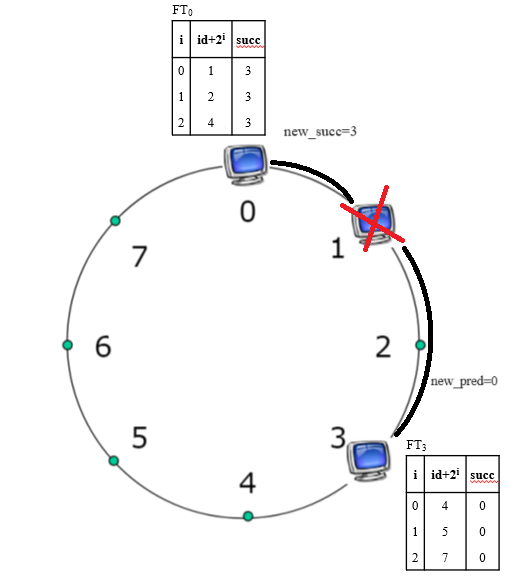


Fig 4 il nodo 1 esce, comunica al predecessore il nuovo successore e al successore il nuovo predecessore, in seguito le Finger Table vengono aggiornate.

# Tolleranza ai guasti

Il sistema non presenta meccanismi di replicazione delle risorse, se un nodo fallisce tutte le risorse che gestisce vengono perse.

Il sistema è comunque in grado di continuare a funzionare: in caso di fault di un nodo, il componente che si accorge del malfunzionamento lo comunica al registry che, dopo aver controllato che il nodo sia effettivamente non funzionante, restituisce a chi l’ha contattato l’IP del suo nuovo successore.

# Limiti del sistema

I principali limiti osservati nel sistema sono i seguenti:

* **Server registry:** il server registry è un componente centralizzato quindi funge da single point of failure, infatti, se c’è un fault nel registry il client non può più interfacciarsi con i nodi del sistema, e collo di bottiglia, poiché durante l’esecuzione viene contattato dai client per interfacciarsi con il sistema e dai nodi che devono entrare nel sistema a cui serve conoscere i loro vicini o che devono verificare se un altro nodo è fallito.
* **Partizionamento della rete:** il sistema non è tollerante rispetto al partizionamento della rete, infatti, questa porterebbe il registry ad essere presente solo in una delle due partizioni portando a perdere le risorse memorizzate nell’altra partizione resa irraggiungibile.
* **Collisioni:** in un sistema di piccole dimensioni possono verificarsi collisioni che potrebbero rendere impossibile salvare alcune risorse poiché sono già presenti altre con lo stesso ID.