# Progetto B1: Sistema Chord

Simone Festa
*mat. 0320408*
*Sistemi distribuiti e cloud computing*
Laurea magistrale in Ingegneria Informatica, Tor Vergata
simone.festa@students.uniroma2.eu

***Abstract*—Il seguente documento descrive le scelte progettuali e strutturali adottate per la realizzazione di un sistema composto da nodi disposti in un anello, seguendo l'organizzazione di Chord. Saranno esposte le seguenti sezioni: tecnologie utilizzate, architetture adottate, scelte tecniche effettuate e problematiche riscontrate.**

## I. Introduzione

Il progetto ha l'obiettivo di simulare l'interazione di un insieme di nodi disposti in un anello, seguendo il protocollo e l'algoritmo fornito da Chord per un sistema peer-to-peer con una hash table distribuita. Il sistema deve essere in grado di gestire la comunicazione e lo scambio di dati tra i nodi. La simulazione del modello è stata realizzata attraverso la virtualizzazione (utilizzando Docker) e l'utilizzo del servizio EC2 di Amazon.

## II. Requisiti e tecnologie

La realizzazione del modello ha richiesto l'utilizzo delle tecnologie descritte di seguito:

- Linguaggi utilizzati:
  - **Go**: La scelta di di tale linguaggio è imposta dai requisiti del progetto. La logica di Chord viene descritta mediante questo linguaggio.
  - **Python**: L'utilizzo di questo linguaggio ha permesso la stesura automatizzata e parametrizzata del file docker-compose.yml
  - **Bash**: Utilizzato per la generazione isolata di un nodo postumo alla creazione dell'anello.
- Altre tecnologie:
  - **Docker**: Necessario per realizzare la virtualizzazione a livello di sistema operativo, tramite l'isolamento delle risorse del kernel Linux.
  - **Docker Compose**: Utilizzato per gestire un sistema multi-container.
  - **Amazon EC2**: Amazon Elastic Compute Cloud (Amazon EC2) è il servizio cloud utilizzato per lo sviluppo dell'applicativo su macchine virtuali nel cloud di Amazon.
  - **goRPC**: Libreria utilizzata per gestire le chiamate a procedura remota, in grado di minimizzare il numero di syscall necessarie per le connessioni, l'invio e la ricezione di dati.

Il sistema utilizza anche file *.json* per configurare i parametri che caratterizzano il sistema (numero di bit usati per gli identificatori e numero di nodi nel sistema). Inoltre, fa uso della libreria *crypto/sha* per utilizzare SHA-1 come meccanismo di hash. È importante gestire accuratamente il risultato di questa funzione in modo da ottenere uno spazio continuo, consentendo ai nodi e alle risorse di essere mappati sullo stesso spazio di indirizzamento.

## III. Descrizione del sistema

### A. Struttura

Il sistema proposto segue una configurazione ad **anello**. In particolare, adottiamo un **overlay network strutturato**, che rappresenta la disposizione dei nodi. Questo tipo di disposizione non dipende dalla distanza fisica tra i nodi, ma mantiene una struttura ben definita. Sebbene questo approccio permetta di ottenere tempi di ricerca rapidi, nell'ordine di $O(\log N)$, richiede una maggiore complessità per le operazioni di inserimento o cancellazione dei nodi. Questa complessità deriva dalla necessità di mantenere la struttura stessa. Di seguito è mostrato un esempio grafico del sistema:

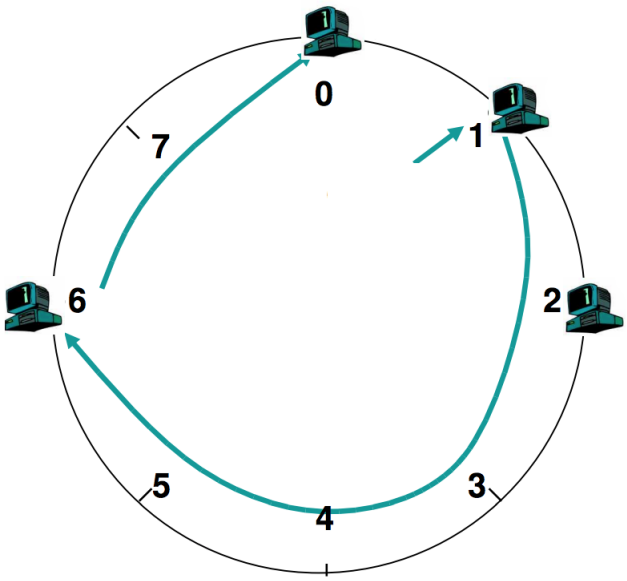


Fig. 1. Rappresentazione della disposizione dei nodi nell'anello.

Attraverso una tecnica specifica di hash, nota come **consistent hashing**, i nodi e le risorse sono mappati nello stesso spazio, rappresentato dall'anello, utilizzando una funzione hash standard come *SHA-1*. Questo processo di mappatura è applicato uniformemente sia ai nodi che alle risorse. Uno dei principali vantaggi del consistent hashing è la riduzione del numero di chiavi che richiedono di essere riassegnate in caso di modifiche alla tabella hash, come ad esempio un ridimensionamento. Nel nostro modello, questo si traduce in un impatto limitato sulle Finger Table dei nodi durante le operazioni di *join* o *leave* nel sistema. Ogni nodo gestisce un intervallo di chiavi hash *contigue*. Pertanto, ciascun nodo è responsabile delle chiavi che cadono tra il suo predecessore e se stesso (incluso), seguendo un percorso orario.

### B. Finger Table

L'introduzione della Finger Table rappresenta un compromesso efficace per consentire a ogni nodo di acquisire una conoscenza parziale del sistema a cui appartiene. Se ogni nodo dovesse possedere informazioni complete su tutti gli altri nodi nell'anello, si otterrebbe un lookup ottimale, ma con sistemi di grandi dimensioni, la quantità di informazioni da memorizzare diventerebbe considerevole. D'altro canto, l'approccio opposto, ovvero la mancanza di memorizzazione di qualsiasi informazione sui componenti, genererebbe un carico di lavoro eccessivo su un server centrale, creando un collo di bottiglia. Attraverso l'utilizzo delle *Finger Table*, si mantiene una lista di nodi progressivamente distanti. Questo permette di avere una conoscenza ben definita dei nodi vicini e, all'aumentare della distanza, una conoscenza più approssimativa ma comunque significativa.

Ogni nodo possiede una propria Finger Table, le quali presentano una stessa dimensione. Il numero di righe è definito da un parametro comune, **m =** #GUID bits. La Finger Table di un nodo $p$, avente $m$ righe, avrà la singola riga $i$ definita da: $FT_p[i] = \text{succ}(p + 2^{i-1}) \bmod 2^m, \quad 1 \leq i \leq m$ dove $\text{succ}(p+2^{i-1})$ è il successore *i-esimo*, calcolato iterativamente dal nodo successore. In questo contesto, ogni nodo possiede informazioni riguardanti il suo predecessore e il suo successore all'interno dell'anello. Durante il processo di calcolo di una riga nella Finger Table, quando un nodo ha la necessità di determinare quale nodo gestisce un certo identificativo, esso interroga direttamente il suo successore per verificare se è proprio lui il nodo responsabile. Nel caso in cui il successore non sia il nodo cercato, quest'ultimo procederà a chiedere al suo successore e così via, iterativamente, fino a quando il nodo corretto verrà individuato. Attraverso questo metodo, l'intervento del server Registry non è necessario nel processo di calcolo della Finger Table. Di conseguenza, il sistema è in grado di eseguire questa operazione senza dipendere da un'entità centralizzata. Il compito del Registry si limita a fornire al nodo appena entrato nel sistema informazioni riguardo ai suoi nodi vicini.

Di seguito, viene proposta una rappresentazione di una Finger Table:

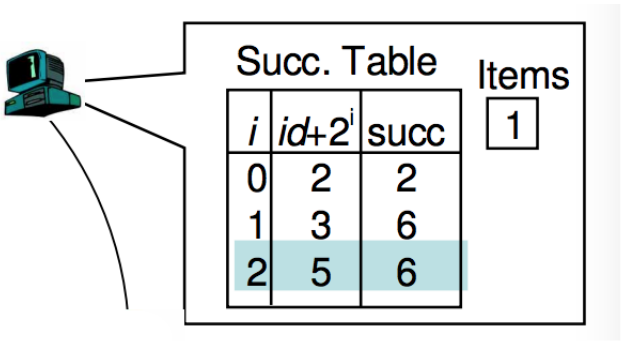


Fig. 2. FT per un nodo avente *id = 1*. La tabella mantiene il successore più vicino a lui (2), ed altri più distanti in ordine (6).

### C. Server Registry

Il server Registry è dotato di un indirizzo IP statico che risulta accessibile agli attori principali all'interno del sistema. Le sue funzionalità chiave comprendono:

- Fornire supporto all'inizializzazione dei nodi che intendono entrare nel sistema. A tal fine, il server Registry fornisce le informazioni necessarie affinché un nodo possa collocarsi in modo corretto all'interno del sistema. Queste informazioni includono i dettagli riguardanti il nodo predecessore e il nodo successore, nonché eventuali risorse che il nodo deve gestire.
- Consentire al client di interagire con il sistema. Tra le operazioni possibili, rientrano l'inserimento, la ricerca e la cancellazione di risorse. E' anche offerta la possibilità di rimuovere nodi presenti nel sistema, portando ad una riallocazione delle risorse in modo appropriato.

## IV. Operazioni a disposizione

Il client avrà a disposizione un insieme di operazioni per interagire con il sistema, consentendo la gestione delle risorse e dei nodi secondo le necessità. Di seguito sono elencate le operazioni disponibili:

- **Inserimento di una risorsa**: Questa operazione permette di memorizzare una risorsa fornita come input nel sistema. Durante questa operazione, viene calcolato un identificativo univoco per la risorsa, che sarà utilizzato per associare la risorsa al nodo responsabile della sua gestione.
- **Ricerca di una risorsa**: Utilizzando l'identificativo di una risorsa, è possibile ottenere il valore della risorsa stessa e il nodo che ne è responsabile, qualora la risorsa sia presente nel sistema.
- **Rimozione di una risorsa**: Specificando l'identificativo di una risorsa presente nel sistema, è possibile procedere alla sua rimozione.
- **Rimozione di un nodo**: Fornendo l'identificativo di un nodo, è possibile eliminarlo dal sistema. Durante questa operazione, le risorse possedute dal nodo verranno redistribuite ad altri nodi, seguendo la stessa logica di assegnazione utilizzata per le nuove risorse.

- **Aggiunta di un nodo**: Utilizzando uno script Bash, è possibile avviare un nuovo nodo all'interno del sistema. Il nuovo nodo contatterà il server Registry per determinare la sua posizione all'interno dell'anello. Successivamente, comunicherà con il suo successore per acquisire eventuali risorse da gestire.

## V. Descrizione della logica del sistema

### A. Uso delle funzioni Hash

La funzione hash utilizzata nel sistema è **SHA-1** di tipo crittografica. L'obiettivo principale è minimizzare le collisioni, ovvero i casi in cui due input diversi producono lo stesso output hash. Le collisioni rappresenterebbero un problema, poiché potrebbero limitare l'aggiunta di nuovi nodi o risorse nel caso in cui i loro identificativi siano mappati su identificativi esistenti. Il parametro **m =** #GUID rappresenta il numero di bit utilizzato per definire la dimensione dello spazio di hash. Per calcolare l'intervallo di hash ridotto, si esegue il calcolo di $2^m$. Questo è particolarmente importante nell'ambito del consistent hashing, dove è necessario definire un intervallo limitato in cui distribuire i valori hash. Tramite oggetto Hash, si esegue la computazione per l'oggetto *key*, sfruttando l'operatore *XOR*. In particolare, iterando per ogni gruppo di hash calcolato (byte per byte), si esegue uno *XOR* e si applica una riduzione *modulo N*, favorendo la distribuzione dei bit in tale intervallo.

### B. Join di un nodo del sistema

Il sistema è in grado di gestire l'aggiunta di nuovi nodi all'interno di un sistema già avviato. Quando un nuovo nodo entra nel sistema, esegue le seguenti fasi:

1) **Comunicazione col Registry**: Il Registry fornisce al nuovo nodo le informazioni necessarie sul nodo precedente e il successore più vicino nel sistema. Queste informazioni consentono al nuovo nodo di collocarsi correttamente all'interno dell'anello tra i nodi già presenti.
2) **Collocazione nel sistema**: Una volta ottenute le informazioni sul nodo precedente e il successore, il nuovo nodo si posiziona tra di loro nell'anello, comunicando ai suoi nodi adiacenti la sua presenza. Questo assicura che il nuovo nodo sia inserito in modo coerente nel sistema.
3) **Verifica e trasferimento delle risorse**: Il nuovo nodo contatta il suo successore, verificando se ci sono risorse assegnate a lui che devono essere gestite. In caso positivo, queste risorse vengono trasferite dal nodo successore al nuovo nodo.
4) **Rimozione delle risorse dal nodo successore**: Dopo il trasferimento delle risorse, il nodo successore rimuove le risorse trasferite dalla sua responsabilità. Questo assicura che le risorse siano correttamente assegnate al nuovo nodo.

### C. Leave di un nodo del sistema

Il sistema permette la rimozione controllata di un nodo nel sistema. In particolare, specificato l'identificativo del nodo da rimuovere:

1) **Comunicazione con il server Registry**: Fornito al server Registry l'identificativo del nodo da rimuovere, questi contatterà il nodo interessato.
2) **Trasferimento delle chiavi al successore**: Il nodo prossimo all'uscita dovrà trasferire le sue risorse al successore, il quale sarà il nuovo responsabile di queste chiavi.
3) **Comunicazione con il nodo successore**: Il nodo in uscita informa il nodo successore che il suo predecessore non è più il nodo in questione, bensì il predecessore del nodo in questione.
4) **Comunicazione con il nodo predecessore**: Il nodo uscente contatterà il nodo predecessore, al quale verrà comunicato unicamente il suo nuovo successore, non essendoci alcun trasferimento di risorse in questo caso.

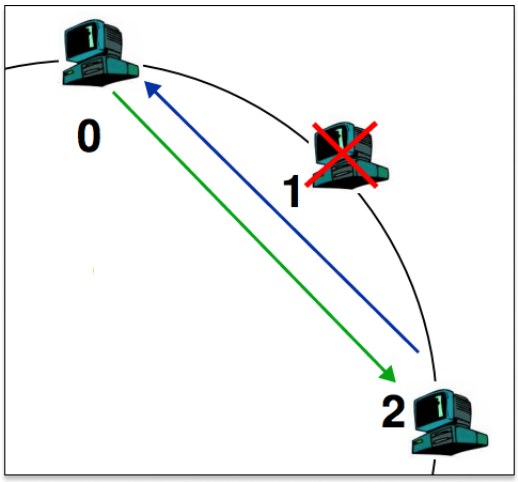


Fig. 3. Il nodo *1*, uscente, comunica al nodo *0* che il suo nuovo successore sarà il nodo *2*, e viceversa. Il nodo *2* riceve le risorse possedute dal nodo uscente.

## VI. Fault tolerance

Il sistema non presenta un meccanismo di replicazione per la gestione delle risorse nel caso in cui un nodo lasci l'anello in modo non pianificato. Tuttavia, il sistema è in grado di riorganizzare correttamente i nodi rimanenti. Ciascun nodo mantiene comunicazioni periodiche con il suo successore al fine di mantenere aggiornata la Finger Table (come illustrato nella sezione B: Finger Table). Se un nodo non riesce più a stabilire una comunicazione con il suo successore, notificherà il Registry. Quest'ultimo cercherà di stabilire una connessione con il nodo per confermare l'eventuale guasto. Una volta confermata l'assenza del nodo, il Registry procederà a mettere in comunicazione i due nodi adiacenti a quello caduto, perdendo però le risorse precedentemente gestite da quel nodo, come spiegato in precedenza.

## VII. Problematiche

Le principali limitazioni osservate riguardano:

- **Possibilità di collisioni**: In un sistema di piccole dimensioni, si possono osservare collisioni tra nodi o tra risorse.

Questo può portare, nel caso dei nodi, ad un sistema con meno entità di quelle volute. Nel caso di collisioni tra risorse, potremmo memorizzare meno oggetti di quelli voluti.

- **Server Registry Bottleneck**: Il server Registry rappresenta un collo di bottiglia per il sistema, in quanto oltre a mitigare il collegamento con il client, deve anche mantenere le informazioni necessarie per il corretto inserimento o rimozione dei nodi nel sistema in esame, come una lista ordinata dei nodi del sistema.
- **Partizionamento della rete**: Il sistema non è tollerante rispetto un partizionamento della rete. Una eventuale partizione obbligherebbe il Registry ad essere presente in solo una delle due reti create, lasciando l'altra senza possibilità di gestirsi in modo isolato.