# 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 seguite per la realizzazione di un sistema composto da nodi disposti in un anello, seguendo l'organizzazione di Chord. Verranno esposte, di seguito: tecnologie, architetture e scelte tecniche adottate.**

## I. Introduzione

Il progetto ha l'obiettivo di simulare l'interazione di un insieme di nodi disposti ad anello, secondo 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 mediante virtualizzazione (tramite Docker) e tramite il servizio EC2 di Amazon.

## II. Requisiti e tecnologie

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

- Linguaggi utilizzati:
  - **Go**: L'utilizzo di tale linguaggio è dovuto ai 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 isolamento delle risorse del kernel Linux.
  - **Docker Compose**: Utilizzato per gestire un sistema di multi-container.
  - **Amazon EC2**: Amazon Elastic Compute Cloud (Amazon EC2) è il service cloud utilizzato per lo sviluppo dell'applicativo su macchine virtuali nel cloud di Amazon.
  - **goRPC**: Libreria per gestire le chiamate a procedura remota, in grado di minimizzare il numero di syscall necessarie per connessioni, invio e ricezione di dati.

Il sistema fa anche utilizzo di file .json per la configurazione di parametri definenti il sistema, oltre alla libreria *crypto/sha* per l'utilizzo di SHA-1 come meccanismo di Hash. Il risultato di tale funzione deve essere gestito adeguatamente per poter ottenere uno spazio continuo, e permettere a nodi e risorse di essere mappate sullo stesso spazio di indirizzamento.

## III. Descrizione del sistema

### A. Struttura

Il sistema proposto presenta una disposizione ad **anello**. In particolare, trattiamo un **overlay network strutturato**, ovvero una disposizione dei nodi non dipendente dalla distanza fisica tra essi, ma che comunque presenta una struttura da mantenere. Tale approccio consente tempi di lookup ridotti, nell'ordine di $O(\log N)$, a scapito di una complessità maggiore in operazioni di inserimento o cancellazione dei nodi, data dalla necessità di mantenere la struttura. Di seguito viene mostrato graficamente un esempio del sistema:

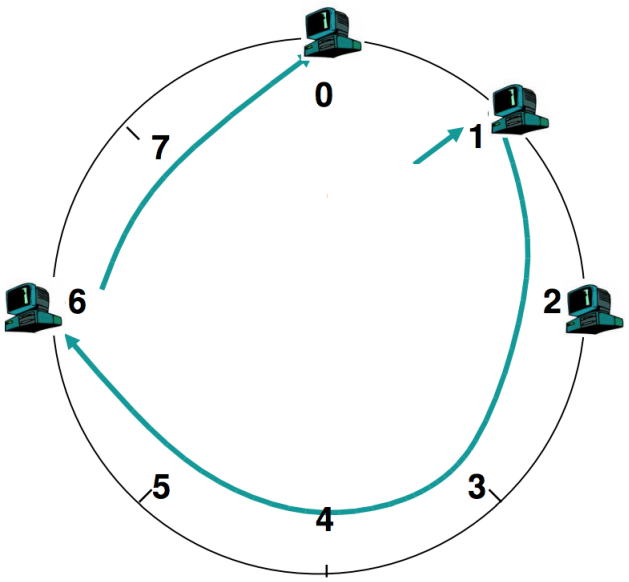


Fig. 1. Rappresentazione della disposizione dei nodi nell'anello.

Mediante una particolare tecnica di hashing, definita come **consistent hashing**, nodi e risorse vengono mappati su un medesimo spazio (l'anello) tramite una funzione hash standard, come SHA-1, mediante lo stesso processo di mappatura. Il vantaggio maggiore del consistent hashing è la riduzione del numero di chiavi che necessitano di unin un remapping in caso di resize della tabella hash. Nel nostro modello, ciò si

traduce nel basso impatto delle modifiche richieste alle Finger Table dei nodi in situazioni di join/leave nel sistema. Ogni nodo gestisce un intervallo di chiavi hash *contigue*. Ogni nodo è responsabile delle chiavi poste tra il suo predecessore e se stesso (incluso), seguendo un percorso orario.

### B. Finger Table

L'introduzione della Finger Table rappresenta un ottimo compromesso per permettere ad ogni nodo di avere una conoscenza parziale del sistema di cui fa parte. Se ogni nodo avesse una conoscenza completa degli altri nodi nell'anello, avremmo un lookup ottimale, ma con grandi sistemi risulterebbero non banali il numero di informazioni da mantenere. Un approccio inverso, ovvero la non memorizzazione di alcun componente, portebbere ad un eccessivo carico di lavoro svolto da un server (il quale dovrebbe conoscere tutti gli elementi dell'anello) creando un bottleneck. Tramite *Finger Table* si mantiene una lista di nodi progressivamente più distanti. Ciò permette di avere una conoscenza dei nodi vicini ben definita, ed all'aumentare della distanza una conoscenza più vaga ma non nulla. Ogni nodo possiede una una propria Finger Table, le quali presentano una stessa dimensione. Il numero di righe è definito da un parametro comune definito da **m =** #GUID bits. La Finger Table di un nodo $p$, avente $m$ righe, avrà la singola riga $i$ definita da: $FT_p[i] = \text{succ}(p + 2^{i-1}) \bmod 2^m$, $1 \leq i \leq m$

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

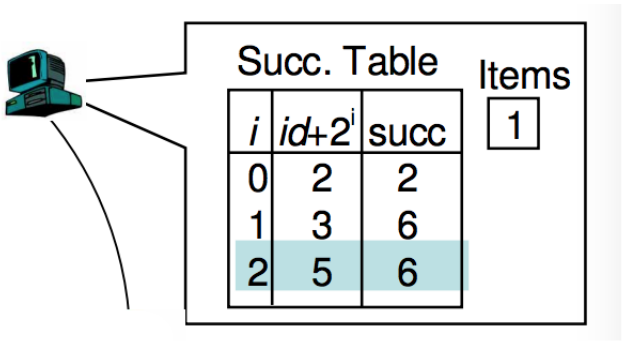


Fig. 2. FT per un nodo avente *id = 1*. La tabella mantiene il successore più vicino a lui (2), ed altri più distanti in ordine (6).

### C. Server Registry

Il server registry presenta un IP statico ed accessibile a due attori principali nel sistema:

- Permette l'inizializzazione dei nodi che vogliono entrare nel sistema, fornendo le informazioni necessarie per consentire ad un nodo di collocarsi correttamente nel sistema.
- Consente ad un client l'interazione con il sistema. In particolare, sono possibili operazioni di inserimento, ricerca e cancellazione di risorse, oltre alla rimozione di nodi presenti nel sistema, con conseguente ridistribuzione delle risorse.

## IV. Operazioni a disposizione

Il client avrà a disposizione le seguenti operazioni da poter eseguire.

- **Inserimento di una risorsa**: Tramite questa operazione, una risorsa fornita in input viene memorizzata nel sistema e ne viene calcolato l'identificativo, successivamente utilizzato per associarla al nodo responsabile della sua gestione.
- **Ricerca di una risorsa**: E' possibile, fornendo l'id di una risorsa, ottenere il valore di tale risorsa ed il nodo che la gestisce.
- **Rimozione di una risorsa**: Fornendo l'id di una risorsa presente nel sistema, si procederà alla sua rimozione.
- **Rimozione di un nodo**: Specificando l'id del nodo desiderato, questo verrà eliminato dal sistema, e le risorse da lui possedute verranno associate ad altri nodi, mediante la stessa logica di assegnazione di una nuova risorsa.
- **Aggiunta di un nodo**: Tramite uno script Bash, è possibile avviare un nuovo nodo. Questi contatterà il server Registry per trovare la sua posizione nell'anello. Successivamente contatterà il suo successore per prelevare eventuali risorse che il nuovo nodo dovrà gestire.

## V. Descrizione della logica del sistema

### A. Uso delle funzioni Hash

La funzione hash utilizzata è **SHA-1**, di tipo crittografica. L'obiettivo è ridurre al più possibile le *collisioni*, in quanto queste limiterebbero l'aggiunta di nuovi nodi o nuove risorse, nel caso il loro identificato venisse mappato su un identificativo già esistente. Definito il parametro **m =** #GUID bits, si procede calcolando $2^m$, in quanto si lavora in un contesto di consistent hashing, ed è quindi necessario definire un intervallo nel quale ridurre il valore hash. Tramite oggetto hash, si esegue la computazione per l'oggetto *key*, sfruttando l'operatore *XOR*. In particolare, interando 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'arrivo di un nuovo nodo nel sistema. Il nodo entrante, dopo aver comunicato con il Registry, otterrà informazioni sul nodo precedente e successore, collocandosi tra essi. Successivamente contatterà il nodo successore per verificare la presenza di risorse che dovranno essere gestite dal nodo entrante. Queste verranno rimosse dalla competenza del nodo successore e passate al nuovo nodo.

### C. Leave di un nodo del sistema

Il sistema permette la rimozione controllata di un nodo nel sistema. In particolare, fornito al server Registry l'identificato del nodo da rimuovere, questi contatterà il nodo interessato. Tale nodo dovrà trasferire le sue risorse al successore, comunicargli che il suo predecessore non è più il nodo in questione, bensì il precedecessore del nodo in questione. Tale operazione verrà svolta anche col predecessore del nodo interessato, al quale verrà comunicato unicamente il suo nuovo successore.

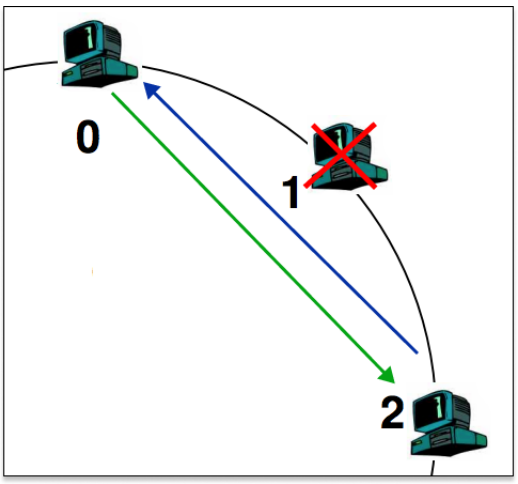


Fig. 3. Il nodo *1*, uscente, comunica al nodo *0* che il suo nuovo successore sarà il nodo *2*, e viceversa. Il nodo *2* riceve le risorse possedute dal nodo uscente.

## VI. Miscellanea

### A. Fault tolerance

Il sistema non presenta un meccanismo di replicazione. Una rimozione inattesa e non controllata di un nodo non permette al Registry di mantenere una lista coerente dei nodi presenti.

### B. Problematiche

La principale 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 delle risorse. Nel caso di collisioni tra risorse, potremmo memorizzare meno oggetti di quelli necessari.
- **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.
- **Partizionamento della rete**: Il sistema non è tollerante rispetto un partizionamento della rete.