Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il management

Ricerca per parole chiave basata su DHT

Relatore:
Chiar.mo Prof.
STEFANO FERRETTI
Correlatore:
Chiar.mo Dott.
MIRKO ZICHICHI

Presentata da:
ALESSANDRO MEMOLI

II Sessione
Anno Accademico 2019/2020

Introduzione

Questo lavoro di tesi mira a fornire una panoramica sulle tecnologie utilizzate nella classe dei sistemi distribuiti, con considerevole enfasi sulle DHT (Distributed Hash Table).

In particolare, verranno descritti i vantaggi che queste reti offrono rispetto ai tradizionali approcci.

Fondamentale sarà sottolineare come le operazioni di memorizzazione, di ricerca e di recupero delle informazioni siano più efficaci da eseguire su sistemi P2P, in quanto permetto il funzionamento corretto delle applicazioni che ne fanno uso. Sarà presentato un famoso protocollo di DHT, Chord. Questo protocollo si differenzia dai servizi tradizionali, in quanto fornisce una scalabilità migliore, ha un tipo di memorizzazione più semplice e presenta una routing table senza incoerenze che viene detta stabile.

Verranno, tuttavia, descritti alcuni dubbi che riguarderanno la ricerca per parole chiave. Pertanto, si analizzerà un sistema preso come riferimento tra le soluzioni a tale problema. Verrà quindi descritta una struttura ad ipercubo a r-dimensioni, con una stringa composta da r-bit. Verrà approfondita applicandola poi ad una struttura applicata a una tabella hash distribuita. Il focus principale della tesi,una volta data l'infarinatura generale dei vari concetti, è quello di analizzare una particolare tecnica di ricerca di oggetti all'interno di reti P2P che evitano problemi di un- balanced load, hot spots, fault tolerance, storage redundancy, e unable to facilitate ranking. La struttura logica che seguirà è la seguente:

Il capitolo 1, dando uno sguardo alle Distributed Hash Table, si focalizzerà

sul protocollo Chord.

Nel capitolo 2 verrà il problema delle keyword search in reti p2p.

Il capitolo 3 seguirà la struttura del capitolo due, rappresentando le soluzioni proposte attraverso l'utilizzo di simulazioni atte a rappresentarne i vantaggi.

Il capitolo 4 sarà dedicato ad alcune considerazioni conclusive.

Indice

Capitolo 1

DHT

In questo capitolo vengono presentati gli algoritmi di tabella hash distribuita (DHT) utilizzati per implementare reti P2P strutturate.
Inizialmente vengono presentate le caratteristiche di base del DHT, e successivamente verrà analizzato un esempio (Chord).
I capitoli successivi analizzeranno una possibile alternativa alla ricerca di keyword, introducento il concetto di Hypercube.

1.1 Sistemi Peer-to-Peer

I sistemi peer-to-peer sono sistemi distribuiti che non presentano alcun controllo centralizzato. Tra i set di funzionalità che caratterizzano le recenti applicazioni peer-to-peer vi sono:

- **Archiviazione ridondante**
- **Selezione di server vicini**
- **Autenticazione**
- **Denominazione gerarchica**

Tuttavia, nella maggior parte dei sistemi peer-to-peer, l'operazione principale è **la localizzazione efficiente degli elementi di dati**.

Un'istanza di un programma che fornisce sia le funzionalità di server che di client è detta Peer ed il paradigma che lo sfrutta è chiamato Peer to Peer (P2P), ovvero da pari a pari, per sottolineare quanto siano equamente importanti ogni nodo della rete.
La diffusione di questo nuovo paradigma offre vantaggi notevoli rispetto a quello che è il tradizionale approccio client-server. L'assenza infatti di un punto di centralizzazione rende il sistema più tollerante ai guasti permettendone l'utilizzo anche quando una parte consistente dei nodi della rete non è funzionante[3]. Ciò che è fondamentale sottolineare è che mentre in un sistema centralizzato operazioni come la memorizzazione, la ricerca ed il recupero delle informazioni si eseguono semplicemente, in un sistema P2P queste risultano essere più complesse: ogni nodo della rete infatti deve essere in grado di eseguire queste operazioni in modo da permettere il funzionamento corretto delle applicazioni che la utilizzano, a prescindere dalla disposizione fisica dei dati.

1.1.1 Sistemi non strutturati

Nei sistemi non strutturati le risorse messe a disposizione di un peer sono memorizzate dal peer stesso.
In questo caso, il problema è caratterizzato dalla ricerca. Dove possiamo trovare l'informazione che presenti le caratteristiche desiderate?

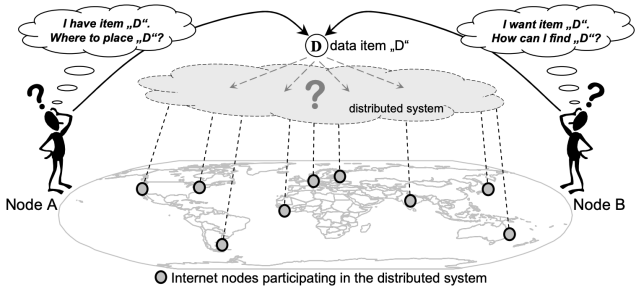


Figura 1.1: DHT

1.1.2 Sistemi strutturati

Nei sistemi strutturati invece l'informazione condivisa può essere memorizzata su un nodo qualsiasi della rete e le informazioni associate ai nodi si basa su un criterio ben preciso che permette a un routing "intelligente" alcune query verso i nodi che le soddisfano.

Quali sono, dunque, i meccanismi che vengono utilizzati per decidere dove deve essere memorizzata l'informazione e quale mezzo utilizzare per recuperare tale informazione? Di seguito vengono descritte le caratteristiche che costituiscono una rete p2p cosìddetta *distributed hash table*.

1.2 Distributed Hash Table

Per organizzare un sistema distribuito, bisogna in primo luogo tenere particolarmente in considerazione la scalabilità del sistema. Il problema della scalabilità viene affrontato con l'arrivo di una nuova categoria di reti Peer To Peer: **le distributed hash table (DHT)**.

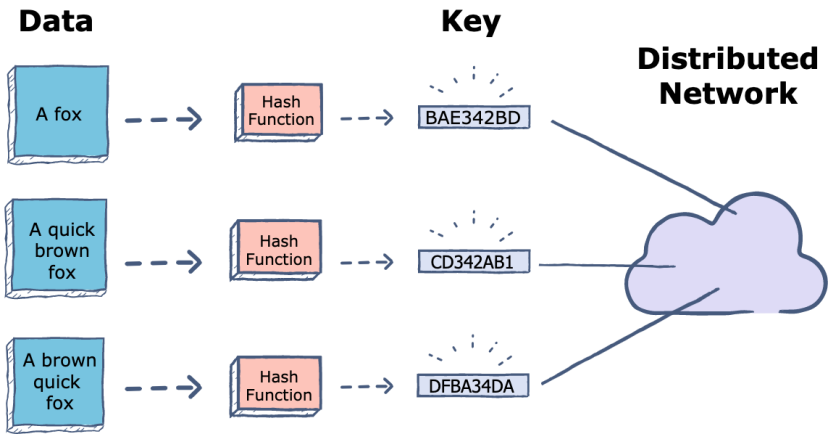


Figura 1.2: DHT

Con questo protocollo, ogni peer che partecipa a una distributed hash table può immettere i dati usando una forma di coppia chiave-valore e può recuperare qualsiasi informazione presente nei dati e immessa da un altro nodo. Le chiavi sono spartite omogeneamente in uno spazio logico, cosiddetto spazio degli identificatori e la DHT determina le caratteristiche di tale spazio.

Tali DHT forniscono un'operazione di ricerca dei dati che garantisce di trovare l'informazione cercata se queste è presente nella rete; inoltre questa operazione di ricerca risulta essere veloce e scalabile dal momento che è un'operazione distribuita che utilizza poche risorse.
Le funzionalità fornite sono molto simili a quelle fornite da una hash table, in quanto gestiscono le informazioni come coppie di chiavi e valori, rispettivamente utilizzati per registrare e ricercare i dati da inserire nella rete.
Tuttavia, il set di coppie chiave-valore viene archiviato in modo distribuito

ed è condiviso tra i nodi della tabella hash distribuita (computer collegati alla rete DHT).

La funzionalità fondamentale è l'efficiente posizione dei nodi e delle risorse. In questi sistemi, a ogni nodo viene assegnato un identificatore univoco, utilizzato per individuare il nodo nello spazio dell'ID virtuale e le risorse vengono solitamente archiviate dai nodi determinati in base alle chiavi delle risorse. La chiave della risorsa è generalmente il risultato di una funzione hash sulla risorsa. La quantità totale delle risorse utilizzate da un nodo per la gestione delle chiavi ad esso assegnate determina il suo carico di lavoro.

Le DHT hanno le seguenti proprietà:

- **Decentralità** I nodi formano collettivamente il sistema senza alcun coordinamento centrale.
- **Scalabilità** Il sistema dovrebbe funzionare in modo efficiente anche con migliaia o milioni di nodi.
- **Tolleranza ai guasti** Il sistema dovrebbe risultare affidabile anche in presenza di nodi che entrano, escono dalla rete o sono soggetti a malfunzionamenti con elevata frequenza.

1.3 Problema della ricerca

I sistemi peer-to-peer e le applicazioni distribuite sollevano molte domande di ricerca. A causa del carattere completamente decentralizzato, uno dei problemi di maggior rilievo è quello della memorizzazione e del reperimento di un dato. Avendo un nodo A che vuole memorizzare un dato all'interno della reta P2P e successivamente un secondo nodo B vuole accedere al dato memorizzato da A, il problema si riassume in due principali domande, dove memorizzare e come reparire,in maniera efficiente il dato in questione?

Diversi protolli già esistenti da tempo sono riusciti a risolvere questo tipo di problema in maniera scalabile e con una complessità temporale logaritmica

rispetto al numero di nodi. Di seguito sarà esposto uno dei tanti protocolli presenti, Chord.

1.4 Chord

Questo protocollo specifica come trovare le posizioni delle chiavi, come i nuovi nodi si uniscono al sistema e come ripristinare il guasto (o la partenza pianificata) dei nodi esistenti. Le caratteristiche che distinguono Chord da molti protocolli di ricerca peer-to-peer sono la sua semplicità, la sua dimostrabile correttezza e le sue comprovabili prestazioni. Chord è una DHT che definisce un protocollo che supporta un'unica operazione, quella di mappare una chiave in un nodo che appartiene alla rete. A seconda dell'applicazione che utilizza Chord, quel nodo può essere responsabile della memorizzazione di un valore associato alla chiave[1].

Per eseguire l'associazione della chiave al nodo, Chord utilizza una variante del Consistent Hashing, la quale tende a bilanciare il carico, facendo in modo che ogni nodo abbia un numero ben distribuito di chiavi ed evitando così il sovraccarico dei nodi. La funzione Consistent Hashing assegna a ogni nodo e chiave un identificatore m-bit utilizzando SHA-1 come funzione hash di base. L'identificatore di un nodo viene scelto mediante l'hashing dell'indirizzo IP del nodo, mentre un identificatore della chiave viene prodotto mediante l'hashing della chiave. Il termine "chiave" si riferisce sia alla chiave originale che alla sua immagine sotto la funzione hash, poiché il significato sarà chiaro dal contesto.
Allo stesso modo, il termine "nodo" si riferirà sia al nodo che al suo identificatore sotto la funzione hash. La lunghezza dell'identificatore m deve essere abbastanza grande da rendere trascurabile la probabilità che due nodi o chiavi abbiano lo stesso identificatore.
Il consistent hashing è parte integrante della robustezza e delle prestazioni di Chord le chiavi e i nodi vengono distribuiti uniformemente nello stesso spazio dell'identificatore e con una minima e trascurabile possibilità di collisione.

Questo fa sì che i nodi si uniscano lasciando la rete senza interruzioni.

1.4.1 Routing

Mentre Chord mappa le chiavi sui nodi, i servizi tradizionali di nome e localizzazione forniscono una mappatura diretta tra chiavi e valori. Un valore può essere un indirizzo, un documento o un elemento arbitrario di dati. Chord implementa questa funzionalità in maniera semplice, memorizzando cioè ogni coppia chiave–valore nel nodo a cui è mappata quella chiave.

L'identificatore logico (ID) viene assegnato ai nodi e alle chiavi autonomamente dal peer che richiede l'informazione e in entrambi i casi viene effettuata tramite una normale funzione di hash. L'assenza di punti di centralizzazione nella routing table permette a Chord di fornire un'ottima scalabilità in termini di numero di peer che partecipano alla rete.

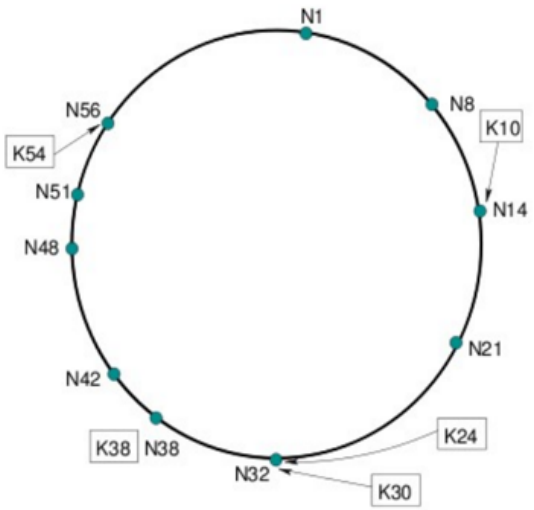


Figura 1.3: Illustrazione 1: rappresentazione dello spazio degli identificatori di una rete Chord con 10 nodi e 5 chiavi: è mostrato l'assegnamento delle chiavi ai nodi.

Chord organizza lo spazio degli identificatori come un anello, uno spazio circolare unidimensionale m-bit, per cui tutte le operazioni aritmetiche sono

modulo 2^m[2]. La funzione di distanza tra due identificatori utilizzata è $d : (id1–id2 + 2^m)mod2^m$: le chiavi sono così assegnate al primo nodo il cui ID è uguale o maggiore del ID della chiave.

Per formare una sovrapposizione ad anello, ogni nodo n mantiene due puntatori ai suoi vicini immediati (Fig. 1.4).

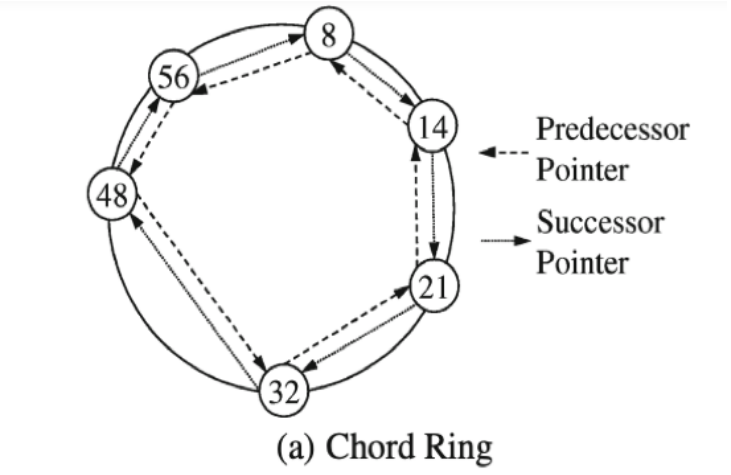


Figura 1.4: Illustrazione 1: Chord Ring

Ogni nodo ha un successore e un predecessore. Il successore del nodo rappresenta il nodo successivo nel cerchio dell'identificatore in senso orario. Il predecessore è invece il successivo in senso antiorario. Se per ogni ID possibile c'è un nodo, il successore del nodo 0 è il nodo 1 e il predecessore del nodo 0 è il nodo $2^m - 1$.

Il concetto di successore viene utilizzato anche per le chiavi, dove il nodo successore di una chiave k è il primo nodo in cui ID è uguale a k o segue k nel cerchio dell'identificatore, detto successor(k). Ogni chiave viene poi memorizzata nel suo nodo successore, così cercare una chiave k significherà interrogare successor(k). Il puntatore del successivo punta al successore (n), ossia il vicino immediato in senso orario.
Nello stesso modo, il puntatore del predecessore punta al predecessore (n), il vicino immediato in senso antiorario.

Chord poi mappa la chiave k sul successore (k), ossia il primo nodo il cui identificatore è uguale o maggiore di k nello spazio dell'identificatore.

Pertanto, il nodo n è responsabile per le chiavi nell'intervallo di (predecessore (n), n], ovvero chiavi che sono maggiori del predecessore (n) ma minori o uguali a n. Tutte le coppie chiave-valore la cui chiave è uguale a k vengono quindi memorizzate sul successore (k) indipendentemente da chi possiede le coppie chiave-valore: questa è la distribuzione dell'elemento dati.
Trovare la chiave k implica che instradiamo una richiesta al successore (k). L'approccio più semplice per questa operazione, come illustrato in Fig. 4, consiste nel propagare una richiesta lungo l'anello Chord in senso orario fino a quando la richiesta arriva al successore (k).
Tuttavia, questo approccio non è scalabile poiché la sua complessità è O (N), dove N indica il numero di nodi nell'anello.

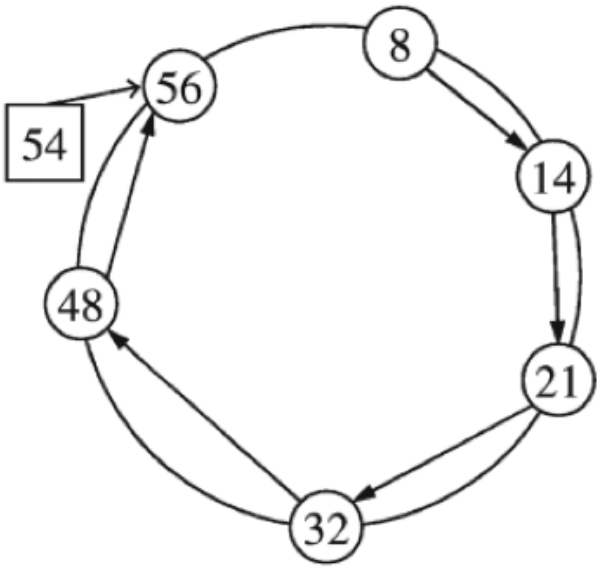


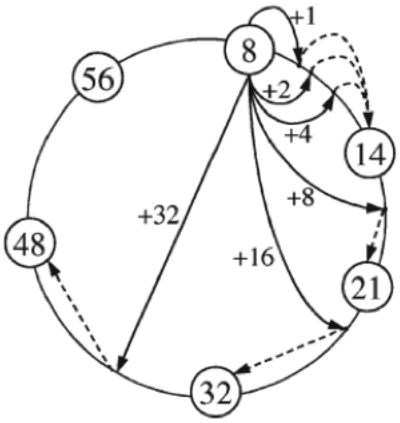
Figura 1.5: Illustrazione 1: Trova il successore

1.4.2 Finger Table

In Chord la porzione della routing table locale ad un nodo è chiamata finger table. Per accelerare il processo di ricerca del successore (k), ogni nodo

n mantiene una finger table di m voci (Fig. 1.6). m è un numero il cui valore è generalmente 128 o 160.
Ogni voce nella finger table è anche chiamata dito (finger). La finger table di ogni nodo è un array, che in ogni casella contiene un collegamento ad un altro nodo con la seguente caratteristica: nella i – esima casella vi è inserito il collegamento al primo nodo noto il cui ID supera l'identificatore del nodo corrente di almeno 2^i–1 . Nelle caselle della finger table ci sono, oltre all'ID del nodo, anche i dati necessari per disporre il collegamento diretto con quel nodo.

L'i - esimo dito di n è indicato come n. finger [i] e punta al successore $(n + 2^i–1)$, dove $1 \leq i \leq m$. Si noti che il primo dito è anche il puntatore successore mentre il dito più grande divide lo spazio identificativo circolare in due metà.



Finger Table of Node 8

8 + *1*	14
8 + *2*	14
8 + *4*	14
8 + *8*	21
8 + *16*	32
8 + *32*	48

Figura 1.6: Illustrazione 1: Finger Table

Utilizzando le tabelle finger, Chord individua il successore (k) in O(log N) salti con alta probabilità. Intuitivamente, il processo assomiglia a una ricerca binaria in cui ogni passaggio dimezza la distanza dal successore (k). Ogni nodo n inoltra una richiesta al nodo precedente noto più vicino di k. Questo viene ripetuto fino a quando la richiesta arriva al predecessore (k), il nodo il cui identificatore precede k, che inoltrerà la richiesta al successore

(k). La Figura 1.6 mostra un esempio di ricerca del successore (54) avviata dal nodo 8.

Il nodo 8 inoltra la richiesta al suo sesto dito che punta al nodo 48. Il nodo 48 è il predecessore della chiave 54 perché il suo primo dito punta al nodo 56 e $48 \leq 54 \leq 56$. Pertanto, il nodo 48 inoltrerà la richiesta al nodo 56.

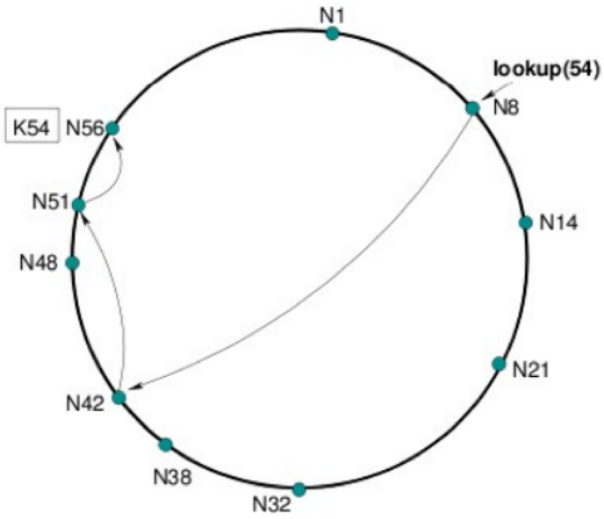


Figura 1.7: Illustrazione 1: Ricerca

In sostanza, la ricerca di una chiave segue un algoritmo distribuito fra alcuni nodi che si trovano sempre vicini alla destinazione; questo algoritmo poi prevede che un nodo che ha il compito di gestire una ricerca di una chiave controlli se è il nodo a cui ne è stata affidata la gestione e in quel caso termina la ricerca, comunicando il risultato della query al nodo che l'ha generata. Nel caso in cui invece il nodo non gestisce la chiave, allora si inoltra la query al nodo noto più vicino alla destinazione.

Dall'integrità dei dati della routing table dipende l'efficienza del routing in Chord; nell'inserimento o nell'uscita di un nodo dalla rete, la routing table ha bisogno di un lasso di tempo per includere le variazioni che vengono apportate dalla modifica dell'insieme dei nodi che fanno parte della rete.
La rete Chord che presenta una routing table senza incoerenze viene detta

stabile. Ogni volta che il routing esegue un hop in una rete Chord stabile, la distanza tra il nodo responsabile della gestione corretta della query e la sua destinazione si riduce. Questa operazione è molto importante per le prestazioni della rete, dal momento che implica che il numero di peer contattati per la gestione di una ricerca cresca in modo logaritmico rispetto al numero di nodi presenti sulla rete.

1.5 Dubbi

≪*Una caratteristica distintiva delle reti P2P strutturate basate su DHT (ad esempio, Tapestry, Pastry, Chord e CAN) è che la ricerca è deterministica: dato l'identificatore di un oggetto (e solo tramite l'identificatore), lo schema di localizzazione e di instradamento sottostante garantisce di trovare l'oggetto a un costo ragionevole. La ricerca per identificatori, tuttavia, è utile solo quando abbiamo piena conoscenza di quali risorse vogliamo, ma spesso abbiamo solo informazioni parziali.*≫[4 pag.1]

Il tipico funzionamento di una DHT che deve immagazzinare e successivamente recuperare un dato potrebbe essere semplificato tramite questo esempio.

Si suppone che una keyspace con un set di stringhe. Un file che presenta i parametri filename e data viene immagazzinato nella DHT dopo aver calcolato la funzione hash(filename), ottenendo così una chiave k.
In seguito viene inviato lungo il network un messaggio put(k, data) al responsabile della chiave interessata a immagazzinare la coppia.
Il file potrà essere richiesto anche da un altro nodo tramite una richiesta get(k), dopo aver calcolato l'hash del filename. Al nodo responsabile verrà inoltrata la richiesta e questo risponderà con una copia del dato immagazzinato. Questa richiesta presenta però una grande limitazione: per poter calcolare l'hash e ricavarne la chiave k, bisogna conoscere a priori il filename. Il problema si ha quando questa informazione viene a mancare e si preferisce una ricerca per parole chiave, ossia una ricerca che permette di ottenere ri-

sultati più pertinenti senza essere in possesso del nome dell'oggetto richiesto. Molti sono i sistemi che hanno cercato o cercano tutt'ora di risolvere il problema del keyword search, alcuni di questi lo hanno fatto attraverso l'utilizzo di una tabella di hash distribuita. Nel corso della tesi si analizzerà un sistema preso come riferimento tra le soluzioni a questo problema, in particolare verrà descritta una struttura a ipercubo, la quale prevede che i nodi vengano rappresentati in un ipercubo a r-dimensioni, con una stringa composta da r-bit. Nei prossimi capitoli dunque verrà approfondita questa tipologia di struttura applicata a una tabella hash distribuita.

Capitolo 2

Keyword Search problem

Nel capitolo precedente abbiamo descritto un parametro fondamentale che si specializza per la ricerca, l'inserimento e l'eliminazione di dati su rete DHT: l'identificatore dell'oggetto.
Abbiamo così anticipato quello che sarà trattato più nel dettaglio in questo capitolo; verranno presentate qui, inoltre, un indice generale di parole chiave e uno schema di ricerca per reti peer to peer strutturate, utili ad evitare i problemi che caratterizzano la ricerca di parole chiave e di attributi negli overlay e vedremo come possono essere eseguiti in modo efficiente le operazioni di inserimento, eliminazione e ricerca di oggetti.

Abbiamo visto che la ricerca nelle reti P2P strutturate basate su DHT (Chord) è caratterizzata da operazioni meccaniche: dato un identificatore di un oggetto, lo schema di localizzazione e di instradamento trova l'oggetto. Come abbiamo potuto osservare però tale ricerca risulta essere utile solo se abbiamo una piena conoscenza delle informazioni dell'oggetto di ricerca, che spesso risultano essere parziali.

2.1 Inverted Index

Il modo più comune per implementare la ricerca per parola chiave nei sistemi informativi è mediante l'indice invertito(inverted index)

≪Un indice invertito è un insieme di voci di coppie (ω, O), dove ω è una parola chiave e O è l'insieme di oggetti che contengono questa parola chiave≫.

Solo dopo aver creato un indice invertito, le parole chiave possono essere inserite e successivamente possono essere trovati tutti gli oggetti che contengono quelle parole chiave. Esempio nella figura.

Key	List of Objects
term1	{Object1, Object2, Object3}
term2	{Object1, Object3}
term3	{Object1, Object2}
term4	{Object3}
term5	{Object2}
...	...

Figura 2.1: Inverted Index

Ogni risorsa, o oggetto, è delineato da un albero valore-attributo (AV-Tree), dove ogni collegamento è la rappresentazione della correlazione tra un attributo e un valore, quest'ultimo rappresentato dal nodo figlio.
Il percorso dalla radice in un albero AV viene chiamato filamento e rappresenta una sequenza di coppie valore-attributo che a sua volta può essere trattata come una parola chiave che descrive la risorsa.
Il concetto di indice invertito può essere quindi usato per cercare le risorse con una data sequenze di coppie attributo-valore. Tuttavia, bisogna notare che alcuni nodi sono responsabili della gestione di risorse che sono già gestite da altri nodi, mentre altri possono gestire un insieme di risorse molto più grandi di altri.

2.2 Problemi con Inverted Index

Questi approcci però presentano alcuni problemi che non sono da sottovalutare.

- Load Balance
  Il primo tra tutti riguarda il fatto che in un corpus di parole, il numero di occorrenze di una data parola all'interno di un oggetto, ossia la frequenza delle parole chiave, è in costante mutamento.
  La distribuzione delle parole segue quindi quella che viene chiamata legge di Zipf (Figura 1), la quale legge mostra che la relazione tra la frequenza con cui compare una parola ed il suo grado segue una legge empirica di proporzionalità inversa; questo significa che alcune parole chiave si verificano molto spesso rispetto ad altre, che invece si verificano più di rado.

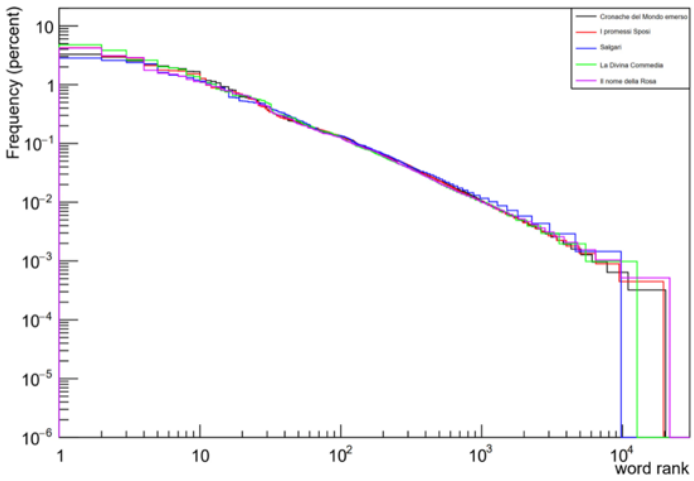


Figura 2.2: Distribuzione di Zipf per alcuni autori o opere in lingua italiana

La conseguenza è che nelle reti DHT, la semplice mappatura delle voci di un indice invertito su un nodo comporterà che la memoria che sarà necessaria per tale scopo risulterà essere completamente disuguale. Ci sarà quindi uno sbilanciamento del carico.

- Ridondanza dello storage
  Un altro problema riguarda, invece, la ridondanza dello storage. Quando creiamo le voci per ogni parola chiave, tutte le informazioni che riguardano l'oggetto σ che contiene le parole chiave si memorizzano ripetutamente in k posizioni diverse; il problema risulta essere ancora più grande se prendiamo in considerazione il fatto che un oggetto spesso può avere nei suoi metadati anche dozzine di parole chiave.
  Così l'eliminazione o l'inserimento di un oggetto vengono rese delle operazioni molto costose, dal momento che devono gestire più accessi peer nella rete.

- Classificazione degli oggetti
  Se considerassimo uno spazio degli oggetto come uno spazio enorme, dovremmo tener conto anche che una query composta da poche parole chiave comuni produrrà un insieme contenente un numero elevato di oggetti.
  Sarebbe dunque necessario una classificazione che aiuterebbe a selezionare gli oggetti interessati. Questo è il terzo problema che riguarda la classificazione degli oggetti, un'operazione che manca nel lavoro descritto sopra.

A questo punto si rende necessario un index scheme atto a risolvere i problemi sopra elencati. Di seguito sarà presentato il modello a ipercubo, dove sarà possibile mappare ogni oggetto in base al set di keyword.

2.3 Ipercubo

Cosa intendiamo dire quando parliamo di ipercubo?
Lo schema di indice distribuito di cui abbiamo parlato sopra è costruito su uno spazio vettoriale di ipercubo r-dimensionale, su una rete DHT e su una struttura logica. L'ipercubo è un grafo il cui insieme di nodi V, è costituito dai vettori booleani 2 "n-dimensionali", cioè vettori con coordinate binarie 0 o 1, dove due nodi sono adiacenti ogni volta che differiscono esattamente in una coordinata[5].

Un ipercubo r-dimensionale $H_r(V, E)$ è costituito da 2^r nodi e ognuno di questi nodi è delineato da una stringa binaria r-bit univoca.
Per V si intende l'insieme di nodi nella rete ed E è l'insieme di collegamenti tra i nodi.
Se, per ogni due nodi u, v in V, u si differenzia da v in un solo bit, allora ci sarà un arco, non orientato, in E. Collegando ogni coppia di nodi con un arco attraverso la (r – 1) esima dimensione, è possibile costruire un ipercubo r-dimensionale da due ipercubi (r- 1) dimensionali.

Una prima definizione:
Definiamo quindi un insieme One(u) di interi per ogni nodo

u $\in V : One(u) = i|u[i] = 1, 0 \leq i \leq r - 1$, le posizioni in cui u ha bit-1.
Successivamente, verrà definito nello stesso modo
$Zero(u) = i|u[i] = 0, 0 \leq i \leq r - 1$.

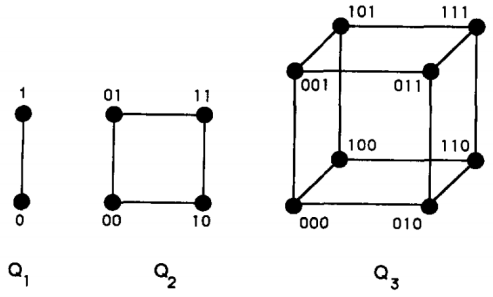


Figura 2.3: Ipercubo per n:1,2,3

2.3.1 Sub-ipercubo

Consideriamo un ipercubo $H_r = (V, E)$ e un nodo $u \in V$. Un sottoipercubo indotto da u, specificato con Hr (u), rappresenta un sottografo G = (U, F) di Hr dove:

- ogni nodo w ∈ V è in U se e solo se w contiene u;
- ogni arco e ∈ E è in F se e solo se i suoi due punti finali sono in U.

Per definizione conosciamo che tutti i nodi w in U, in ogni posizione in One (u), hanno bit-1. Muovendo tali bit, ogni nodo w forma una stringa $(r - |One(u)|)bit$, ossia una stringa $|Zero(u)| - bit$ e sappiamo che ogni due nodi si ha un arco se e solo se differiscono per un bit. Il grafico risultante sarà un ipercubo $|Zero(u)| - dim$. Allora Hr (u) sarà un isomorfo a un ipercubo $|Zero(u)| - dim$[4].

2.3.2 Spanning Binomial Tree

Utilizzando lo Spanning Binomial Tree, la trasmissione negli ipercubi avviene in maniera molto efficiente. Si definisce uno Spanning Binomial Tree (SBT) come una struttura dati ad albero che si basa sul sub-ipercubo $Hr(u)$ descritto sopra, contenente lo stesso numero di nodi e una radice dell'albero u.
In uno Spanning Binomial Tree così definito, tutti i nodi che contengono u e che divergono da esso solo di un bit, sono a livello 1, sono cioè figli di u. In questo modo un nodo si presenterà solo una volta all'interno dell'albero.

Sappiamo che un albero binomiale spanning SBT (u) di Hr ha una profondità r e un nodo al l-esimo livello di SBT (u) ha una distanza di Hamming l da u. La distanza di Hamming tra due stringhe binarie r-bit u e v è Hamming $(u, v) = r1i = 0(u[i]v[i])$.
Ciò significa che un Spanning Binomial Tree contiene nodi che sono a l profondità dalla radice e hanno l bit diversi dalla radice nei loro ID. Questa è una proprietà che risulterà essere utile per la nostra ricerca per parole chiave.

2.4 Mapping dei nodi dell'ipercubo ai nodi DHT

Posizionare l'ipercubo su un DHT può risultare vantaggioso. Il modo più semplice per costruire $H_r(V, E)$ su un DHT fisico è mappare $G : V \to V'$, facendo in modo che ciascun nodo logico abbia un nodo fisico corrispondente nella rete. La trasmissione di messaggi tra due nodi qualsiasi nell'ipercubo costerà $O(logN)$ messaggi nel DHT.
Un ulteriore vantaggio è dimostrato dal fatto che l'ipercubo presenta una dimensione disaccoppiata dalla dimensione del DHT, dove il primo è determinato dall'insieme di oggetti da indicizzare e il secondo dal numero di nodi partecipanti nel sistema.
Se la dimensione dell'ipercubo è maggiore della dimensione del DHT e quindi ci sono più nodi logici che nodi fisici, possiamo usare una funzione hash per bilanciare il carico, in modo da mappare uniformemente i nodi logici (tenendo conto dei loro ID) ai nodi fisici.
Se, invece, la dimensione dell'ipercubo dovesse essere inferiore, allora solo una parte dei nodi fisici sarà responsabile dell'indicizzazione degli oggetti, il ché consente di manovrare con un certo margine la selezione dei nodi da indicizzare. Possiamo così selezionare nodi stabili o potenti necessari come nodi indicizzazione nell'ipercubo.

2.5 Operazioni nella rete

Consideriamo W l'insieme di tutte le parole chiave considerate nel sistema. $h : W \to 0, 1, ..., r - 1$ è la funzione hash uniforme che mappa ogni parola chiave in W su un numero interno in 0, 1, ..., r - 1.

Definiamo invece una mappatura $F_h : 2^w \to V$ in questo modo: $F_h(K) = u$ se, e solo se, $Uno(u) = h(w)|w \in K$.

Questo significa che u è responsabile di K se $F_h(K) = u$ e che per ogni insieme di parole chiave, c'è un nodo univoco nell'ipercubo responsabile

dell'insieme.

L'insieme degli insiemi di parole chiave di cui u è responsabile verrà definito $R_u = K \subseteq W | F_h(K) = u$.

Al fine di costruire lo schema dell'indice, lasciamo che, per ogni oggetto σ associato al set di parole chiave $K\sigma$, il nodo nell'ipercubo mantenga una voce $K\sigma$ nella sua tabella degli indici. Dichiariamo che σ è indicizzato nel nodo e denotiamo con O_u l'insieme di oggetti indicizzati in u

$O_u = \sigma \in O | K\sigma \in R_u$.

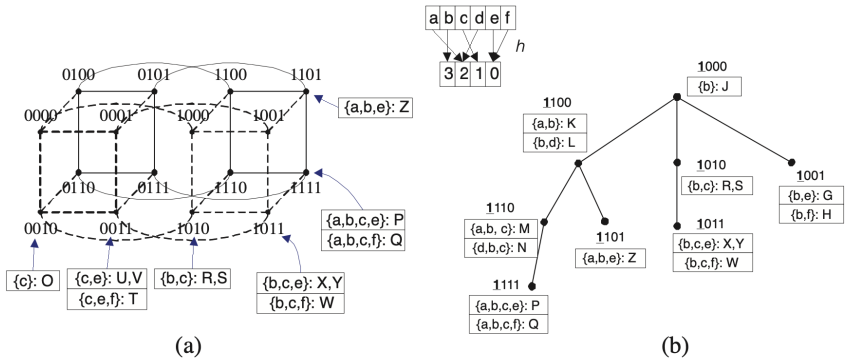


Figura 2.4: schema indice

La figura 2.4 descrive quanto detto quanto detto.

2.5.1 La ricerca

In un insieme di parole chiave K, individuiamo una copia dell'oggetto associato a k rintracciando innanzitutto il nodo responsabile di K in H_r, il quale è determinato da $F_h(K)$.

Solo dopo aver rintracciato tale nodo, possiamo metterci alla ricerca della sua tabella indice in modo da ottenere l'ID di un oggetto σ associato alla sequenza di parole chiave K. Attraverso una chiamata get(σ) alla rete DHT sottostante, che restituirà una copia di σ. Ciò significa che la pin search è possibile direttamente dallo schema.

Per quanto riguarda, invece, la superset search, ciò che è necessario fare è recuperare gli oggetti che possono essere descritti da K. Precedentemente abbiamo visto che un sub-ipercubo $H_r(u)$ di H_r indotto da u contiene tutti i nodi w in V, che a loro volta contengono u. Ciò significa che ogni nodo in H_r responsabile di un superset di K è nel sub-ipercubo indotto da u = F_h(k). Cercando il sub-ipercubo, sarà possibile individuare gli oggetti descritti da K. Durante la ricerca nel sub-ipercubo ci sarà inoltre consentito di esplorare l'albero binomial spanning SBT H_r (u) radicato in u.

Tenendo conto che K_v contiene almeno i più parole chiave di K_u, possiamo dunque individuare, gli oggetti i cui set di parole chiave associati si estendono man mano e questo consente alle applicazioni di livello superiore di individuare in modo più efficiente gli oggetti pertinenti.

Verranno qui descritte nel dettaglio le operazioni necessarie a costruire lo schema dell'indice.

Nell'ipercubo H_r ogni nodo u ha una tabella indice di voci del formato (keywordSet, objectId). Una voce (K, σ) nella tabella significa che nella rete c'è un oggetto σ, il quale è associato all'insieme di parole chiave K. L'insieme di voci (K, σ1,) ..., (K,σ n) che ha lo stesso set di parole chiave può ovviamente essere fusa in una singola voce K, (σ1, ..., σ_n).

2.5.2 Insert

Ciò che è necessario fare per proseguire con l'inserimento di un oggetto nell'overlay è prima di tutto prendere l'oggetto σ in input. Questo sarà formato dal set di chiavi K_σ associate ad esso e il valore data dell'oggetto.

Il salvataggio delle coppie (idObject, data) e (Kσ, idObject) avverrà nella rete, dove la prima coppia verrà immagazzinata dall'algoritmo Chord sottostante, mentre la seconda verrà memorizzata dal nodo responsabile del determinato set di keyword K_σ.

La richiesta di *insert* viene delegata al nodo u, il quale gestisce in input data e la keywordSet K associato. La variabile data rappresenta un indirizzo, ossia una stringa e data la sua semplicità, viene preso come filename questo

valore. Per determinare l'id univoco dell'oggetto σ, viene utilizzata la funzione hash SHA-1. Successivamente viene associata all'hash table objects la coppia (idObject, data), grazie all'aiuto dal nodo u che ha eseguito la richiesta tramite un messaggio di put(id, data). Dal momento che in una DHT, questo messaggio avrebbe dovuto percorre altri nodi prima di raggiungere quello che corrisponde all'id. Qui entra in gioco l'algoritmo Chord, delineato da una index table che contiene un'entry del tipo (idObject, idNodo). Questo permette di risalire al nodo che possiede la copia dell'oggetto.

Quello che conviene fare, in seguito, è creare la referenza che consente di collegare $K_\sigma \rightarrow$ idObject.

Così viene mandato un messaggio Insert($(F_h(K_\sigma))$, K_σ, σId).

2.5.3 Delete

Per eliminare una copia dell'oggetto σ pubblicata in precedenza, il nodo u utilizza stessa procedura usata per l'insert per individuare il nodo responsabile della gestione del riferimento ¡σ, u¿. Quindi richiede l'operazione Delete (L (σ), σ, u), operazione che terminerà quando ci sarà un'altra copia di σ. Dal momento che nella rete non esiste una copia di σ, l'indice dell'oggetto sarà cancellato dall'ipercubo.

2.5.4 Superset search

Questa operazione è composta da un set di parole chiave K e una soglia t e ha come obiettivo quello di restituire un insieme di oggetti i quali possono essere definiti da K.

Anche se ogni nodo del sub-ipercubo può indicizzare alcuni oggetti che possono essere definiti da K, esistono alcune piccole differenze tra gli oggetti. Se un nodo x si trova a una distanza d dalla radice dello Spanning Binomial Tree SBT H_r (F_h (k)), significa che ciascun oggetto indicizzato in x che può essere definito da K è accostato a un insieme di parole chiave contente almeno

d parole chiave in più rispetto a K. Quindi è probabile che sia più specifico rispetto agli oggetti con esattamente la parola chiave impostata K.

Lo Spanning Binomial Tree può essere esplorato ampiamente(breath-first-search), a seconda delle applicazioni, in uno stile ampio, ossia top down, i cui risultati danno la preferenza agli oggetti più generali, o bottom up, che invece preferisce oggetti più specifici. Esplorando l'albero dall'alto verso il basso o dal basso verso l'alto, è possibile recuperare gli oggetti in base alla loro specificità (misurata dal numero di parole chiave aggiuntive).

Comunemente, il modo più usato di implementare la ricerca per parola chiave nei sistemi informativi è attraverso l'utilizzo dell'indice invertito. In questo capitolo è stato presentato un nuovo schema di indice per la ricerca di parole chiave nelle reti P2P strutturate che, differenziandosi dagli approcci fondati sull'uso dell'indice invertito, si è basato invece sull'utilizzo di un ipercubo, con l'obiettivo di indicizzare gli oggetti in base ai loro set di parole chiave.

È stato poi evidenziato come anche le operazioni di insert e delete si muovano autonomamente nella ricerca, indipendentemente dalle dimensioni del set di parole chiave dell'oggetto.

Nel capitolo successivo sarà presentata una simulazione di quanto affermato fin ora.

Capitolo 3

Simulazione

Sarà presentata un'implementazione di quanto detto finora.

Innanzitutto il lavoro è stato quello di partire da una struttura che simulasse l'organizzazione dei nodi di una *Distributed Hash Table*, riprendendo l'algoritmo Chord sviluppato nel primo capitolo.

L'implementazione che viene presentata di seguito, cercherà di dare forma alle teorie presentate nel Capitolo 2, focalizzandosi sull'idea generale presentata: il modello a ipercubo, con il quale vengono ottimizzati i percorsi di routing indipendentemente dall'ID dell'oggetto.

3.1 Simulazione di una DHT

Come anticipato, l'attenzione è stata focalizzata sulla rete DHT, in quanto grazie ai meccanismi efficienti di comunicazione tra i nodi, permette che il sistema funzioni in modo affidabile e corretto.

Una DHT è una sovrapposizione costruita su una rete fisica. La sovrapposizione può essere definita da un grafo orietato G = (V, E) dove V è l'insieme di nodi nella rete ed E è l'insieme di collegamenti tra i nodi.

I nodi che vanno a formare la DHT sono chiamati u e a ciascuno di esso viene assegnata un'identificazione univoco a bit, un iD (n-bit binario).

3.1.1 Nodi

```
//Struttura Nodo e creazione Id binario

public class Node extends Thread implements Comparable<Node> {
    public static final int PORT_BASE = 3000;
Node successor, predecessor;
private int myport = 0;
private String myname;
private int global_rep;
private String idObject;
int ring_size;
    private int r;
    private int n;
    private String id;
    private BitSet bitset;
    private Hashtable<String, Integer> keywordsMap;
    private ArrayList<Node> neighbors;
    private Map<Integer, String> references;
    private Map<String, String> objects;


private String createBinaryID(int n){
     String idString = Integer.toBinaryString(n);
        while (idString.length() < getR()){
        idString = "0" + idString;
    }
    return idString;
}
```

3.1.2 Mapping

Quando non vi sono ambiguità, si usano semplicemente i nodi u per l'identificazione.

La coppia(u, v) in E significa che u conosce un modo diretto per inviare un messaggio a v.

Tuttavia, esistono oggetti o nella DHT che possono rappresentare files o solo parti di files. Un insieme O di oggetti è condiviso nella rete. Ogni oggetto $o \in$ O ha anch'esso un ID univoco e ha un insieme di repliche sparse nella rete.

Questo significa che ogni oggetto o può essere replicato in più nodi u, e che quindi per tenere traccia delle repliche, a ciascun nodo u che memorizza una copia di σ, usiamo (σ, u) per denotare un riferimento alla replica.

Questo avviene tramite un Hash Map referenziata dall'ipercubo.

```
private Map<String, String> mapping = new HashMap<>()


String idObject = getMd5(oValue);
public void addMapping (String idObject, String idNode){
    this.mapping.put(idObject, idNode);
}
```

Il riferimento (σ, u) consiste tipicamente nell'IP / porta di u(idNode) e dall'indirizzo fisico di σ all'interno del nodo. È necessario ottenere un riferimento a σ per accedere a una copia di σ. Quindi i riferimenti fungono da indice per gli oggetti e individuare un oggetto equivale a individuare un riferimento all'oggetto.

I nodi v della DHT sono i nodi che mantengono le referenze (o,u), ma che non necessariamente mantengono anche oggetti.

Nella DHT, un riferimento di un oggetto non è necessariamente memorizzato nello stesso nodo che memorizza la copia fisica dell'oggetto. Questo implica che alcuni nodi u possono mantenere referenze per altri nodi e quindi essere considerati come nodi di tipo v.

La mappatura L, mappa in modo deterministico e uniforme ogni oggetto $\sigma \in$ O (tramite il suo ID) esattamente su un nodo, rappresentato come una stringa binaria a bit, per gestire l'oggetto. Il meccanismo di instradamento determina, per ogni due nodi u e v in V, almeno un percorso (u, v) in G.

3.1.3 Inserimento

Per inserire un oggetto nell'overlay della DHT la prima operazione da svolgere è quella di definire in input l'oggetto σ, formato da un set di chiavi K_σ collegate a tale oggetto insieme al suo valore data. Questi ultimi verranno poi salvete nella rete definendo le rispettive coppie idObject, data e K_σ, idObject. Le coppie così definite prenderanno strade diverse nell'operazione di immagazzinamento.

La prima infatti verrà partizionata dall'algoritmo della DHT, la seconda invece verrà memorizzata da K_σ, ossia dal nodo responsabile del set di keyword associato all'oggetto.

Il nodo prenderà in gestione la richiesta di Insert prendendo prima in input data e la keywordSet K associata. La variabile data definisce un indirizzo, ossia una stringa e quindi sarà questo valore ad essere considerato come filename. La funzione hash SHA-1 successivamente sottopone la stringa definita dalla variabile data per determinare l'id univoco dell'oggetto σ. Quando otteniamo l'ID, la coppia ¡idObject, data¿ sarà unito all'hash table objects, che a sua volta è determinata dal nodo u che ha effettuato la richiesta, attraverso un messaggio put(id,data). Questa operazione rappresenta un'operazione importante che viene eseguita su una DHT, l'operazione di inserimento di una coppia chiave-valore.

Un esempio di inserimento di una coppia chiave-valore nella DHT è presentato qui.

```
insert,something,502
Main says: I forwarded the command to Node with ID: 23
Node 23: Got message: insert,something,502-1-localhost-3009
Node 23: Replica_counter is now 1
id oggetto41
Node 23: I forwarded the query to localhost:3007
Node 26: Got message: insert,something,502-1-localhost-3009
Node 26: Replica_counter is now 1
id oggetto41
Node 26: I forwarded the query to localhost:3008
Node 31: Got message: insert,something,502-1-localhost-3009
Node 31: Replica_counter is now 1
id oggetto41
Node 31: I forwarded the query to localhost:3010
Node 37: Got message: insert,something,502-1-localhost-3009
Node 37: Replica_counter is now 1
id oggetto41
Node 37: I forwarded the query to localhost:3001
Node 43: Got message: insert,something,502-1-localhost-3009
Node 43: Replica_counter is now 1
id oggetto41
Node 43: I am responsible for :something
Node 23: Got message: ANSWER-node 43 Inserted pair (something,502)
Node 23: node 43 Inserted pair (something,502)
Node 43: I am the last thread inserting and going to sleep
```

Figura 3.1: Output dopo una richiesta di inserimento nella rete DHT

Come possiamo vedere viene eseguito un routing del messaggio in tutti i nodi, fino ad arrivare al nodo che se ne occuperà(Node 43). L'esempio mostrato non rispecchia a pieno la situazione analizzata nel corso della tesi, poichè tratta i nodi di una DHT senza l'uso dell'ipercubo.

L'operazione successiva consiste nel realizzare la referenza responsabile del collegamento $K_\sigma \rightarrow idObject$, inviando un messaggio Insert ((Fh(K_σ)), K_σ, σId). Questo passa di nodo in nodo attraverso l'algoritmo di routing, fino a raggiungere il responsabile Fh(K_σ). Qui verrà aggiunta la referenza nella sua tabella references.

```
Aggiunge alla lista di reference la coppia <$K_sigma, sigma>

private void Insert(Node n, Set<String> oKey, String idObject){
```

```
responsible.addReference(oKey, idObject);
}
```

3.1.4 Ricerca

Per fornire un servizio di ricerca per parole chiave, è necessario progettare uno schema di indice distribuito in modo che un oggetto possa essere individuato specificando alcune parole chiave in una query. Due funzioni possono essere identificate utili a questo tipo di servizio:

Ricerca pin: dato l'insieme di parole chiave K, il servizio dovrebbe restituire l'insieme $\sigma|K_\sigma$ = K di oggetti che sono associati esattamente con l'insieme di parole chiave K. In questo modo la ricerca prevede l'inserimento in input del set di chiavi K = (k1, ..., k_n) dall'utente che ha l'obiettivo di recuperare gli oggetti caratterizzanti il set di chiavi.

Quest'ultimo, successivamente, viene memorizzato in un HashSet String, che a sua volta viene convertito nel corrispondente BitSet. Il procedimento in questione avviene attraverso la funziona che abbiamo visto precedentemente e che è stata nominata Fh(k). L'utente viene collegato al nodo u per eseguire la richiesta e da qui verrà inviato un messaggio getReference(F_h(K),K); attraverso l'algoritmo di routing, il messaggio verrà trasmesso al nodo responsabile del set di keyword che, ricevuto il messaggio, restituirà tutte le references degli oggetti da lui gestiti: ArrayList idObject.

```
private static void pinSearch(Hypercube hypercube, Node n, int r){
 Set<String> kStringSet = new HashSet<String>();
String targetNodeId;
ArrayList<String> idObjects;

kStringSet = insertKeywords();


System.out.print("Il set di keyword e': " + kStringSet);

//visualizzo a schermo il nodo che si occupa della keyword
targetNodeId = getStringSearched(n.generateBitSet(kStringSet), r);
System.out.println("\nNodo che si occupa della keyword: " + targetNodeId);
System.out.println("Cerco il nodo: " + targetNodeId);

try {
idObjects = new ArrayList<String>(n.requestObjects(kStringSet));
System.out.println(n.getObjects(hypercube, idObjects));
} catch (NullPointerException e) {
System.out.println("Nessun oggetto trovato");
}
}
```

Una volta ottenuto l'id dell'oggetto entra in gioco l'algoritmo della DHT richiamando la funzione che preleva l'oggetto passando l'idObject.

Anche se la pin search è molto efficiente per localizzare un oggetto particolare, spesso si preferiscono i risultati che si basano sul superset di parole chiave che si presentano in un set di query K, cioè:
ricerca supeset: Dato l'insieme di parole chiave K e una certa soglia t, il servizio dovrebbe restituire un insieme di oggetti min (t, — OK —) che possono essere descritti da K. La ricerca superset estende quella della Pin Search anche ad altri nodi, ossia i nodi che rappresentano gli altri set di keyword.

In questo caso si prende in input della richiesta, non solo il set di keyword K, ma viene definita anche una variabile c(nel codice la variabile c viene passata di default a 100 nella richiesta requestObjects(kStringSet 100)), la quale determina il valore minimo dei risultati in risposta. Ciò significa che il sistema risponderà con almeno c valori descritti dal set K. Dal nodo Fh(K) parte la ricerca e da qui però si estende a tutto il sub-ipercubo indotto dal nodo stesso.

Nell'albero SBT(u), istanziato per ricercare nel sub-ipercubo, troviamo che u è proprio il nodo Fh(k).

Per fare cio', la ricerca parte sempre dal nodo Fh(K), ma da qui si estende a tutto il sub-ipercubo indotto da Fh(K). Per ricercare nel sub-ipercubo, 'e istanziato l'albero SBT(u), dove u 'e proprio il nodo Fh(K). Bisogna ricordare che un sub-ipercubo Hr (u) di Hr indotto da u è formato da tutti i nodi w in V che contengono u (cioè u [i] $\rightarrow$ w [i]).

Ciò significa che ogni nodo del sub-ipercubo è responsabile di un set di parole chiave che è un super set di K, e ciascun nodo in H_r che è responsabile di un super set di K e anche nel sub-ipercubo. Questa caratteristica ci consente di cercare solo il sub-ipercubo se siamo interessati a cercare qualsiasi altro oggetto che può essere descritto anche da K.

```
private static void search(Hypercube hypercube, Node n, int r) {

Set<String> kStringSet = new HashSet<String>();

String targetNodeId;
ArrayList<String> idObjects;

System.out.print("Il set di keyword e': " + kStringSet);

targetNodeId = getStringSearched(n.generateBitSet(kStringSet), r);

System.out.println("Cerco il nodo: " + targetNodeId);
```

```
try {
idObjects = new ArrayList<String>(n.requestObjects(kStringSet, 100));

System.out.println(n.getObjects(hypercube, idObjects));
} catch (NullPointerException e) {

System.out.println("Nessun oggetto trovato");
}
}
```

Capitolo 4

Conslusioni

Per sintetizzare in generale i vari contenuti, nel primo capitolo, descrivendo le caratteristiche che costituiscono una rete DHT, è stato analizzato Chord, un protocollo di ricerca peer-to-peer.

In particolare, ciò che interessa a questo lavoro di tesi riguardano i meccanismi che vengono utilizzati nella decisione di memorizzazione dell'informazione e dello strumento da utilizzare per recuperare tale informazione.

Nel secondo capitolo viene trattato un po' nel dettaglio il parametro fondamentale alla ricerca su rete DHT, ossia l'identificatore dell'oggetto.

Ciò che interessa di più a fini del lavoro di tesi è il modo per implementare la ricerca per parola chiave e la differenza tra l'uso dell'indice invertito (inverted indecx), il quale presenta alcuni problemi che sono stati descritti, e uno schema di indice basato sull'utilizzo dell'ipercubo per indicizzare gli oggetti a seconda del loro set di parole chiave.

Possiamo concludere, attraverso questo lavoro, che abbiamo potuto costatare come, nella ricerca di chiavi in DHT, un oggetto possa essere localizzato in modo efficiente e deterministico se, all'operazione di tale ricerca, venga fornita la sua serie di parole chiave. In particolare, possiamo dire che quando si cercano oggetti descritti da un set di parole chiave, il nostro schema dell'indice ipercubo è tale che diventa più efficiente quando vengono fornite più parole chiave.

L'indice invertito, al contrario, è più efficiente quando i set di parole chiave interrogati sono piccoli, in quanto le query di piccole serie di parole chiave sono inevitabili e influisce di tanto la precisione della ricerca.

Un altro aspetto importante è che l'indice ipercubo facilita l'operazione di posizionamento e di espansione delle query. Esplorare gli oggetti corrispondenti, infatti, nella struttura dell'indice ipercubo, equivale ad attraversare l'albero binomiale spinning radicato nel nodo responsabile del set di parole chiave. misurando quindi il numero di parole chiave aggiuntive, è possibile recuperare gli oggetti in base allo loro specificità, esplorando l'albero dall'alto verso il basso o dal basso verso l'alto.

Bibliografia

[1] I. Stoica et al. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, 2003.

[2] Wikipedia. Chord (protocollo peer-to-peer) — wikipedia, l'enciclopedia libera, 2019.

[3] Arturo Crespo and Hector Garcia-Molina. *Routing indices for peer-to-peer sys- tems.*, 2001. Technical Report 2001-48, Stanford University, CS Department.

[4] Li-Wey Yang e Chien-Tse Fang Yuh-Jzer Joung. Keyword search in dht-based peer-to-peer networks, 2007.

[5] John P. Hayes e Horng-Jyh Wu Frank Harary. A survey of the theroy of hypercube graph, 1988.

[6] I.-J Wang et al. Supporting content-based music retrieval in structured peer-to-peer overlays, 2016.

[7] Antony Rowstron e Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, 2001.

[8] Artur Olszak. Hycube: A distributed hash table based on a variable metric, 2016.