Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Ricerca su Registri Distribuiti: un Approccio Basato su Distributed Hash Tables

Relatore:
Chiar.mo Prof.
Stefano Ferretti

Presentata da:
Cesare Giansante

Correlatore:
Chiar.mo Dott.
Mirko Zichichi

III Sessione
Anno Accademico 2019/2020

Sommario

In questo lavoro di tesi viene proposta e realizzata una Distributed Hash Table (DHT). Una Distributed Hash Table è un sistema di archiviazione decentralizzato che fornisce schemi di ricerca e archiviazione simili a una tabella hash, permettendo la memorizzazione di dati sotto forma di coppie chiave-valore.
Gestisce i dati distribuendoli su un numero di nodi e implementando uno schema di routing che consente di cercare in modo efficiente il nodo su cui si trova l'elemento della ricerca. La caratteristica della DHT realizzata è quella di avere una struttura ad ipercubo. È stato utilizzato un simulatore, PeerSim, con il quale è stato possibile simulare la rete di nodi su cui si appoggia la DHT. Lo scopo di questo strumento è quello di permettere a chiunque lo utilizzi, di cercare in maniera facile e veloce dei dati specifici. Questo è permesso grazie all'implementazione del meccanismo di ricerca basato su keywords.
Il lavoro proposto fa parte di una ricerca più ampia, la quale, mira a realizzare un'architettura di sistema per promuovere lo sviluppo di sistemi di trasporto intelligenti (ITS) utilizzando registri distribuiti e tecnologie correlate.
Saranno implementate due tipologie di ricerca e, infine, verranno eseguiti dei test per valutare l'efficienza di queste operazioni.
Lo scopo è quello di constatare quanto sia efficiente il meccanismo di routing implementato, andando ad analizzare il rapporto tra il numero dei nodi della rete e il numero degli scambi di informazione tra i vari nodi necessario per completare una richiesta di ricerca.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

I dati sono ormai diventati una risorsa fondamentale per molte aziende e per molte realtà industriali. La possibilità e la capacità di poter analizzare e migliorare il proprio servizio per renderlo più idoneo e più vicino al consumatore finale, spinge sempre di più all'acquisizione e all'adozione di grandi quantità di informazioni. Risulta, quindi, sempre più importante la capacità di saper entrare in possesso di dati specifici e in quantità necessarie su cui poi effettuare delle analisi. Il riuscire ad acquisire tutto il materiale necessario per le proprie analisi si scontra con una realtà in cui i dati non sono reperibili facilmente, poiché quest'ultimi si trovano sotto varie forme e non si ha la garanzia che ciò che si sta acquisendo sia vero. Diventa cosi un problema fondamentale il riuscire ad entrare in possesso di quantità di dati che siano facilmente reperibili ma soprattutto che non siano falsi.
Per questo motivo, strumenti come le Distributed Ledger Technology (DLT) e le Distributed Hash Table (DHT) riscontrano un notevole interesse poiché consentono l'archiviazione e la gestione di dati in maniera distribuita, permanente e di facile accesso. I registri distribuiti utilizzano algoritmi di consenso che garantiscono alti livelli di sicurezza, trasparenza e immutabilità, e utilizzano protocolli di tipo peer-to-peer in cui non esiste il governo di nessuna autorità centrale. Le DHT consentono di entrare in possesso di determinati tipi di dati potendo effettuare delle ricerche mirate all'obiettivo. Entrambe queste tecnologie sono caratterizzate dal fatto di essere decentralizzate poiché non esiste un server centrale che gestisce le informazioni ma ognuna di esse è distribuita sui vari client che compongono la rete. La decentralizzazione è la nuova tendenza che sta caratterizzando e che sarà protagonista negli anni a venire. Applicazioni e sistemi vengono realizzati seguendo questo modello perché la possibilità di avere risorse non confinate

in un unico sistema ma distribuite su più dispositivi permette di avere molti benefici e rende ognuno protagonista dei propri dati. Il lavoro proposto mira alla realizzazione di una DHT che permetta di effettuare operazioni di ricerca in maniera semplificata. È possibile cercare dati specifici inserendo delle keywords che sono associate al dato richiesto. Verranno implementate due tipologie di ricerche e su queste si andrà a svolgere una fase di valutazione. Lo scopo è quello di capire quanti scambi di informazioni tra i vari nodi della rete sono necessari per completare una richiesta di ricerca e quindi constatare quanto sia efficiente il meccanismo di routing del sistema. Nello specifico, i test mirano a capire la correlazione tra il numero dei nodi presenti e il numero degli scambi necessario tra quest'ultimi per rispondere ad una ricerca, rapportando questi valori anche con la quantità di oggetti archiviati nella DHT. Hypeercube è la DHT che è stata realizzata e che rientra in un'architettura di sistema più ampia [1] nella quale i dati una volta che vengono rilevati e raccolti sono archiviati in un registro distribuito. In seguito, la DHT in questione può essere interrogata per la ricerca di informazioni specifiche in quanto lo strumento si collega al registro e ai dati contenuti in esso. Per la ricerca si utilizza un sistema basato su keywords. La struttura del sistema ha origine da questo lavoro [2] [3]. La capacità di poter entrare in possesso facilmente di dati permette la creazione di un mercato nella quale sempre più utenti sono consumatori e venditori allo stesso tempo. I grandi competitor sono sempre alla ricerca di modi per acquisire ed entrare in possesso di grandi porzioni di dati con cui ricavare informazioni importanti per il proprio business.

Il lavoro di tesi ha una struttura che segue questa logica: Il primo capitolo parla dello stato dell'arte, cioè una panoramica dei concetti e delle tecnologie in cui si inquadra il lavoro effettuato. Il secondo capitolo approfondisce l'architettura di sistema nella quale la DHT in questione rientra come uno dei componenti fondamentali. Il terzo capitolo spiega le fasi di progettazione e implementazione di Hypeercube. Il quarto capitolo si concentra sulla valutazione dei risultati sperimentali ottenuti e, infine, un quinto e ultimo capitolo che trae delle considerazioni sul lavoro svolto.

Capitolo 1

Stato dell'Arte

Anno dopo anno i dati acquisiscono sempre più importanza, sono paragonati all'oro per il loro valore. Il business dei dati non conosce limiti, che essi siano trattati in maniera legale o illegale apparentemente risulta avere poca importanza. Ormai, la caccia ai dati coinvolge tutti i settori economici e chi opera in questi settori fa di tutto per riuscire ad acquisirne più quantità possibili.
Dai dati acquisiti è possibile ricavare informazioni che possono risultare vitali al proseguo della propria attività finanziaria. Capire, conoscere ma soprattutto anticipare le varie tendenze e le varie preferenze degli utenti diventa fondamentale per una realtà che vuole consolidare la propria posizione nel mercato. Migliorare la qualità del proprio servizio che viene offerto al consumatore finale porta quest'ultimo a beneficiare delle novità apportate e, di conseguenza, ad essere disponibile anche a pagare di più per un prodotto che risulti migliore.
Ottenere grandi quantità di dati su cui poter lavorare non è facile. Questo perché oltre dover riuscire a raccoglierne molti e significativi per il proprio scopo, potrebbe risultare complicato ottenere dati veritieri e non di origine dubbia o peggio, falsa. Basare le proprie analisi su dati non veritieri e di conseguenza basare le proprie azioni sulle informazioni ricavate, comporterebbe un rischio troppo elevato per la propria attività.
Ne consegue che riuscire a tracciare e quindi essere sicuri dell'origine dei dati su cui poi si andrà a lavorare diventa una garanzia in più, una sorta di marchio di qualità. Il lavoro che è stato elaborato in questa tesi è un tassello che rientra in una ricerca più ampia [1].

L'idea è quella di realizzare un'architettura di sistema per promuovere lo sviluppo di sistemi di trasporto intelligenti (ITS) utilizzando registri distribuiti e tecnologie correlate. Grazie a questi sistemi diventa possibile creare, archiviare e condividere i dati generati dagli utenti attraverso i sensori sui propri dispositivi o veicoli mentre sono in movimento. Si propone un'architettura basata su Distributed Ledger Technologies (DLTs) per offrire funzionalità come l'immutabilità, la tracciabilità e la verificabilità dei dati. IOTA, una promettente DLT per l'IoT, viene utilizzata insieme agli archivi di file decentralizzati (DFS) per archiviare e certificare i dati (e i relativi metadati) provenienti dai veicoli o dai dispositivi degli utenti stessi (smartphone). La blockchain di Ethereum viene sfruttata come piattaforma di smart contract che coordina la condivisione dei dati attraverso meccanismi di controllo degli accessi. Le garanzie sulla privacy sono fornite dall'utilizzo di sistemi di gestione delle chiavi distribuite e Zero Knowledge Proof. Il lavoro sviluppato consiste nella progettazione e implementazione di una Distributed Hash Table (DHT), in cui i dati una volta che sono stati acquisiti e registrati in dei registri distribuiti, vengono caricati nella DHT. Questo strumento facilita la ricerca in quanto utilizza un sistema basato su keywords [2] [3]. In questo modo si semplifica notevolmente il processo di ricerca dei dati aiutando chiunque li richiede.
Prima di andare a trattare nello specifico il progetto, verrà effettuata una panoramica generale sulle DHT, più in generale sui sistemi distribuiti e le reti peer-to-peer, andando a descriverne le caratteristiche e le proprietà.

1.1 Reti Peer-to-Peer

Una rete Peer-to-Peer [5] indica un modello di architettura logica per applicazioni distribuite. In un sistema P2P le entità partecipanti condividono le proprie risorse per contribuire attivamente alla fornitura del servizio in questa tipologia di sistema. Queste entità sono identificate nei nodi, i quali non sono strutturati in una forma gerarchica quale client o server ma, sotto forma di nodi equivalenti o paritari (peer), fungendo al contempo sia da client che da server verso gli altri nodi terminali (host) della rete.

Infatti, il modello P2P si contrappone proprio alla tradizionale architettura client/server. Ogni nodo agisce sia come client che come server e in questo sistema ognuno ottiene ed offre risorse dalla comunità. Tutti i nodi hanno la stessa importanza in linea di principio e non esiste nessun controllo centralizzato. In realtà, possono essere presenti alcuni nodi con funzionalità diverse rispetto ad altri, i cosiddetti supernodi.
Il numero di peer che compone questo tipo di infrastruttura può essere dell'ordine delle centinaia di migliaia e possono essere altamente dinamici ed autonomi. Un nodo può entrare o uscire dalla rete P2P in ogni momento. Nessun peer ha una visione globale del sistema. I comportamenti dei vari nodi che coinvolgono anche molti altri nodi della rete possono essere intesi come dei comportamenti globali composti da interazioni locali di ognuno.
Infatti, ogni peer si relaziona con un numero limitato di altri componenti, questo numero è rappresentato dal numero dei nodi vicini al nodo stesso. Il numero dei vicini è dato in base a come la rete viene strutturata.

Figura 1.1: Connessioni peer-to-peer nella rete. [18]

1.1.1 Routing delle informazioni

Nei sistemi P2P i vari utenti accedono alle risorse in seguito ad una fase di ricerca. I peer non utilizzano il DNS ma sono caratterizzati da una connettività tipicamente non permanente in quanto la rete può risultare molto dinamica. Non essendoci un server centrale in grado di far connettere tra di loro i diversi nodi, viene solitamente integrata una overlay network virtuale (ovvero una "rete sovrapposta"), con la quale i nodi formano una sottorete rispetto alla rete fisica principale.
L'overlay è utilizzato principalmente per indicizzare e mappare i nodi. A seconda di come i nodi sono collegati tra loro e dalla loro distribuzione, è possibile dividere le reti P2P in "strutturate" e "non strutturate".

Una rete strutturata è caratterizzata da una topologia specifica, la quale assicura che ogni richiesta può essere soddisfatta, sempre che la risorsa sia presente nella rete. Per facilitare questo compito, ogni rete strutturata può sfruttare varie tecniche per raggiungere questo scopo. Una Distributed Hash Table (DHT) permette di organizzare secondo uno schema ben preciso i vari nodi e instaura dei canali di scambio delle informazioni tra i nodi tra loro vicini per facilitare la connessione della rete.

Invece, una rete non strutturata è caratterizzata da un'apparente disorganizzazione ed è formata da nodi che creano collegamenti casuali con altri della rete. Si tratta, dunque, di reti di facile "formazione", che non richiedono il rispetto di parametri particolarmente stringenti.
Allo stesso tempo, però, la mancanza di una struttura e un'organizzazione interna, rende particolarmente complesso e lungo il compito di cercare file o risorse. La richiesta, infatti, dovrà essere inviata a tutti i nodi che condividono il file. Ciò, ovviamente, genera un gran volume di traffico, senza la certezza di riuscire a individuare la risorse cercata.

1.1.1.1 Architetture P2P

Tre sono le tipologie di architetture P2P:

- **Decentralizzazione pura**: Il sistema è totalmente decentralizzato, nessun coordinatore centrale. Ogni peer può assumere un ruolo sia di client che di server.

- **Parzialmente centralizzato**: In questo modello sono presenti dei supernodi che fungono da server centrali. I nodi quando necessitano di un determinato contenuto, prima contattano individualmente uno di questi supernodi e poi inoltrano la richiesta.

- **Decentralizzazione ibrida**: Presenza di un server centralizzato che facilita l'interazione tra i peer.

1.1.2 Routing in reti non strutturate

1.1.2.1 Sistemi con decentralizzazione ibrida: directory centralizzata

Per directory centralizzata si intende un insieme di servizi che provvedono ad organizzare, gestire e memorizzare informazioni e risorse condivise e scambiate all'interno di reti formate da molti utenti. Queste informazioni vengono rese disponibili agli utenti, sui quali viene messo in atto un controllo degli accessi delle risorse, utile al lavoro dell'amministratore di sistema. I servizi di directory forniscono dunque uno strato di astrazione intermedio tra le risorse da una parte e gli utenti dall'altra.
Si parla direttamente di directory riferendosi alla struttura di dati ordinata nella quale viene immagazzinata l'informazione. Una directory viene considerata, per analogia, un database organizzato, da un punto di vista logico, secondo una modalità gerarchica. Si tratta di un particolare tipo di database specializzato che ha caratteristiche differenti dai tradizionali database relazionali.
Una directory, per sua stessa natura, riceve quasi esclusivamente accessi in lettura in quanto la fase di scrittura è limitata ai proprietari delle singole informazioni o agli amministratori di sistema. Per questo motivo, le directory sono ottimizzate per la lettura, mentre i tradizionali database relazionali devono supportare sia la lettura sia la scrittura dei dati.
L'accesso alle directory di solito utilizza il modello di comunicazione client/server. La directory possiede il mapping delle risorse con i vari peer, fornendo un servizio di discovery dei peer e di lookup delle risorse. I limiti di questo modello di decentralizzazione

ibrida si ritrovano innanzitutto sui costi di gestione di una directory centralizzata e poi anche sul fatto che rappresenta un single point of failure e quindi un rischio in più per la gestione delle risorse. Senza contare il fatto che avendo un nodo centrale in cui i vari utenti richiedono informazioni, elevate richieste potrebbero creare dei colli di bottiglia e quindi limitare il servizio.

1.1.2.2 Sistemi decentralizzati puri: ricerca flood-based

Il flooding è un protocollo di instradamento che permette di inoltrare un pacchetto in ingresso su tutte le linee ad eccezione di quella da cui proviene. Ogni peer propaga la richiesta ai peer vicini, che a loro volta inviano la richiesta ai loro vicini, sempre escludendo il nodo da cui hanno ricevuto la richiesta.
Questo procedimento genera un vasto numero di pacchetti duplicati, a meno che non si applicano determinate misure per limitare il processo. Infatti, essendoci una crescita esponenziale del numero di messaggi, c'è più probabilità di congestione e anche di attacchi di tipo DOS. Per evitare l'invio infinito di pacchetti duplicati si possono utilizzare due metodi:

- **contatore di salto**: si inserisce nel pacchetto un contatore da decrementare ad ogni nuovo router attraversato. Idealmente il valore di tale contatore deve essere uguale al percorso minimo fra sorgente e destinazione ma non conoscendo la topologia della rete si può assegnare un valore uguale al diametro della rete.

- **numero di sequenza**: ogni router deve conoscere la presenza degli altri router e per ogni router dovrà solo controllare che il pacchetto proveniente da quello abbia un numero sequenza maggiore del precedente. Per evitare la crescita all'infinito si adotta una soglia k che riassume la ricezione di tutte le sequenze fino ad appunto k. Raggiunta la soglia il numero si azzera.

1.1.2.3 Sistemi parzialmente centralizzati

Nei sistemi parzialmente centralizzati non tutti i peer sono uguali. I nodi meglio connessi e con buona capacità computazionale possono avere funzioni speciali. Si identificano quindi come supernodi o broker ed ognuno gestisce un insieme di nodi a cui il supernodo fa capo. Si ha un'organizzazione gerarchica nella quale i supernodi agiscono da rappresentanti dei loro sottoposti. Hanno funzione di directory semicentralizzata e indicizzano le risorse disponibili nei peer che gestiscono.

Il flooding delle risorse nella rete riguarda solo i supernodi, i quali poi indirizzeranno le risorse nei nodi specifici che gestiscono. Così facendo si sfrutta l'eterogeneità dei nodi presenti in una rete P2P. Rispetto ai sistemi decentralizzati puri si riduce il tempo di ricerca.

1.1.3 Routing in reti strutturate

1.1.3.1 Sistemi con Distributed Hash Table (DHT)

In un sistema come una DHT, ogni peer della rete è identificato tramite un ID ed è collegato ad un certo numero di nodi. Quindi ogni nodo ha dei vicini con i quali scambiarsi le risorse che vengono instradate. Le risorse sono rappresentate da una coppia chiave-valore (K,V) in cui K è una parola chiave che identifica la risorsa e V consiste nel risultato della funzione hash applicata alla risorsa. Ogni risorsa è identificata solo mediante il valore della chiave.

Il messaggio viene instradato e verrà ricevuto dal nodo responsabile di memorizzare quei dati. I nodi destinatari delle risorse sono scelti in base a come viene partizionata la rete. Le DHT offrono elevata scalabilità e rappresentano un'astrazione distribuita della struttura dati hash table. Le proprietà e le caratteristiche relative a questa tipologia di struttura vengono approfondite in un paragrafo successivo.

Quali sono le caratteristiche e proprietà di un sistema distribuito nel quale operano l'insieme dei nodi?

1.1.3.2 Caratteristiche e proprietà di un sistema distribuito

Un sistema distribuito [6] è un sistema costituito da un insieme di processi interconnessi tra loro in cui le comunicazioni tra i vari componenti, i nodi, avvengono esclusivamente tramite lo scambio di specifici messaggi.
Ogni nodo del sistema esegue un insieme di componenti che comunicano tra di loro utilizzando uno strato software detto middleware che permette all'utente di percepire il sistema come un'unica entità.
In questo sistema sono specifici degli algoritmi che permettono ai vari nodi di poter scambiare messaggi o in generale effettuare delle operazioni che ipoteticamente possono coinvolgere l'intera rete.
Quali sono i vantaggi ma anche gli svantaggi legati ad un sistema distribuito?

Vantaggi

- **Affidabilità e tolleranza ai guasti**: Data la presenza ridondante di macchine all'interno della rete, questo tipo di sistema previene l'integrità e la presenza delle informazioni all'interno del sistema anche se una delle macchine non risultasse più funzionante. Il malfunzionamento di una macchina comporterà solo un rallentamento e non andrebbe a colpire l'intero sistema.

- **Trasparenza**: Lavorando con questo sistema non ci si accorgerà di lavorare con più macchine, bensì con una sola.

- **Integrazione**: Questo tipo di sistemi sono in grado di mettere in comunicazione tra di loro svariate macchine estremamente diverse tra di loro anche come tipologia, purché siano in grado di interfacciarsi allo stesso sottosistema di comunicazione.

- **Economicità**: I costi di un sistema distribuito risultano più bassi rispetto a quelli di un sistema centralizzato.

- **Apertura**: Grazie alla definizione di protocolli standard saranno possibili processi di interoperabilità, cioè, la possibilità di implementazione su hardware diversi. In generale si ha la possibilità di ampliare sia hardware che software.

- **Scalabilità**: La capacità di erogare le medesime prestazioni, in termini di throughput e latenza, rispetto agli utilizzatori nonostante l'aumento del carico operativo sul sistema. Per aumento carico si possono intendere picchi di carico di accesso alle risorse dovuto all'aumento degli utenti del sistema in conseguenza all'evoluzione nel contesto.

- **Autonomia**: Un sistema distribuito non ha un singolo punto dal quale può essere controllato, coordinato e gestito. La collaborazione va ottenuta inviando messaggi tra le varie componenti del sistema e gestita tramite politiche di condivisione e di accesso che devono essere rigorosamente seguite.

- **Evoluzione**: Un sistema distribuito può cambiare sostanzialmente durante la sua vita, sia perché cambia l'ambiente sia perché cambia la tecnologia utilizzata. L'obiettivo è quello di assecondare questi cambiamenti senza costi eccessivi.

Svantaggi

- **Complessità**: Un sistema distribuito è fisicamente più complesso da gestire rispetto ad uno centralizzato e richiede complicate tecniche di comunicazione.

- **Sicurezza**: Essendo per definizione distribuito su più host, c'è bisogno di adoperare degli standard e delle tecniche che permettono la sicurezza in rete al fine di tutelare tutti coloro che usufruiscono del sistema.

- **Non prevedibilità**: I tempi di risposta dipendono dal carico del sistema e dal carico della rete, i quali, possono cambiare anche rapidamente.

- **Gestibilità**: Risulta necessario uno sforzo maggiore per la gestione dell'ambiente di esecuzione e delle applicazioni.

Figura 1.2: La nuova via della decentralizzazione. [19]

1.1.3.3 Sicurezza e privacy

Oltre agli attacchi che una rete aperta può subire, anche i computer che ad essa accedono possono essere soggetti a problematiche di sicurezza e privacy. Per la stessa filosofia del P2P quasi tutti i programmi di file-sharing richiedono di avere sul proprio computer dei file condivisi e che quindi possano essere a disposizione degli utenti che ne fanno richiesta. Questo implica da un lato la condivisione di un'area del disco sulla quale mettere i file a disposizione, dall'altro consentire il libero accesso ad alcune porte del computer.
Già di per sé questo porta ad avere un aumento dei problemi di sicurezza, in quanto chiunque ha la possibilità di entrare su quelle porte. Se poi si considera l'enorme incremento degli utenti questi problemi diventano prioritari. Ciò rende fondamentale l'utilizzo di sistemi di difesa come antivirus, firewall, programmi di pulizia dei file di registro e di rimozione degli agenti infettivi: virus, spyware, trojan o malware.
Infatti, i vari utenti malevoli, sfruttano queste tipo di reti per condividere file infettati da malware e spyware. A volte il diritto alla riservatezza può nascondere l'azione di quei

soggetti che operano in maniera da danneggiare gli altri utenti della rete mettendo intenzionalmente in condivisione file infetti, corrotti o non corrispondenti a quanto dichiarato.

Il modello P2P si contrappone alla tradizionale architettura client/server. Su quali aspetti differiscono?

1.1.3.4 Il modello Client/Server

Il modello client/server indica un'architettura di rete nella quale genericamente un computer client o terminale si connette ad un server per la fruizione di una certa risorsa, un certo servizio offerto da quest'ultimo.
È una modalità di comunicazione su cui si basa gran parte dei servizi di internet. La presenza di un server permette di condividere le risorse lasciando che sia appunto il server a gestire gli accessi per evitare conflitti di utilizzo da parte dei client.
Il modello client/server quindi prevede due entità:

- l'entità **Client** che richiede il servizio
- l'entità **Server** che offre il servizio

Il server svolge un ruolo passivo nella comunicazione. Resta in attesa di richieste di servizio da parte dei client e serve ogni richiesta trasmettendo un messaggio di risposta. Il client svolge un ruolo attivo in quanto interroga il server per ottenere un servizio, una risorsa specifica e resta in attesa di un messaggio di risposta.

Tipicamente il rapporto tra server e client è uno-a-molti: la responsabilità del servizio non è affidata omogeneamente a tutte le entità partecipanti.
All'aumentare del numero di richieste, le prestazioni di un sistema client/server diminuiscono. Al crescere degli utenti diminuisce la larghezza di banda disponibile per la singola connessione, e aumentano i tempi di attesa. Le prestazioni del sistema dipendono dal server. Per le prestazioni di un server i miglioramenti possono avvenire soltanto investendo risorse per aumentarne le capacità.
Il server che possiede la risorsa è l'unica entità che può fornire il servizio. In caso di crash del server (malfunzionamento hardware, di rete, etc.) il servizio non è più disponibile.

Per garantire un'adeguata qualità del servizio è necessario adottare opportuni accorgimenti (ridondanza nei sistemi di alimentazione, nello storage, etc.).
Il server, oltre alla gestione logica del sistema, deve implementare tutte le tecniche di gestione degli accessi, allocazione e rilascio delle risorse, condivisione e sicurezza dei dati o delle risorse.

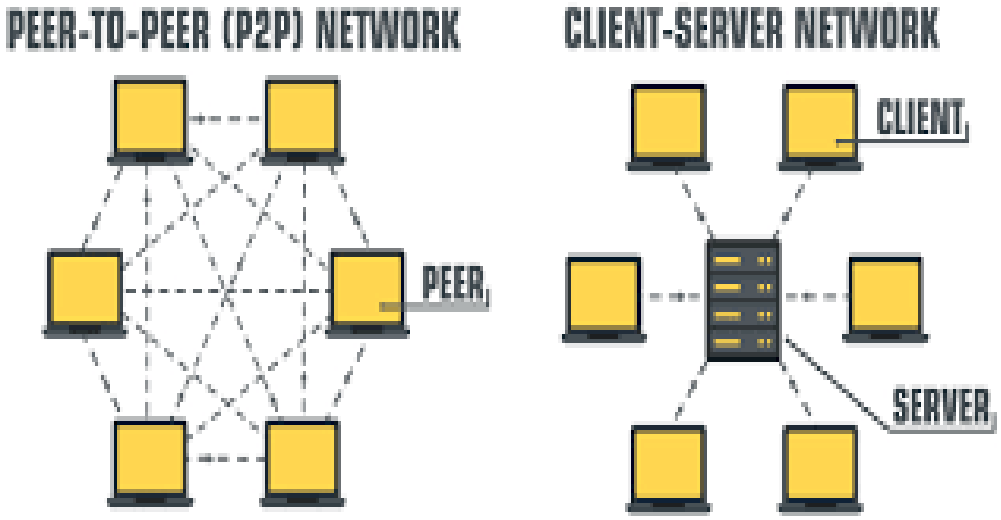

Figura 1.3: Modelli di rete a confronto. [20]

Le interazioni principali tra client e server sono di due tipi:

- **interazione connection oriented**: viene stabilito un canale di comunicazione virtuale prima di iniziare lo scambio dei dati.
- **interazione connectionless**: non c'è connessione virtuale, ma semplice scambio di messaggi.

La scelta tra i due tipi dipende dal tipo di applicazione e anche da altre caratteristiche proprie del livello di comunicazione sottostante. Per esempio, i protocolli del livello di trasporto più utilizzati sono TCP oppure UDP.

1.2 Hash Tables

Una tabella è una struttura dati che rappresenta i dati sotto forma di coppie chiave-valore, cioè, mette in corrispondenza una data chiave con un dato valore. Lo scopo di questo tipo di struttura è quello di memorizzare le varie informazioni e associare le stesse ad una chiave in modo da essere individuate tramite la chiave corrispondente.
Le principali operazioni che si possono effettuare sono:

- Inserimento;
- Eliminazione;
- Ricerca.

Queste sono le classiche operazioni che possono essere effettuate in una qualsiasi tipologia di struttura dati ma per le tabelle, queste operazioni, presentano dei requisiti specifici. Inserire un nuovo elemento è possibile soltanto se la chiave associata non risulti già presente all'interno della tabella e per quanto riguarda la ricerca di un elemento, c'è bisogno della chiave specifica per quell'elemento che si vuole ricercare.
Per ogni elemento E_i si può definire la lunghezza di ricerca, denotata con L_i, come un numero di prove necessarie per poter trovare E_i. Questo numero di prove è inteso come il numero di elementi analizzati prima di trovare l'elemento della ricerca.
La lunghezza media di ricerca S è la media delle S_i per tutti gli elementi della tabella.
Esistono due tipologie di implementazione per le tabelle:

- Tabelle ad indirizzamento diretto;
- Tabelle Hash.

1.2.1 Tabelle ad indirizzamento diretto

Questa tecnica è utilizzata principalmente quando l'insieme delle chiavi totali presenti è ragionevolmente piccolo, limitato. Le chiavi sono rappresentate da numeri interi, l'universo delle chiavi sarà U = {0, 1, ..., n-1}.

Ogni slot della tabella di indirizzi diretti T = {0, 1, ..., n-1} contiene un puntatore all'elemento che corrisponde ai dati. L'indice dell'array T è la chiave stessa e il contenuto di T è un puntatore all'insieme [chiave, elemento]. Se non è presente alcun elemento per una chiave, viene lasciato come NULL. Si ha quindi una corrispondenza biunivoca tra chiavi e posizioni della tabella: l'elemento con chiave k è memorizzato nello slot k.

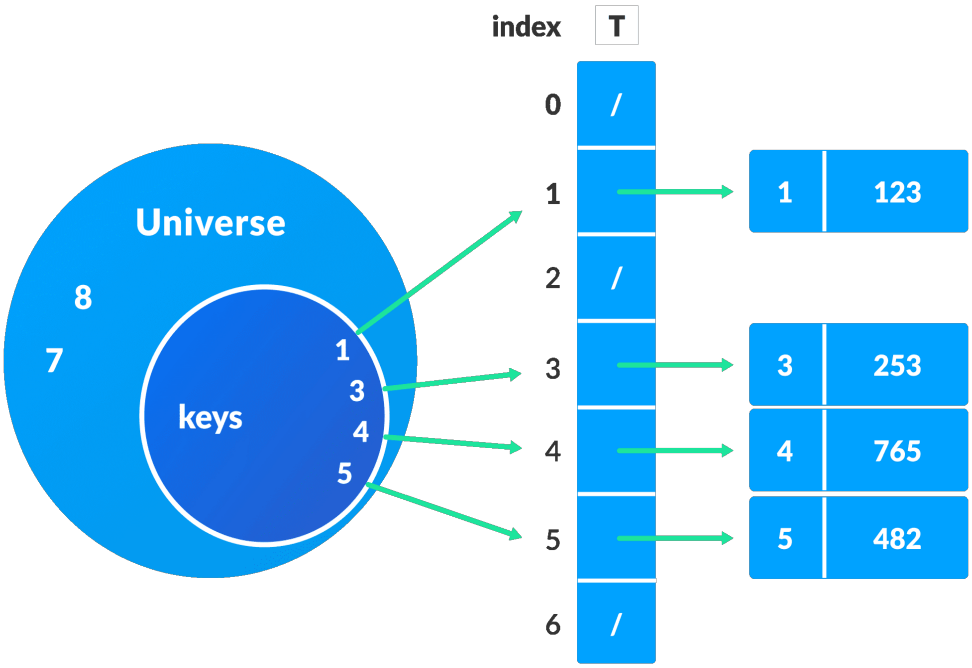

Figura 1.4: Indicizzazione delle chiavi con i rispettivi valori. [21]

Le operazioni di ricerca, inserimento e cancellazione sono facili da implementare ed hanno un costo computazionale costante $\Theta(1)$. Utilizzare questo metodo presenta degli inconvenienti legati soprattutto alla gestione dello spazio.
La dimensione della tabella è data dalla cardinalità di tutte le chiavi presenti, cioè l'universo U delle chiavi; Lo spazio allocato è indipendente dal numero di elementi effettivamente memorizzati (le chiavi effettive, inserite nella tabella). Se l'insieme K delle chiavi effettivamente utilizzate è più piccolo rispetto all'insieme U delle chiavi totali allora la maggior parte dello spazio allocato per la tabella T sarebbe inutilizzato.

Ci può essere, quindi, un grande spreco di memoria ecco perché si potrebbero utilizzare le tabelle hash, le quali consentono un buon compromesso tra lo spazio allocato e il tempo di ricerca.

1.2.2 Tabelle Hash

Il metodo delle tabelle hash [7] permette di dimensionare la tabella in base al numero di elementi presenti utilizzando una particolare funzione per indicizzare la tabella. Questa particolare funzione è la funzione hash. Data una chiave k $\in$ U, l'insieme delle chiavi, restituisce la posizione nella tabella in cui l'elemento con chiave k viene memorizzato.

$$h : U \mapsto [0, 1, \ldots., m\text{-}1]$$

La dimensione m della tabella può non coincidere con la |U|, anzi m $<$ |U| in generale.
Nella tabella hash, l'idea è quella di definire una funzione di accesso che consente di ottenere la posizione di un elemento in base alla sua chiave. La chiave k viene memorizzata nella cella h(k) e viene utilizzata come indice per l'elemento. Generalizzando, le chiavi vengono elaborate per produrre un nuovo indice che esegue il mapping all'elemento richiesto. Questo processo è chiamato hashing.
Il vantaggio che si ottiene è quello di ridurre lo spazio necessario per la memorizzazione di elementi all'interno della tabella. Con la funzione hash quindi, si riesce a risparmiare spazio rispetto ad un indirizzamento diretto visto in precedenza, però, anche questo metodo presenta degli svantaggi.
Utilizzando l'hashing si perde la corrispondenza tra le chiavi e le posizioni nella tabella e potrebbero sorgere dei conflitti tra le chiavi, le cosiddette collisioni.

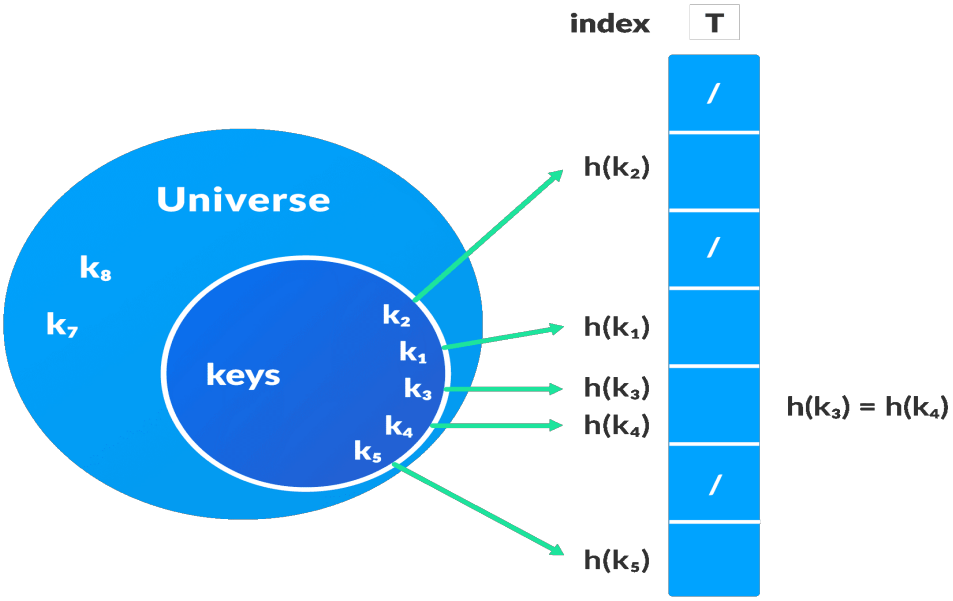

Figura 1.5: Funzione di hashing applicata alle chiavi. [21]

Il fenomeno della collisione si verifica quando due chiavi k1 e k2 corrispondono alla stessa posizione nella tabella, ossia quando h(k1) = h(k2). Per evitare che accade ciò, viene scelta una funzione hash adatta. Tuttavia, è impossibile produrre tutte le chiavi univoche perché |U| > m, cioè la cardinalità dell'insieme delle chiavi è maggiore della grandezza della tabella. Una buona funzione hash potrebbe non impedire completamente le collisioni, però può ridurre il numero delle stesse.

Se si andasse a confrontare il tipo della tabella cioè tabella hash rispetto alla tabella degli indirizzi diretti si può notare che i problemi principali con la tabella degli indirizzi diretti sono la dimensione dell'array e il valore probabilmente grande di una chiave. La funzione hash riduce l'intervallo dell'indice e quindi viene ridotta anche la dimensione dell'array.

Ad esempio, se k = 7845648451, h(k) = 11 (utilizzando una funzione hash). Questo aiuta a risparmiare la memoria sprecata fornendo l'indice di 7845648451 alla tabella.

Ci sono diverse tecniche per risolvere la collisione:

- **Indirizzamento aperto**: tutti gli elementi sono contenuti nella tabella e nel momento in cui una cella risulti occupata, se ne cerca un'altra libera.
- **Liste di collisione**: gli elementi collidenti sono contenuti in liste esterne alla tabella; T_i punta alla lista di elementi tali che h(k) = i.

1.2.3 Indirizzamento aperto

Questa tecnica prevede che si usi solo lo spazio della tabella, allocando gli elementi che determinano delle collisioni in posizioni diverse da quelle che gli spetterebbero.
Nel caso in cui si vuole inserire un elemento con chiave k e la sua posizione "naturale" h(k) è già occupata, si andrà a cercare la cella vuota, nel caso ci fosse, scandendo le celle secondo una sequenza di indici.
Per inserire una nuova chiave si esamina una successione di posizioni della tabella, si esegue quindi una scansione finché non si trova una posizione vuota e poi si inserisce la chiave. La sequenza delle posizioni esaminate dipende dalla chiave che deve essere inserita.

1.2.4 Liste di collisione

Nelle liste di collisione gli elementi collidenti vengono inseriti nella stessa posizione della tabella in una lista concatenata.
La risoluzione delle collisioni mediante concatenamento è una tecnica che, nel momento in cui una funzione hash produce lo stesso indice per più elementi, questi elementi vengono memorizzati nello stesso indice utilizzando un elenco con delle linked list.

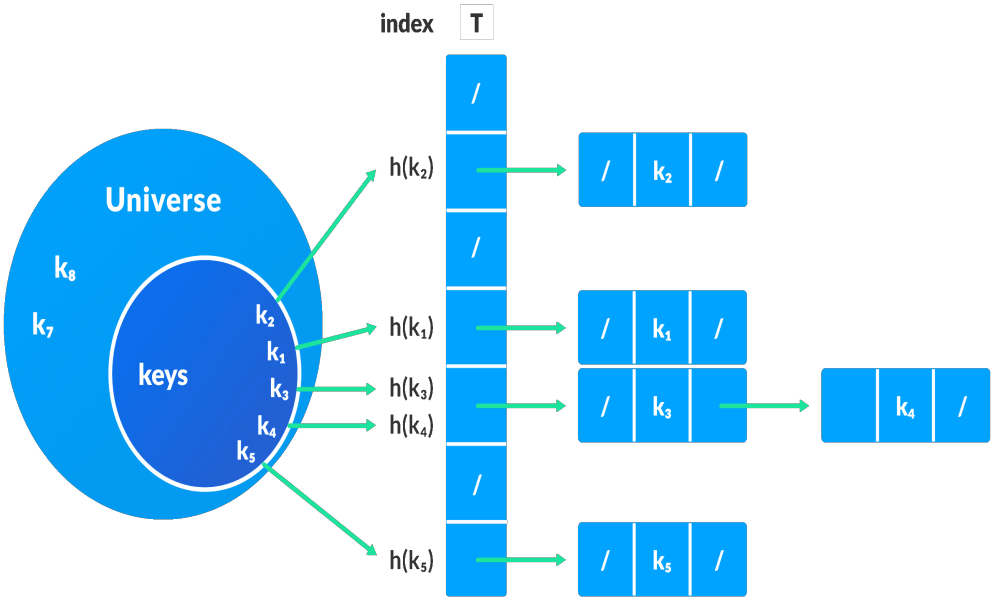

Figura 1.6: Concatenamento degli elementi nella tabella. [21]

In generale, una buona funzione hash deve avere le seguenti caratteristiche:

- Non dovrebbe generare chiavi troppo grandi per evitare lo spreco di spazio.
- Le chiavi generate non dovrebbero essere né molto vicine né troppo lontane.
- La collisione deve essere minimizzata il più possibile.

1.3 Distributed Hash Tables

1.3.1 Caratteristiche e proprietà

Una Distributed Hash Table (DHT) [8] è un sistema di archiviazione decentralizzato che fornisce schemi di ricerca e archiviazione simili a una tabella hash, permettendo l'archiviazione di dati sotto forma di coppie chiave-valore. Gestisce i dati distribuendoli su un numero di nodi e implementando uno schema di routing che consente di cercare in modo efficiente il nodo su cui si trova l'elemento della ricerca. Ogni nodo in un DHT è responsabile delle chiavi assegnategli insieme ai valori mappati.
Inoltre, ogni nodo memorizza una vista parziale dell'intero sistema distribuito, il quale distribuisce efficacemente le informazioni di instradamento. In base a queste informazioni, la procedura di instradamento attraversa tipicamente diversi nodi, avvicinandosi alla destinazione a ogni hop, fino a quando non viene raggiunto il nodo di destinazione.

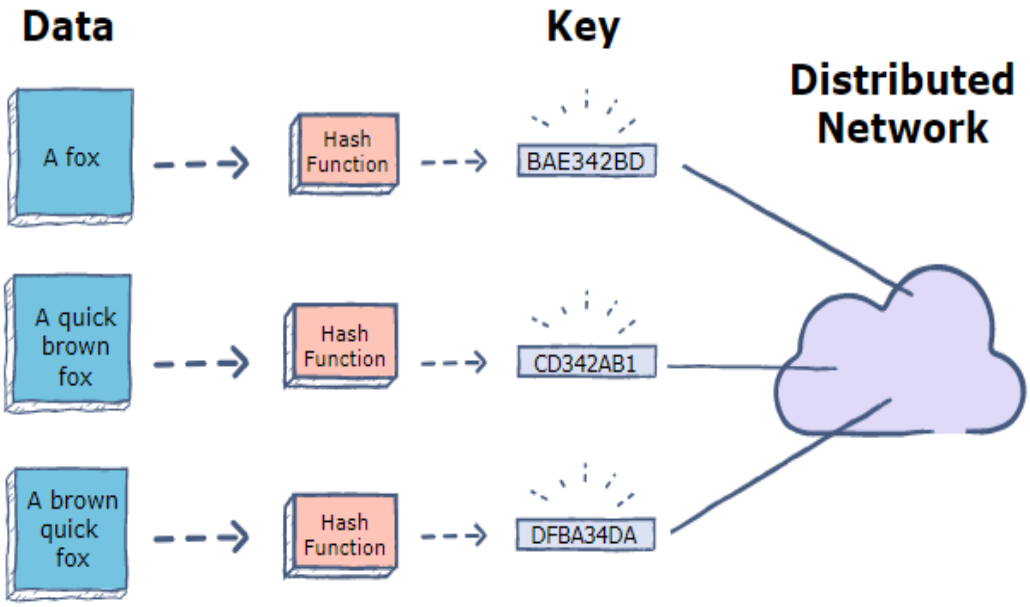

Figura 1.7: Dati sottoposti ad hashing prima di essere instradati nella rete [22]

Proprio come nelle tabelle hash, i valori mappati rispetto alle chiavi in una DHT possono essere qualsiasi forma arbitraria di dati.
Pertanto, le DHT seguono una strategia proattiva per il recupero dei dati strutturando lo spazio di ricerca e fornendo uno schema di routing deterministico. Questa strategia si traduce in query inondate su richiesta in tutta la rete perché il routing non può essere diretto verso l'obiettivo di ricerca. Con un sistema centralizzato, la strategia di ricerca è implicita: l'instradamento di una query non è necessario poiché la procedura di ricerca stessa è limitata a un singolo sistema.
Le DHT sono tipicamente progettate per gestire un numero elevato di nodi ed essere efficienti anche nei casi in cui ci sono nuovi continui ingressi o uscite dal sistema stesso da parte degli utenti.
Questo tipo di infrastruttura è molto utilizzata per servizi quali sistemi peer-to-peer di file sharing, file system distribuiti, multicast, web caching cooperativo.
In un sistema DHT, ad ogni elemento che viene inserito viene assegnato un ID identificativo, un valore univoco dallo spazio degli indirizzi.
Questo valore può essere scelto liberamente dall'applicazione, ma spesso è derivato dai dati stessi tramite una funzione hash resistente alle collisioni. Ad esempio, l'ID di un file potrebbe essere il risultato dell'hashing del nome del file o del file binario completo. Pertanto, la DHT memorizzerebbe il file nel nodo responsabile della parte dello spazio degli indirizzi che contiene l'identificatore.

Le DHT hanno le seguenti proprietà:

- **Decentralizzazione**: i nodi formano nell'insieme il sistema di rete senza alcuna autorità centrale.
- **Scalabilità**: il sistema è configurato per il funzionamento anche con migliaia o milioni di nodi connessi.
- **Tolleranza ai guasti**: il sistema è in grado di risultare sempre affidabile, continuamente adoperabile, nonostante quantità anche elevate di nodi che si uniscono, si disconnettono o escono volontariamente dalla rete in ogni momento.

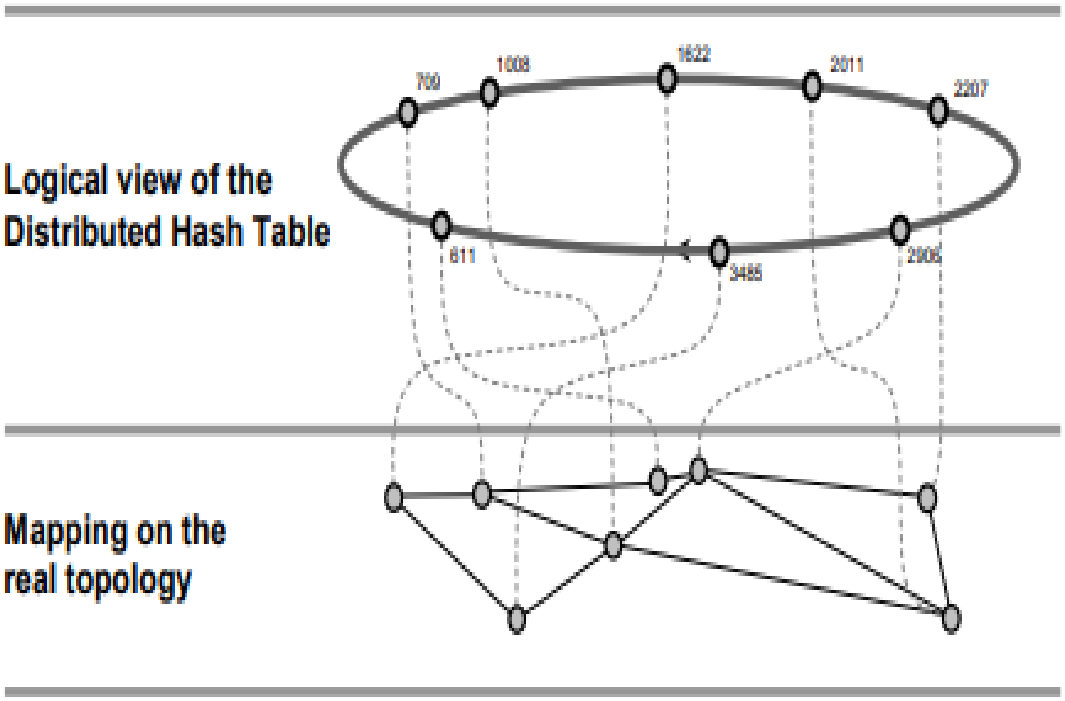

Figura 1.8: Vista dell'overlay e underlay di una Distributed Hash Table. [23]

Ciascun nodo all'interno della rete è in contatto con un gruppo ristretto di altri nodi della rete. Ogni nodo, quindi, ha dei vicini grazie alla quale riesce a scambiare informazioni e con questo procedimento si riesce a trasferire un'informazione da una parte all'altra della rete.
Queste strutture DHT implementano continuamente meccanismi di sicurezza per contrastare azioni malevoli o eventuali problemi che potrebbero insorgere.

1.3.2 Routing delle informazioni

Essendo un sistema distribuito, i nodi comunicano tra di loro per individuare il nodo responsabile di un determinato bucket e per fare questo occorrono meccanismi di comu-

nicazione scalabili ed efficienti.
Il routing è una funzionalità fondamentale della Distributed Hash Table. In base a una procedura di instradamento, i messaggi con i relativi ID di destinazione vengono consegnati al nodo della rete che gestisce l'ID di destinazione. Pertanto, sono gli algoritmi di instradamento che risolvono il problema di ricerca. I sistemi DHT esistenti implementano un'ampia varietà di approcci al routing.
Tuttavia, il principio fondamentale è fornire a ciascun nodo una vista limitata dell'intero sistema memorizzando su di esso un numero limitato di collegamenti ad altri nodi. Quando un nodo riceve un messaggio, se non è il destinatario del messaggio, lo inoltra a uno dei vicini. Questo processo viene ripetuto in modo ricorsivo finché non viene trovato il nodo di destinazione.
La scelta del nodo successivo è determinata dall'algoritmo di routing e dalla metrica di routing. Una metrica tipica è quella della vicinanza numerica: i messaggi vengono sempre inoltrati al nodo che gestisce gli identificatori numericamente più vicini all'ID di destinazione del messaggio. Idealmente, un tale schema instrada in modo affidabile un messaggio alla sua destinazione in un piccolo numero di hop. Ovviamente, è difficile progettare algoritmi e metriche di routing in modo tale che i guasti dei nodi e le informazioni di routing errate abbiano un impatto limitato o scarso sulla correttezza del routing e sulla stabilità del sistema.

1.3.3 Data Storage

Esistono due possibilità per memorizzare i dati in una Distributed Hash Table.
In una DHT che utilizza l'archiviazione diretta, i dati vengono copiati al momento dell'inserimento nel nodo responsabile (Figura 1.9 (a)).
Il vantaggio di questo metodo è che i dati si trovano direttamente nel sistema e il nodo che li ha inseriti può successivamente lasciare la rete senza che i dati divengano indisponibili. Lo svantaggio è il sovraccarico in termini di archiviazione e larghezza di banda di rete. I dati devono essere replicati su diversi nodi per aumentarne la disponibilità. Inoltre, per dati di grandi dimensioni, è necessaria un'enorme quantità di spazio di archiviazione su

ogni nodo.
L'altra possibilità è memorizzare i riferimenti ai dati. Il nodo di inserimento posiziona solo un puntatore ai dati nella DHT. I dati stessi rimangono su questo nodo, portando a un carico ridotto nella rete (Figura 1.9 (b)). Tuttavia, i dati sono disponibili solo fino a quando il nodo è disponibile.

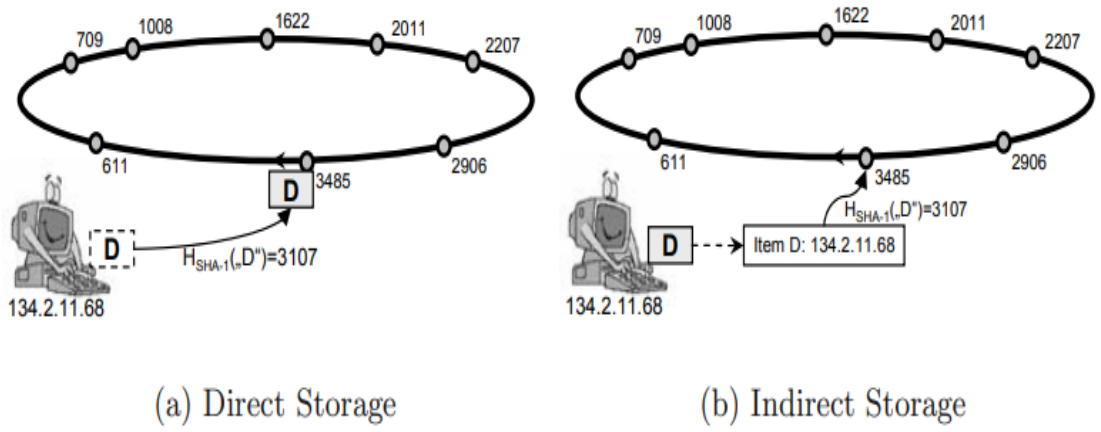

Figura 1.9: Due metodi per l'archiviazione dei dati nelle Distributed Hash Table. [23]

Le principali funzioni di una DHT sono:

- Gestione delle informazioni e partizionamento degli indirizzi.
- Assegnazione di informazioni ai nuovi nodi.
- Re-assegnamento e re-distribuzione delle informazioni in caso di fallimento o disconnessione volontaria dei nodi dalla rete.
- Bilanciamento delle informazioni tra i nodi.

1.3.3.1 Gestione delle informazioni e partizionamento degli indirizzi

In una DHT, ogni nodo è responsabile della gestione di uno o più bucket. Per bucket si intende un insieme di parole chiavi che vengono affidate ad un particolare nodo per la gestione. La struttura, lo schema di partizionamento dell'overlay network in cui ogni nodo è connesso ad altri nodi e gestisce un sottoinsieme dell'insieme totale del keyspace, è un aspetto molto importante e caratteristico di una DHT. Parlare di spazio degli indirizzi o keyspace è indifferente in quanto fanno riferimento allo stesso concetto, cioè, si riferiscono a quella porzione specifica di dati che deve essere gestita da un nodo specifico, il tutto in base al partizionamento della rete.
Diverso è il modo in cui una DHT implementa lo schema per partizionare le informazioni all'interno della rete. Un tipico funzionamento di una DHT consiste inizialmente nel preparare il file per la memorizzazione andando a calcolare l'hash del filename con una funzione di hashing. Così facendo, si produce una chiave k che verrà inserita in un messaggio insieme al contenuto del file stesso. Il messaggio *put(k,data)* viene inviato con destinazione il nodo predisposto al suo mantenimento.
Il messaggio è inoltrato di nodo in nodo attraverso l'overlay network fino a che esso non raggiunge il singolo nodo che è responsabile per la chiave k. In qualsiasi momento, si possono recuperare i contenuti del file calcolando a sua volta l'hash del filename per ottenere k e chiedere ad un qualsiasi nodo della rete DHT di trovare il dato associato a k tramite un messaggio *get(k)*. Il messaggio verrà quindi inoltrato attraverso l'overlay verso il nodo responsabile per k, che risponderà con una copia del dato immagazzinato.
Ciascun nodo mantiene un set di collegamenti verso i nodi vicini, cioè quei nodi che hanno un collegamento diretto verso il nodo in questione. Tutti questi nodi insieme formano questa overlay network, e vengono organizzati in modo tale da formare la topologia della rete. Per ricercare un'informazione all'interno di questa rete, si inoltra il messaggio al vicino il cui ID è più vicino alla chiave k utilizzando un algoritmo di ricerca.
La ricerca procede fino a quando non si trova il nodo che è in possesso delle informazioni richieste.
L'aspetto da tenere in considerazione nel momento in cui si utilizza un determinato algoritmo è quello che concerne il tempo della ricerca. Un algoritmo è efficiente se conduce sempre ad una soluzione, nel caso essa esista, e se lo fa in un tempo ottimale. La scelta

dell'algoritmo è quindi importante in quanto bisogna cercare di mantenere sempre un numero di passi relativamente basso durante le ricerche che si effettuano.

1.3.3.2 Assegnazione di informazioni ai nuovi nodi

L'entrata di un nuovo nodo deve essere predisposta in modo tale da non creare scompiglio con gli altri nodi. L'inserimento di un nodo comporta l'assegnazione di una porzione del keyspace che dovrà gestire per conto della rete. Alla base per permettere questa operazione in maniera efficace c'è un buon partizionamento del keyspace.
Molte DHT usano alcune varianti del consistent hashing per mappare le chiavi ai nodi. Considerando che ogni nodo è identificato con un ID, il consistent hashing ha la caratteristica fondamentale che la rimozione o l'inserimento di un nodo modifica solo l'insieme di chiavi possedute dai nodi con ID adiacente ad esso, senza coinvolgere tutti gli altri nodi.
Se lo si confronta con un hash table tradizionale, si nota in maniera evidente che l'inserimento o la rimozione di un nodo causa un riassetto di quasi tutto l'intero keyspace. Ogni modifica nel sottoinsieme di chiavi di cui un nodo è responsabile corrisponde tipicamente ad un intenso movimento da un nodo all'altro di oggetti immagazzinati nella DHT. Una minimizzazione di questi movimenti è necessaria al fine di far fronte in maniera efficiente ad un numero elevato di ingressi e di uscite dei vari nodi.
Una volta assegnato al nodo la porzione del keyspace, questo viene inserito nella DHT e si collega con i nodi vicini.

1.3.3.3 Re-assegnamento e re-distribuzione delle informazioni in caso di fallimento o disconnessione volontaria dei nodi dalla rete

Nel momento in cui un nodo si disconnette in modo inatteso o esce volontariamente dal network, tutti i dati memorizzati nel nodo vengono persi a meno che non siano stati memorizzati su altri nodi.

Quando un nodo esce dalla rete la responsabilità viene ceduta ad un altro ed è quindi importante attuare processi di memorizzazione ridondante (replicazione) dei dati presenti in un nodo verso altri nodi vicini. Partizionare la propria porzione di dati permette di evitare una perdita completa delle informazioni. Adoperare percorsi di routing alternativi/ridondanti consente di avere una maggiore sicurezza nel momento in cui un nodo esce dalla rete.
In linea di principio, i nodi che lasciano volontariamente una DHT potrebbero essere trattati allo stesso modo dei nodi guasti. Tuttavia, le implementazioni DHT spesso richiedono che i nodi in partenza notifichino al sistema prima di partire. Ciò consente ad altri nodi di copiare i dati dell'applicazione dal nodo uscente e di aggiornare immediatamente le proprie informazioni portando a una migliore efficienza di instradamento. Se attivati in modo esplicito, i meccanismi di replica e bilanciamento del carico possono anche funzionare in modo più efficiente e affidabile.

1.3.3.4 Bilanciamento delle informazioni tra i nodi

La maggior parte dei sistemi DHT tenta di distribuire uniformemente il carico di messaggi di instradamento e di memorizzare i dati sui nodi partecipanti.
Tuttavia, ci sono diverse ragioni per cui alcuni nodi del sistema possono subire carichi più elevati di altri: un nodo gestisce una porzione molto ampia dello spazio degli indirizzi, un nodo è responsabile di una porzione dello spazio degli indirizzi con un numero molto elevato di elementi o un nodo gestisce gli elementi che sono particolarmente popolari. In queste circostanze, meccanismi di bilanciamento del carico sono un aspetto molto importante da tenere in considerazione.
Gestire grosse porzioni del keyspace può comportare sbilanciamento del carico e questo può causare minore robustezza del sistema e minore scalabilità. Nel momento in cui un nodo deve gestire diverse queries perché ha accumulato una grande porzione del sottospazio, si va a creare una sorta di collo di bottiglia nel quale il nodo si ritrova a dover rispondere a numerose richieste, andando così ad incidere sull'efficienza della rete. La complessità del tempo di ricerca pari a O(log N) potrebbe non essere garantita.

Per ovviare a questo problema c'è bisogno di definire dei buoni algoritmi di bilanciamento del carico. Servono principalmente per rendere omogenee le informazioni presenti tra tutti i nodi della rete. Fondamentale per l'efficienza della ricerca.

1.3.4 Interfaccia DHT

Esistono due angolazioni da cui è possibile visualizzare le funzionalità delle tabelle hash distribuite: possono essere interpretate come sistemi di routing o come sistemi di archiviazione. La prima interpretazione si concentra sulla consegna di pacchetti ai nodi in una DHT in base a un ID di destinazione. Nella seconda, una tabella hash distribuita appare come un sistema di archiviazione simile a una tabella hash.
Queste nozioni si riflettono nell'interfaccia che una tabella hash distribuita fornisce alle applicazioni.

1.3.4.1 Interfaccia di routing

L'instradamento in una tabella hash distribuita viene eseguito nello spazio degli indirizzi logici suddiviso tra i nodi partecipanti. Qualsiasi chiave dello spazio degli indirizzi può servire come indirizzo di destinazione per un messaggio. Pertanto, la funzionalità fornita dalla DHT è di inoltrare un messaggio al nodo responsabile di questa chiave.
Un'interfaccia con due primitive è sufficiente per costruire applicazioni distribuite su questa base. La primitiva di invio accetta un ID di destinazione e un messaggio e consegna il messaggio da un nodo arbitrario nel sistema al nodo che gestisce l'ID di destinazione. La primitiva di ricezione passa i messaggi in arrivo e i loro identificativi di destinazione all'applicazione sul nodo ricevente.
Tutti gli altri dettagli della gestione DHT, come l'arrivo e la partenza del nodo o i meccanismi di riparazione, vengono implementati dalla tabella hash distribuita stessa e non sono esposti all'applicazione. Questa interfaccia generica e senza stato implementa poche funzionalità ma lascia molta flessibilità alla progettazione dell'applicazione.

In particolare, la memorizzazione e il recupero dei dati, comprese le strategie di bilanciamento del carico, possono essere implementate sopra l'interfaccia di instradamento.

1.3.4.2 Interfaccia di archiviazione

Come sistema di archiviazione, una tabella hash distribuita implementa un'interfaccia per l'archiviazione persistente e il recupero affidabile dei dati in modo distribuito. Su ogni nodo, l'interfaccia dell'applicazione fornisce le due primitive principali di una tabella hash.
La primitiva *put* prende una coppia (chiave, valore) e la memorizza sul nodo responsabile della chiave. Allo stesso modo, la primitiva *get* accetta una chiave e restituisce il valore associato alla chiave specificata.
L'implementazione di questa interfaccia aggiunge a una tabella hash distribuita un altro livello di complessità oltre al routing corretto ed efficiente. Il livello di archiviazione deve affrontare gli errori di instradamento, prevenire la perdita di dati a causa di errori del nodo attraverso la replica, ottenere il bilanciamento del carico.

1.3.4.3 Interfaccia client

Date le interfacce di cui sopra, un nodo può utilizzare le sue primitive solo dopo essersi unito a una tabella hash distribuita. Tuttavia, un sistema distribuito può anche essere strutturato in modo tale che i nodi che partecipano alla DHT rendano disponibili i servizi DHT ad altri host non partecipanti. In un tale ambiente, questi host agiscono come client dei nodi DHT.
Questa configurazione può essere utile quando, ad esempio, la tabella hash distribuita viene eseguita come servizio di infrastruttura su un set di nodi dedicato per una maggiore affidabilità.
L'interfaccia tra client e nodi DHT è anche adatta per realizzare il controllo degli accessi e la contabilità per i servizi disponibili sulla tabella hash distribuita. Si noti che questa

interfaccia può essere implementata come un'applicazione sopra il livello di routing o archiviazione DHT.

Di seguito sono spiegate in maniera più dettagliata le funzioni hash e il consistent hashing, elementi fondamentali in una DHT.

1.3.5 Funzioni Hash

Una funzione hash h, o message digest function, è una funzione che mappa un qualsiasi tipo di dato m in una stringa di lunghezza prefissata, in genere un numero intero, cercando di far in modo che da questa stringa non si possa risalire al messaggio che l'ha generata. Queste funzioni permettono di dare un'impronta digitale tale da identificare univocamente il tipo di dato sottoposto ad hashing.
La lunghezza della stringa finale è direttamente correlata con la sicurezza della funzione di hash, perché più una stringa è lunga, minore sarà la probabilità di trovare due messaggi con lo stesso digest, cioè con la stessa stringa finale. La stringa d=h(m) può essere vista come l'impronta digitale del messaggio, teoricamente irripetibile.
Le caratteristiche principali di una funzione di hash sono:

1. l'efficienza computazionale, perché il messaggio m potrebbe essere molto lungo;
2. la probabilità che d=h(m) per un qualsiasi messaggio casuale m deve essere 2^n, dove il digest è lungo n bit.
3. one way property o preimage resistance: dato il digest d, dev'essere computazionalmente impossibile calcolare il messaggio che l'ha generato;
4. dato il messaggio m tale per cui d=h(m) , deve essere computazionalmente impossibile trovare un altro messaggio m' tale per cui h(m)=h(m')=d;
5. deve essere computazionalmente impossibile trovare due messaggi m' ed m" che collidono, qualunque sia il loro digest.

Le funzioni hash hanno molti usi e per ognuno possono essere desiderate proprietà differenti. Esiste un tipo di funzione hash nota come funzione hash crittografica, che deve soddisfare un insieme restrittivo di proprietà e viene utilizzata per scopi di sicurezza, comprese applicazioni come la protezione della password, il controllo dell'integrità dei dati e l'impronta digitale dei messaggi.

1.3.6 Consistent Hashing

Nei sistemi distribuiti l'inserimento o l'uscita di nuovi nodi nella rete potrebbe causare dei problemi nella mappatura delle chiavi. Il consistent hashing è stato progettato per evitare il problema di dover modificare l'assegnazione delle chiavi ai vari nodi nel momento in cui uno di essi viene aggiunto o rimosso.
Il consistent hashing si può definire come uno schema di hashing distribuito che opera indipendentemente dal numero di nodi e delle chiavi in una DHT, assegnando ad essi una posizione su un cerchio astratto, il cosiddetto hash ring.

L'idea principale è quella di utilizzare una funzione hash che permetta di mappare casualmente sia le chiavi che i nodi su questo cerchio astratto. Ogni chiave viene quindi assegnata al nodo successivo che appare sul cerchio, in senso orario. Ciò consente una distribuzione uniforme delle chiavi sui nodi e, cosa più importante, se un nodo si disconnette o viene rimosso dal cerchio, solo le chiavi che sono state mappate su quel nodo devono essere riassegnate al nodo successivo in senso orario. Discorso analogo per quanto riguarda l'inserimento di un nuovo nodo.
Il consistent hashing consente di bilanciare correttamente il carico dei vari nodi della rete e consente al sistema di essere scalabile senza influire sul suo funzionamento.

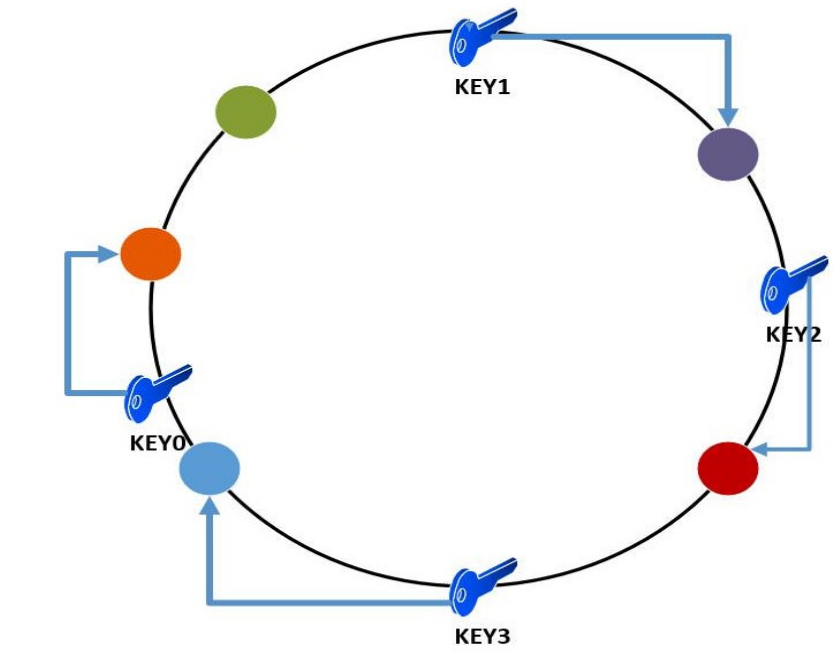

Figura 1.10: Consistent hashing sull'hash ring. [24]

1.4 Protocolli di routing su DHT

Diverse soluzioni sono state proposte per permettere una ricerca più accurata e semplificata e ovviare al problema di dove e come fare per reperire in maniera sicura dei dati specifici. Di seguito verranno approfonditi due protocolli, Chord e Kademlia, che implementano dei meccanismi per gestire questo problema.

1.4.1 Chord

I sistemi e le applicazioni peer-to-peer sono sistemi distribuiti senza alcun controllo centralizzato o organizzazione gerarchica, dove l'esecuzione del software in ogni nodo è equivalente in funzionalità. I compiti di una applicazione peer-to-peer sono molteplici ma l'operazione centrale nella maggior parte dei sistemi distribuiti è la locazione efficiente dei dati.
Chord [9] è un protocollo scalabile per la ricerca in sistemi dinamici peer-to-peer con frequenti entrate ed uscite di nodi. Data una chiave, il protocollo Chord assegna tale chiave ad un nodo. Per eseguire l'associazione della chiave al nodo, Chord utilizza una variante del consistent hashing, la quale tende a bilanciare il carico dei nodi in modo tale che ognuno abbia un numero ben distribuito di chiavi ed evitando così il sovraccarico degli stessi.
La funzione consistent hashing assegna ad ogni nodo e chiave un identificatore *m-bit* utilizzando SHA-1 come funzione hash. L'ID di un nodo viene scelto mediante l'hashing dell'indirizzo IP del nodo e lo stesso processo viene effettuato sulla chiave. La lunghezza dell'identificatore m deve essere abbastanza grande da evitare che due o più nodi e chiavi possano avere lo stesso ID.
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 con una minima possibilità di collisione.

1.4.1.1 Meccanismi di routing

Gli identificativi sono ordinati in un "identifier circle", una sorta di anello, modulo 2^m. La chiave K è assegnata al primo nodo il cui identificativo è uguale o segue K nello spazio degli identificativi. Questo nodo è chiamato "successor node" della chiave K, denotato da successor(K). Se gli identificativi sono rappresentati come un cerchio di numeri da 0 a 2^m-1, allora successor(K) è il primo nodo in senso orario da K. Infatti, 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'‘e un nodo, il successore del nodo 0 è il nodo 1 e il predecessore del nodo 0 è il nodo 2^m-1. Nella Figura 1.11 è mostrato un identifier circle con m=3. Il circle ha tre nodi: 0, 1, e 3. Il successore dell'identificativo 1 è 1, così la chiave 1 dovrebbe essere allocata al nodo 1. Similmente la chiave 2 dovrebbe essere allocata al nodo 3, e la chiave 6 al nodo 0.

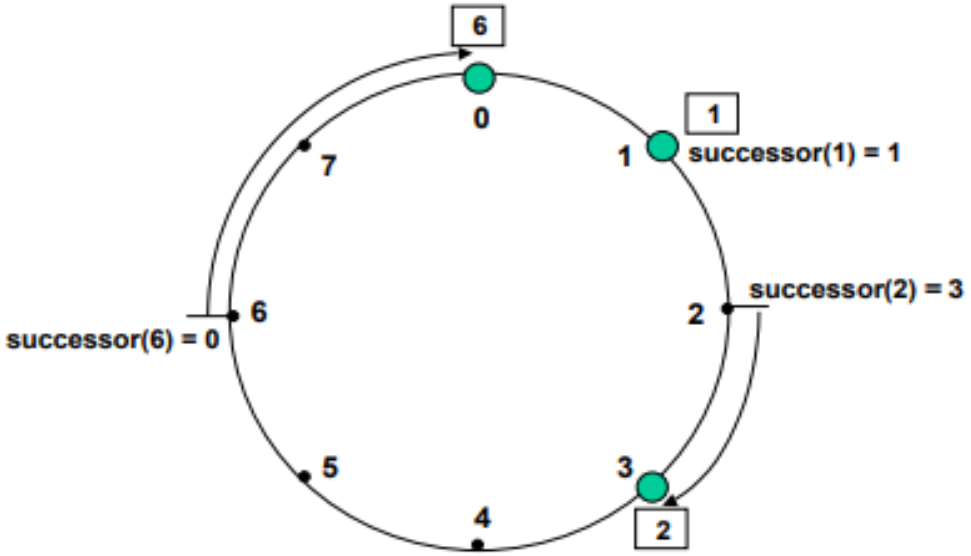

Figura 1.11: Identifier circle con nodi e chiavi memorizzate. [9]

Il consistent hashing è progettato per permettere ai nodi di entrare e lasciare la rete con il minimo disordine. Per mantenere l'assegnazione effettuata dal consistent hashing quando un nodo n entra nella rete, inevitabilmente alcune chiavi assegnate al successore di n ora vengono assegnate ad n. Quando un nodo lascia la rete, ognuna delle sue chiavi viene riassegnata al successore di n. 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.

Un numero veramente piccolo di informazioni di routing è necessario per implementare consistent hashing in un ambiente distribuito. Ogni nodo ha necessità di conoscere

solo il nodo successore nell'identifier circle. Le richieste per un determinato identificatore possono essere passate lungo l'anello attraverso i puntatori ai successori fino a che non si incontra un nodo che succede l'id.
Una porzione del protocollo Chord mantiene questi puntatori ai successori, in modo da far sì che ogni ricerca sia risolta correttamente.
Tuttavia, questo schema di risoluzione è inefficiente: potrebbe richiedere di attraversare tutti gli N nodi per trovare l'assegnazione adeguata. Per accelerare questo processo, Chord mantiene un'informazione di routing addizionale.

1.4.1.2 Locazione scalabile delle chiavi

Ogni nodo n mantiene una tabella di routing con al più m elementi, chiamata "finger table". L'i-esimo elemento nella tabella del nodo n contiene l'identità del primo nodo, s, che succede n di almeno 2^{i+1} nel identifier circle, ovvero s = successor(n + 2^{i+1}) dove $1 \leq$ i $\leq$ m (tutto modulo 2^m). Chiamiamo il nodo s l'i-esimo finger del nodo n. Un elemento della finger table include insieme l'identifier e l'indirizzo IP del nodo rilevante. Notare che il primo finger di n è l'immediato successore nell'identifier circle; Nell'esempio mostrato in Figura 1.12, la finger table del nodo 1 punta ai nodi successori degli identifiers $(n + 2^0) mod\ 2^3 = 2$, $(n + 2^1) mod\ 2^3 = 3$, e $(n + 2^2) mod\ 2^3 = 5$, rispettivamente.
Il successore dell'identifier 2 è il nodo 3, dal momento che è il primo nodo che segue 2, il successore dell'identifier 3 è il nodo 3, e il successore di 5 è il nodo 0. Questo schema ha due importanti caratteristiche.
Primo, ogni nodo mantiene informazione riguardo ad un piccolo numero di altri nodi, e conosce di più circa i nodi più prossimi che lo seguono nell'identifier circle piuttosto che sui nodi molto lontani. Secondo, una finger table di un nodo generalmente non contiene abbastanza informazioni per determinare il successore di un'arbitraria chiave K. Per esempio, il nodo 3 in Figura 1.12 non conosce il successore di 1, e il successore del nodo 1 non appare nella finger table del nodo 3.
Cosa succede quando un nodo n non conosce il successore di una chiave K?

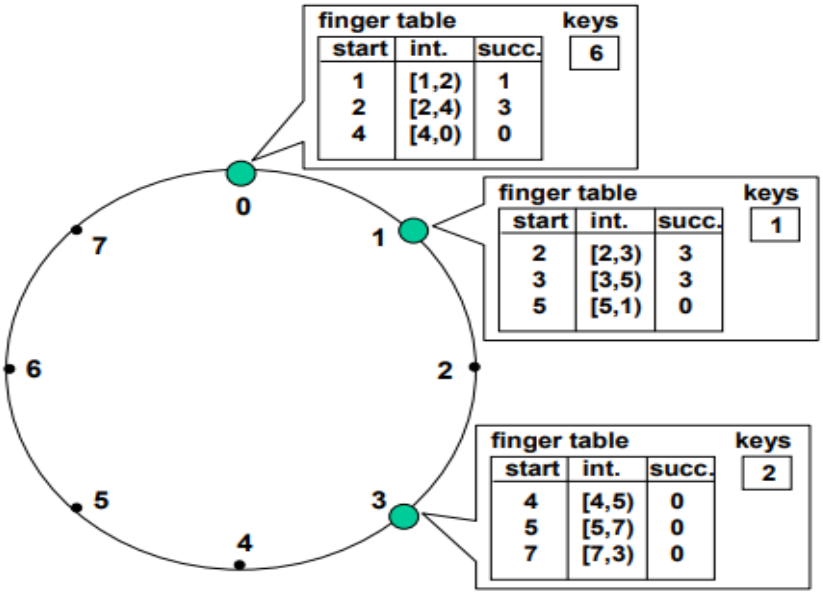

Figura 1.12: Identifier circle e finger table. [9]

Se n può trovare un nodo il cui ID è più vicino del proprio a K, allora tale nodo conoscerà di più riguardo all'identifier circle nella regione di K di quanto sappia n. Quindi n cerca nella sua finger table un nodo j il cui ID preceda K da vicino, e inoltra a j la richiesta. Ripetendo questo processo, n impara circa i nodi con ID più vicini e più vicini a K.

Utilizzando le tabelle finger, Chord individua il successore(K) in O(log N) hop con alta probabilità. 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.

1.4.2 Kademlia

Kademlia [10] è un protocollo di rete peer-to-peer che regola la comunicazione tra i nodi e il modo in cui deve essere effettuata.
Kademlia è una DHT nella quale ogni nodo è identificato da un identificativo di 160 bits utilizzando la funzione di hashing SHA-1. Anche le chiavi hanno un identificativo pari a 160 bits. I nodi della rete comunicano tra di loro utilizzando il protocollo di trasporto UDP.
Come rete P2P, le associazioni tra i nodi e gli oggetti che essi condividono non viene registrata su server particolari ma viene gestita dai nodi stessi della rete. Ogni client agisce come server per l'associazione tra alcune chiavi e i rispettivi oggetti. Ogni oggetto che vuole essere memorizzato nella rete gli viene calcolato l'hash corrispondente e instradato verso quel nodo che gestisce il sottoinsieme a cui corrisponde.
Essendo una DHT, vengono memorizzate nei nodi coppie di valori secondo una metrica definita sull'OR esclusivo. Ogni coppia viene assegnata al nodo il cui identificatore è più vicino alla chiave.
Ad esempio la distanza tra l'identificatore ID1 = 0100 e l'identificatore ID2 = 0111 è la seguente: Distanza(0100,0111) = 0100 XOR 0111 = 0011 = 3. Infatti, la rete definisce un concetto di distanza che permette di stabilire la prossimità tra due nodi in modo che, dato un nodo A è sempre possibile determinare se un nodo B risulta più vicino di un nodo C. Ogni nodo ha una conoscenza della rete che diminuisce all'aumentare della distanza dal nodo stesso, e, quindi, ha una conoscenza molto dettagliata della rete nel proprio vicinato, una discreta conoscenza della rete a media distanza e solo una conoscenza sparsa dei nodi molto lontani.

1.4.2.1 Meccanismi di routing

Kademlia appartiene alla famiglia delle DHT basate su prefix-matching (a cui appartengono anche altre come Pastry e Tapestry). Nel prefix-matching, la ricerca è basata sul metodo *key lookup*. Se un nodo A cerca un oggetto la cui chiave è K, questo ricerca all'interno della propria routing table un nodo B il cui ID ha una configurazione binaria

coincidente nei *b bits* più significativi con la configurazione di K.
Il nodo B ricerca all'interno della propria routing table quel nodo C identificato da un ID la cui configurazione binaria coincide almeno nei *2b bits* più significativi con K. Il lookup prosegue ad ogni hop andando a trovare un match di lunghezza sempre maggiore tra la chiave e l'identificatore dei nodi. Il lookup termina non appena non è possibile individuare all'interno della tabella di routing un match di lunghezza maggiore.
Le Prefix Match DHT sono caratterizzate da tabelle di routing strutturate ad albero. In Kademlia la tabella di routing di ogni nodo può essere rappresentata come un albero binario non bilanciato. Ogni foglia dell'albero binario corrisponde ad un prefisso binario e contiene un insieme di puntatori a nodi caratterizzati da quel prefisso.
La lista di riferimenti ai nodi memorizzati nella tabella di routing permette al nodo di poter scegliere, ad ogni lookup, tra diversi nodi a cui inoltrare la richiesta.
Il fatto di avere diverse scelte garantisce una maggiore robustezza e tolleranza ai guasti e si ha la possibilità di scegliere percorsi di routing alternativi.
Il nodo che invia una richiesta di lookup coordina l'intero processo di ricerca perché ad ogni passo, un nodo invia una richiesta di lookup ed attende una risposta, la quale, indica quale è il successivo passo di routing. Si ha quindi un routing iterativo su cui è basata la rete Kademlia e le varie ricerche vengono soddisfatte con un numero di hop medio pari a O(log N).

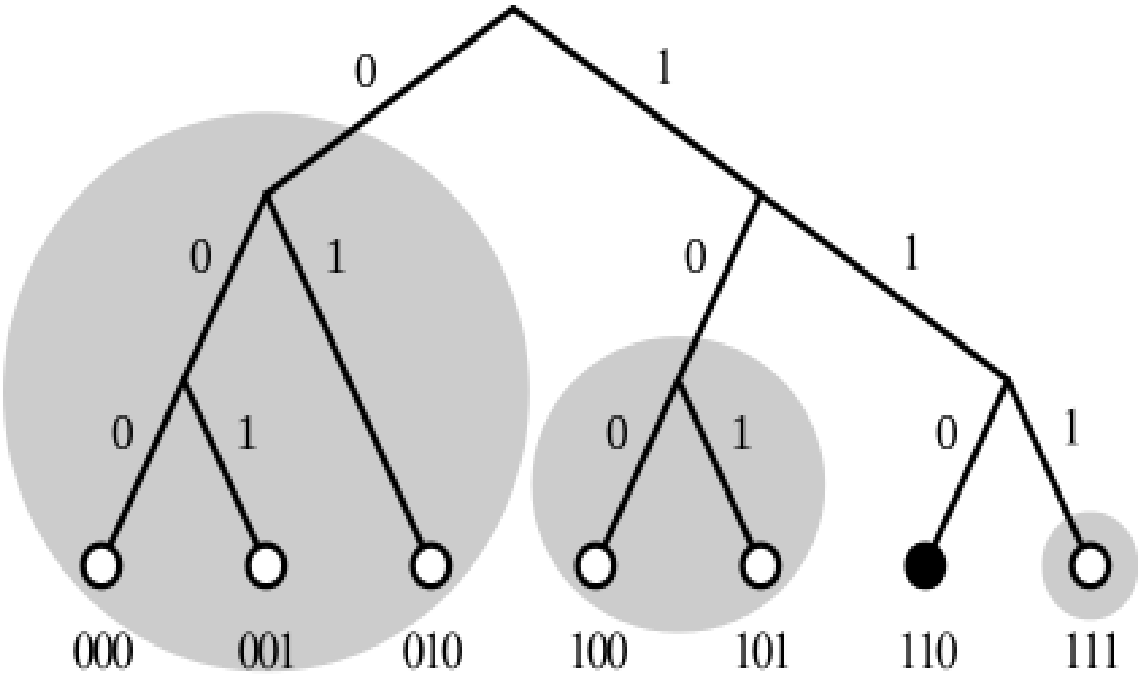

Figura 1.13: Nella tabella di routing, gli elementi corrispondono alle foglie dell'albero binario. [10]

L'architettura della ricerca sui sistemi di trasporto intelligenti (ITS) fa uso di tecnologie che si basano su DLT, cioè, registri distribuiti. Nello specifico viene utilizzata la blockchain di Ethereum come piattaforma di smart contracts e la DLT di IOTA per archiviare e certificare i dati.

1.5 Distributed Ledger Technology

Si indica con DLT, cioè Distributed Ledger Technology, un registro distribuito [11] su diversi nodi che partecipano ad una rete peer-to-peer in comune.
Ognuno di questi nodi replica e salva una copia identica del libro mastro e questa copia

si aggiorna in maniera indipendente.
Il vantaggio principale nell'utilizzo di una DLT è la mancanza di un'autorità centrale o un intermediario che elabora, convalida o autentica le transazioni. Ogni volta che viene inserita una transazione nel registro, i nodi votano tramite un algoritmo di consenso su quale copia sia corretta.
Una volta che il consenso è stato determinato, questi record vengono memorizzati nel libro mastro e tutti gli altri nodi si aggiornano con la nuova copia corretta del libro mastro. La sicurezza è garantita da chiavi e firme crittografiche.
Tutti i file nel registro distribuito vengono quindi contrassegnati con data e ora e dotati di una firma crittografica univoca. La tecnologia fornisce una cronologia verificabile di tutte le informazioni memorizzate su quel particolare set di dati.
Dunque, il primo vero grande passaggio tra la gestione dei ledger tradizionali e i ledger distribuiti, come per esempio la blockchain, è data dal fatto che i libri mastro sono molteplici e che sono accessibili a tutti. Il secondo grande passaggio riguarda il fatto che tutti possono effettuare una transazione o modificarne una esistente. In entrambi i casi questa richiesta potrà essere attuata solo se tutti (o la maggior parte degli utenti) accettano di attuarla.
A prescindere comunque dal fatto che l'operazione sia autorizzata da tutti o da un determinato numero di partecipanti certamente la richiesta di transazione sarà accettata solo se i partecipanti concordano sulla sua legittimità. Ora la domanda è come avviene questo controllo? Possibile che tutti i partecipanti debbano effettuare controlli personali su ciascuna transazione? Con la blockchain, questi controlli vengono eseguiti in modo affidabile e automatico per conto di ciascun utente. Ogni operazione contribuisce a creare un sistema di ledger rapido e sicuro che per il fatto di essere distribuito presso tutti i partecipanti (tutti i partecipanti hanno una copia di ciascuna operazione) è anche in grado di resistere a eventuali manomissioni.

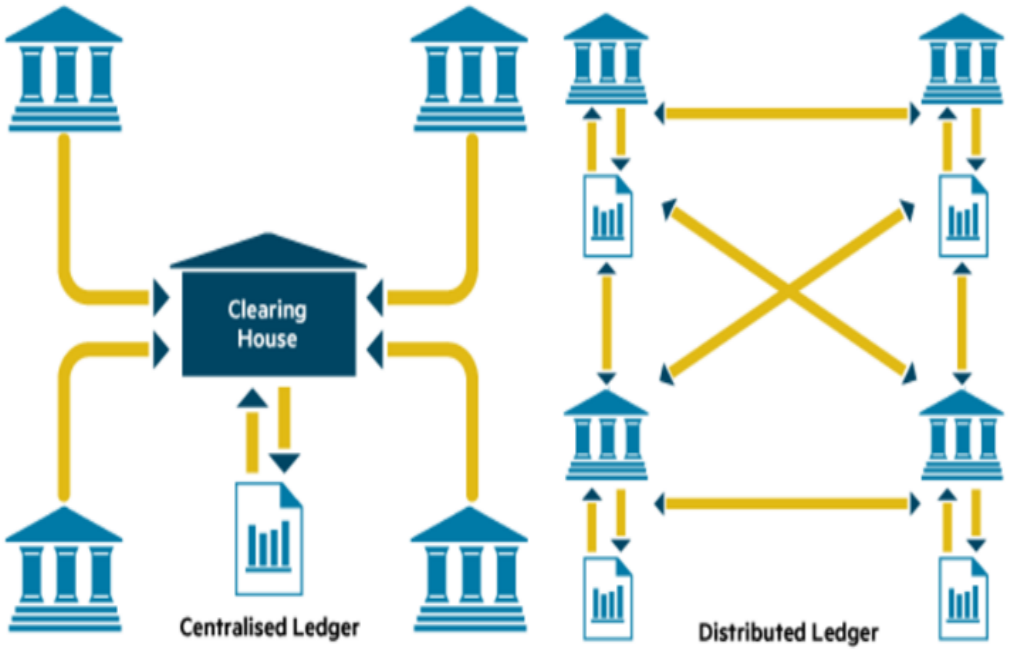

Figura 1.14: Centralised vs Distributed Ledger. [25]

1.6 Blockchain

1.6.1 Caratteristiche e proprietà

La blockchain [12] è una tecnologia che permette la creazione di un grande registro distribuito per la gestione di transazioni condivisibili tra più nodi di una rete. Si tratta di un registro strutturato in blocchi, nei quali sono presenti le transazioni, e sono tra loro collegati in rete in modo che ogni transazione avviata sulla rete deve essere validata dalla rete stessa. La blockchain risulta così costituita da una catena di blocchi.
Ogni nodo è chiamato a vedere, controllare e approvare tutte le transazioni creando una rete che condivide su ciascun nodo l'archivio di tutta la blockchain e dunque di tutti i blocchi con tutte le transazioni.
Ciascun blocco è per l'appunto anche un archivio per tutte le transazioni e per tutto lo

storico di ciascuna transazione, le quali, possono essere modificate solo con l'approvazione dei nodi della rete. Le transazioni possono essere considerate immodificabili (se non attraverso la riproposizione e la "ri"-autorizzazione delle stesse da parte di tutta la rete). Da qui il concetto di immutabilità.

Tali tecnologie sono incluse nella più ampia famiglia delle Distributed Ledger, ossia sistemi che si basano su un registro distribuito. Non è richiesto che i nodi coinvolti conoscano l'identità reciproca o si fidino l'uno dell'altro. Difatti, per garantire la coerenza tra le varie copie, l'aggiunta di un nuovo blocco è globalmente regolata da un protocollo condiviso. Una volta autorizzata l'aggiunta del nuovo blocco, ogni nodo aggiorna la propria copia privata: la natura stessa della struttura dati garantisce l'assenza di una sua manipolazione futura.
Le caratteristiche che accomunano i sistemi sviluppati con le tecnologie Blockchain e Distributed Ledger sono digitalizzazione dei dati, decentralizzazione, disintermediazione, tracciabilità dei trasferimenti, trasparenza/verificabilità, immutabilità del registro e programmabilità dei trasferimenti.
Grazie a tali caratteristiche, la blockchain è considerata pertanto un'alternativa in termini di sicurezza, affidabilità, trasparenza e costi alle banche dati e ai registri gestiti in maniera centralizzata da autorità riconosciute e regolamentate (pubbliche amministrazioni, banche, assicurazioni, intermediari di pagamento, ecc.).

Le principali proprietà della blockchain:

- **Affidabilità**: la blockchain è affidabile. Non essendo governata da una autorità centrale, ma dando a tutti i partecipanti diretti una parte di controllo dell'intera catena, la blockchain diventa un sistema meno centralizzato, meno governabile, e allo stesso tempo molto più sicuro e affidabile, ad esempio da attacchi di malintenzionati. Se infatti soltanto uno dei nodi della catena subisce un attacco e si danneggia, tutti gli altri nodi continueranno comunque a essere attivi e operativi, saldando la catena e non perdendo in questo modo informazioni importanti.

- **Trasparenza**: le transazioni effettuate attraverso la blockchain sono visibili a tutti i partecipanti, garantendo così trasparenza nelle operazioni.

- **Convenienza**: effettuare transazioni attraverso la blockchain è conveniente per tutti i partecipanti, in quanto vengono meno interlocutori di terze parti, necessari in tutte le transazioni convenzionali che avvengono tra due o più parti (ovvero le banche e altri enti simili).

- **Solidità**: le informazioni già inserite nella blockchain non possono essere modificate in alcun modo. In questo modo le informazioni contenute nella blockchain sono tutte più solide e attendibili, proprio per il fatto che non si possono alterare e quindi restano così come sono state inserite la prima volta.

- **Irrevocabilità**: con la blockchain è possibile effettuare transazioni irrevocabili, e allo stesso tempo più facilmente tracciabili. In questo modo si garantisce che le transazioni siano definitive, senza alcuna possibilità di essere modificate o annullate.

- **Digitalità**: con la blockchain tutto diventa virtuale. Grazie alla digitalizzazione, gli ambiti applicativi di questa nuova tecnologia diventano tantissimi.

Talvolta risulta possibile che alcuni nodi della rete producano simultaneamente più blocchi "concorrenti" (ossia collegati a uno stesso blocco già esistente, ma diversi tra loro nel contenuto): ciò dà origine a una biforcazione (fork) nella catena. Il protocollo di aggiornamento specifica quale regola i nodi debbano adottare per selezionare il blocco da accettare tra quelli concorrenti. I blocchi non selezionati per l'inclusione nella catena sono chiamati "blocchi orfani".
In ciascun blocco sono presenti dunque diverse transazioni e anche l'hash del blocco precedente. Nel momento in cui un blocco deve essere validato, viene calcolato l'hash dello stesso contenente transazioni e il rimando al blocco che lo precede. In questo modo si legano tra di loro i vari blocchi andando a creare una catena di blocchi.
La transazione contiene invece informazioni relative all'indirizzo pubblico del ricevente, le caratteristiche della transazione e la firma crittografica che garantisce sicurezza e autenticità della transazione.

Figura 1.15: Un blocco della blockchain contenente diverse transazioni. [26]

La blockchain è da vedere come un registro pubblico e condiviso costituito da una serie di client o di nodi.
La blockchain è organizzata per aggiornarsi automaticamente su ciascuno dei client che partecipano al network. Ogni operazione effettuata deve essere confermata automaticamente da tutti i singoli nodi.

1.6.2 Il ruolo dei Miner

Perché un nuovo blocco di transazioni sia aggiunto alla blockchain è necessario appunto che sia controllato, validato e risolto. Solo con questo passaggio può poi essere aggiunto alla blockchain.
Per effettuare questo passaggio, è necessario che ogni volta che viene composto un blocco venga risolto un complesso problema matematico che richiede un cospicuo impegno anche in termini di potenza e di capacità elaborativa.
Questa operazione viene definita come “mining” ed è svolta dai miner.

Il lavoro del miner è assolutamente fondamentale nell'economia della gestione delle blockchain. Chiunque può diventare un miner e può competere per essere il primo a risolvere il complesso problema matematico legato alla creazione di ogni nuovo blocco di transazioni in modo valido e crittografato che possa essere aggiunto alla blockchain.

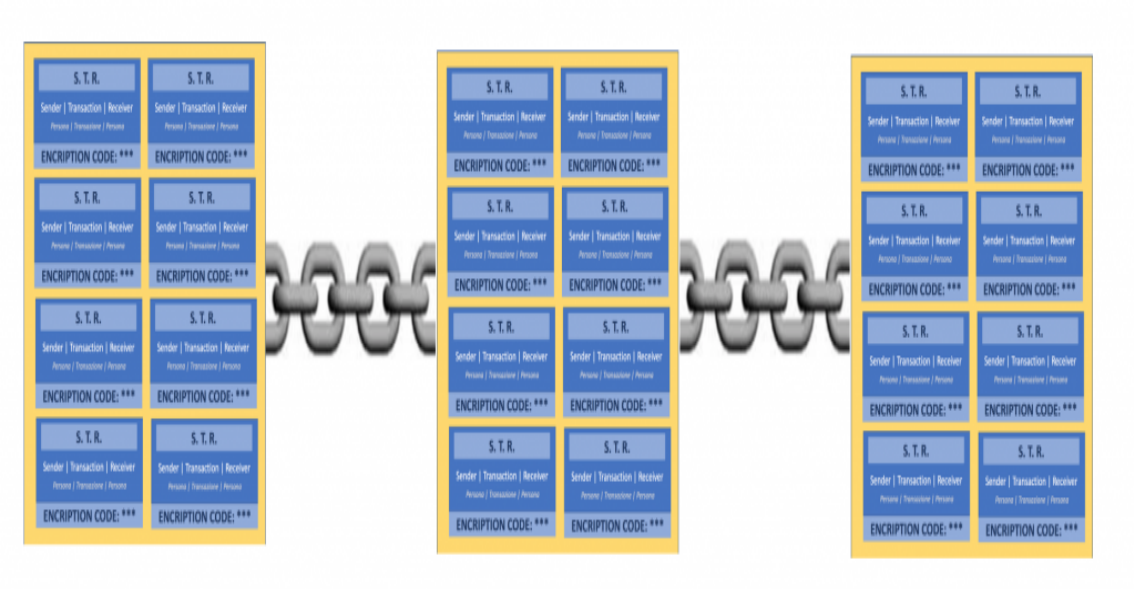

Figura 1.16: La catena di blocchi della blockchain. [26]

1.6.3 La sicurezza della blockchain

Una delle caratteristiche più importanti della blockchain è la sicurezza. Il funzionamento non è garantito da un ente centrale, ma ogni singola transazione è validata dall'interazione di tutti i nodi.
La marca temporale consente di associare una data e un'ora certe e legalmente valide a un documento informatico. In altre parole, la marca temporale consente di definire una validazione temporale che può essere opponibile a terzi.
La marca temporale impedisce anche che l'operazione, una volta eseguita, venga alterata o annullata. La marca temporale o timestamp è costituita da una sequenza specifica di

caratteri che identificano in modo univoco, indelebile e immutabile una data e/o un orario per fissare e accertare l'effettivo avvenimento di un certo evento. La rappresentazione della data è sviluppata in un formato che ne permette la comparazione con altre date e permette di stabilire e definire un ordine temporale. La pratica dell'applicazione di tale marca temporale è detta timestamping.

1.6.4 Consenso distribuito

La tipologia di consenso distribuito può cambiare in base alla blockchain su cui si opera. Se si prende come esempio la blockchain di Bitcoin o quella di Ethereum, il processo di validazione prevede una fase di verifica e di approvazione basata su risorse di calcolo che sono finalizzate alla risoluzione di problemi complessi o puzzle crittografici. Si ha in questo modo un consenso distribuito e non più di un consenso basato su un intermediario terzo o su un ente o istituzione centralizzata.
Coloro che partecipano alla risoluzione del problema e che dunque concorrono alla validazione dei blocchi sono i miner e il loro intervento, che necessita per essere svolto di importanti risorse, viene remunerato attraverso l'emissione di una moneta virtuale o cryptocurrency.

La logica che sta alla base di questo processo parte dal presupposto che per evitare rischi di frodi in particolare da parte di un "nodo" della blockchain è necessario creare degli ostacoli e delle complicazioni su tutto il processo di validazione. Nello specifico, ogni nodo che intende partecipare alla validazione deve anche risolvere un complesso problema nella forma di un puzzle crittografico.
Il puzzle è concepito per mettere in competizione tutti i nodi e tutti contribuiscono alla risoluzione mettendo a disposizione la propria potenza di calcolo. Il nodo che riuscirà a risolvere il puzzle crittografico avrà il diritto di validare il blocco. Proof of Work è il nome di questa prova, prova che devono svolgere i miner della rete, la tipologia di questo consenso distribuito.
Per questo impegno e per questo risultato, il nodo viene appunto remunerato con una

unità di valore che dipende dalla tipologia di blockchain.

1.7 Ethereum

Ethereum [13] è una piattaforma decentralizzata che funziona contemporaneamente su migliaia di dispositivi in tutto il mondo. È una piattaforma che può essere adottata da tutti coloro che desiderano entrare a far parte della rete e che in questo modo avranno a disposizione una soluzione che consente a tutti i partecipanti di disporre di un archivio immutabile e condiviso di tutte le operazioni effettuate nel corso del tempo e che nello stesso tempo è concepita per non poter essere fermata, bloccata o censurata.
Ethereum è progettata per essere adattabile e flessibile e per creare facilmente nuove applicazioni. Ethereum è cioè una Programmable Blockchain che non si limita a mettere a disposizione "operations" predefinite e standardizzate, ma permette agli utenti di creare le proprie "operations". Di fatto è una blockchain platform che permette di dare vita a diverse tipologie di applicazioni blockchain decentralizzate non necessariamente limitate alle sole cryptocurrencies.
In poche parole, l'idea centrale alla base di Ethereum è che gli sviluppatori possono creare e implementare codice che viene eseguito attraverso un network distribuito, invece di esistere su un server centralizzato. Questo significa che, in teoria, queste applicazioni non possono essere bloccate o censurate.

1.7.1 Smart Contract

Attraverso la blockchain Ethereum si possono vincolare le decisioni prese consensualmente nel network, piuttosto che subordinarle a un ente centrale che autorizzi tutte le attività, come per esempio gli smart contracts. Nonostante si chiamino contratti, non devono essere compilati o riempiti.
Gli smart contracts servono a eseguire porzioni di codice che vengono interessate da una transazione; gli smart contracts esercitano un controllo diretto sul proprio conto di

valuta ether (la criptovaluta di Ethereum) e sul proprio valore: l'obiettivo è conservare traccia delle variabili in gioco, per garantire tracciabilità e trasparenza. Quando si parla di transazioni ci si riferisce a un pacchetto di dati contenenti un messaggio diretto a un account esterno. All'interno di una transazione sono presenti:

- il nominativo del destinatario del messaggio;
- la firma del mittente;
- la quantità di Ether oggetto della transazione;
- un valore che rappresenta il numero massimo di operazioni che possono essere eseguite nella transazione;
- un valore pari alla commissione pagata dal mittente per lo step computazionale.

Dunque, all'interno della rete è necessario pagare, in valuta Ether, lo stesso network per poter usufruire della potenza computazionale. Ether dunque è la criptovaluta necessaria per effettuare le transazioni, ossia inviare e ricevere pagamenti e far circolare gli smart contracts.
Il contratto tuttavia, a differenza di un accordo scritto, prevede delle variabili; significa che si comporta autonomamente perché reagisce agli input rispondendo con output consequenziali; Tuttavia, lo smart contract, se è vero che prevede delle variabili, in ogni caso rispetta e vincola i contraenti a regole condivise, a cui non è possibile derogare.
I contratti all'interno della blockchain Ethereum permettono di effettuare tutta una serie di operazioni importanti, come la registrazione di un dominio, l'avvio di un crowdfunding e la tutela della proprietà intellettuale.

1.7.2 Ethereum Virtual Machine EVM

Il motore di Ethereum è rappresentato dalla Ethereum Virtual Machine (EVM) che rappresenta di fatto l'ambiente di runtime per lo sviluppo e la gestione di smart contracts in Ethereum.

Figura 1.17: Ecosistema Ethereum. [27]

EVM opera in modo protetto, ovvero risulta completamente separato dalla rete.
Il codice gestito dalla virtual machine non ha accesso alla rete e gli stessi smart contracts generati sono indipendenti e separati da altri smart contracts. Sono cioè disponibili sulla blockchain in EVM bytecode (un Ethereum-specific binary format), sono scritti in Ethereum high level language, trasformati in byte code con un compiler EVM e caricati sulla blockchain con un client Ethereum.

1.7.3 Applicazioni decentralizzate (DApps)

Le applicazioni decentralizzate (DApp) sono applicazioni che vengono eseguite su un sistema informatico distribuito, ovvero una rete blockchain. Sebbene esistano vari modi

per definire una DApp, di solito vengono descritti come applicazioni che hanno le seguenti caratteristiche:

- **Open Source**: Il codice sorgente è intenzionalmente reso disponibile al pubblico, il che significa che chiunque è in grado di verificare, utilizzare, copiare e modificare il codice.

- **Decentralizzato**: poiché le DApp vengono eseguite su reti blockchain, non sono controllate da una singola entità o autorità. Invece, sono mantenuti da più utenti (nodi).

- **Crittograficamente sicuro**: l'applicazione è protetta da crittografia, il che significa che tutti i dati vengono registrati e mantenuti in una blockchain pubblica. Non esiste un point of failure.

Il vantaggio principale di scegliere una DApp rispetto a un'app tradizionale è che quest'ultima utilizza un'architettura centralizzata memorizzando i propri dati su server controllati da una singola entità. Ciò significa che è suscettibile a problemi tecnici e attacchi dannosi.
Un server centralizzato compromesso potrebbe interrompere l'intera rete dell'applicazione, rendendola temporaneamente o permanentemente inutilizzabile. Oltre a questo, i sistemi centralizzati subiscono spesso fughe di dati o furti, mettendo a rischio le aziende e i singoli utenti.
Esiste una grande varietà di DApp, con diversi casi d'uso. Possono includere giochi, piattaforme di social media, portafogli di criptovaluta e applicazioni finanziarie (DeFi). Le applicazioni decentralizzate alimentano la propria attività attraverso un sistema tokenizzato (token digitali creati attraverso l'utilizzo di smart contracts).
I token possono essere specifici per una particolare DApp, oppure possono essere nativi della blockchain che ospita la DApp.

1.8 IOTA

1.8.1 DAG e il Tangle

Successivamente alla blockchain altre forme di registro distribuito hanno preso forma. Tra queste si può individuare il tangle.

Il tangle [14] si basa su una struttura dati denominata Directed Acyclic Graph (DAG). Questa struttura, come dice il nome stesso, ha una forma di grafo in cui sono presenti nodi collegati l'uno con l'altro. I collegamenti tra i nodi non presentano delle forme cicliche ma ogni collegamento tra di essi ha una direzione, per questo l'aggettivo directed. A differenza della blockchain in cui le transazioni sono contenute in dei blocchi, nel tangle le transazioni rappresentano i nodi stessi della struttura.
Per rendere questo registro immutabile, ogni nuova transazione sul tangle, denominata *tip*, è referenziata a due precedenti transazioni non ancora validate all'interno del grafo. La referenza si dice diretta se esiste un arco che collega le due transazioni, mentre è indiretta se esiste un percorso che permetta di arrivare da una transazione all'altra. Il modo in cui una nuova transazione sceglie due tip non è casuale, ma è determinata da un algoritmo definito Markov Chain Monte Carlo (MCMC). Questo permette a una transazione di approvare e referenziare transazioni relativamente giovani, quindi verso il fondo del tangle.

1.8.2 Caratteristiche e proprietà

IOTA [15] è un progetto nato a fine 2015 da Sergey Ivancheglo, Dominik Schiener, David Sonstebo e Serguei Popov, allo scopo di fornire una valida alternativa al registro blockchain nel campo dell'IoT. IOTA consiste in un registro distribuito permissionless e dalla criptovaluta omonima.
A differenza delle altre criptovalute, IOTA non è solamente un ambiente in cui scambiare valuta virtuale, infatti, il registro permette anche la presenza di transazioni zero-value, in cui sono i dati la componente principale di esse.

Il registro distribuito su cui si basa IOTA è il tangle. Sul tangle vengono inoltrate e raccolte tutte le transazioni del network. Queste vengono ispezionate da una serie di attori per verificarne l'integrità e l'autenticità. La rete IOTA è formata da un network di nodi, i quali permettono ai client di collegarsi alla rete, leggere informazioni e scrivere sul tangle. Quando un nodo riceve una transazione, esso cercherà di inoltrarla ai suoi vicini, così che questi possano convalidare la transazione e aggiungerla alla loro copia del registro.

La prima transazione avvenuta nel tangle è definita genesis transaction ed è l'unica che ha generato token all'interno di IOTA. I token IOTA infatti sono fissi, pari a 2.779.530.283.277.761 e non sono previsti reward di alcun tipo. Per poter emetter una transazione sul tangle, un nodo IOTA deve:

- Scegliere due tip da approvare, secondo l'algoritmo MCMC.
- Accertarsi che le due transazioni non siano in conflitto e non approvare quelle che lo sono.
- Una volta che un nodo ha accertato la validità delle due transazioni, svolge una Proof-of-Work simile a quella di Bitcoin ma molto meno dispendiosa.

Funzione molto importante all'interno di questo registro distribuito la svolge una stringa denominata seed. Il seed è la chiave principale per un utente ed è fondamentale non rivelarla. Esso infatti permette di creare nuovi address dai quali è possibile inviare transazioni. Inoltre, permette di dimostrare, per il detentore del seed, la proprietà delle stesse transazioni. Ogni utente è responsabile della creazione del proprio seed.

Il protocollo IOTA cerca di far fronte alle limitazioni che la blockchain ha, proponendo un nuovo modello di registro distribuito che non prevede la presenza di una catena di blocchi. Il protocollo IOTA prevede che per ogni transazione immagazzinata nel tangle, questa ne approvi altre due già presenti. In questo modo, in un modello ideale, più transazioni vengono immagazzinate e più transazioni vengono confermate.
Il modello blockchain invece presenta un collo di bottiglia nel momento in cui i nodi vanno a inserire le transazioni all'interno di un nuovo blocco, il quale ha dimensioni finite e non può contenere tutte le transazioni effettuate.

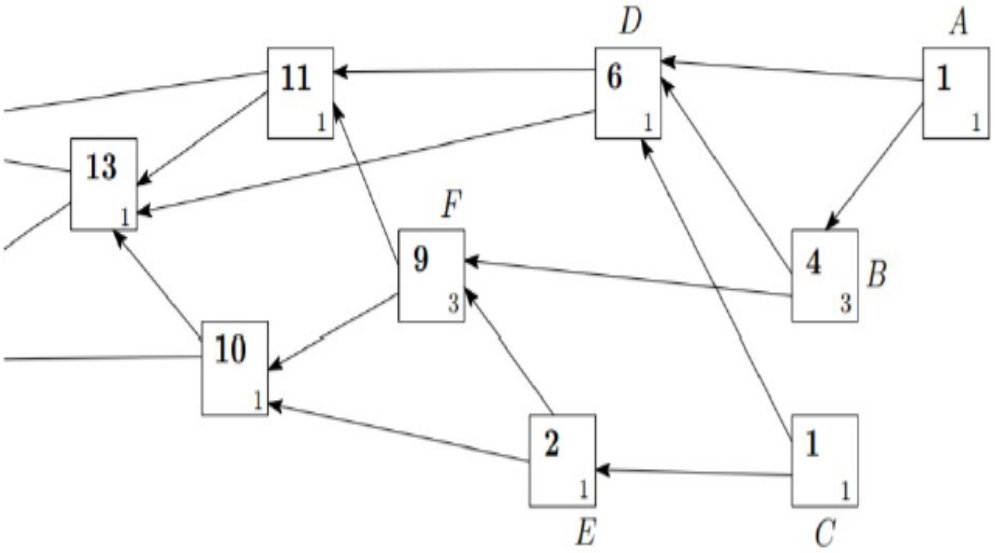

Figura 1.18: Transazioni sul Tangle. [14]

Questo problema si porta dietro anche il problema dell'aumento delle fees. Infatti, perché una transazione sia immagazzinata in un blocco, gli utenti sono costretti a pagare alte fees, essendo questo il parametro che i miner guardano quando decidono di quali transazioni prendersi carico.

IOTA è, in teoria, infinitamente scalabile, perché, all'aumentare del numero di transazioni, aumenta il numero di transazioni validate (ogni nuova transazione ne valida 2 precedenti). Nella blockchain invece, l'inserimento a velocità costante di un blocco alla blockchain è di fatto un collo di bottiglia per le prestazioni della rete.

1.8.3 Peso di una transazione

Si definisce *weight* il peso di una transazione. Questo parametro assume valori nell'ordine di 3^n, dove n è un valore positivo proporzionale alla quantità di lavoro che un nodo svolge per immettere una transazione nel registro. Ogni transazione possiede quindi un peso e l'idea generale è quella che più è alto questo valore e più la transazione è importante.

Si definisce inoltre il parametro *cumulative weight* di una transazione, dato dalla somma dei pesi delle transazioni che approvano direttamente o indirettamente la stessa.

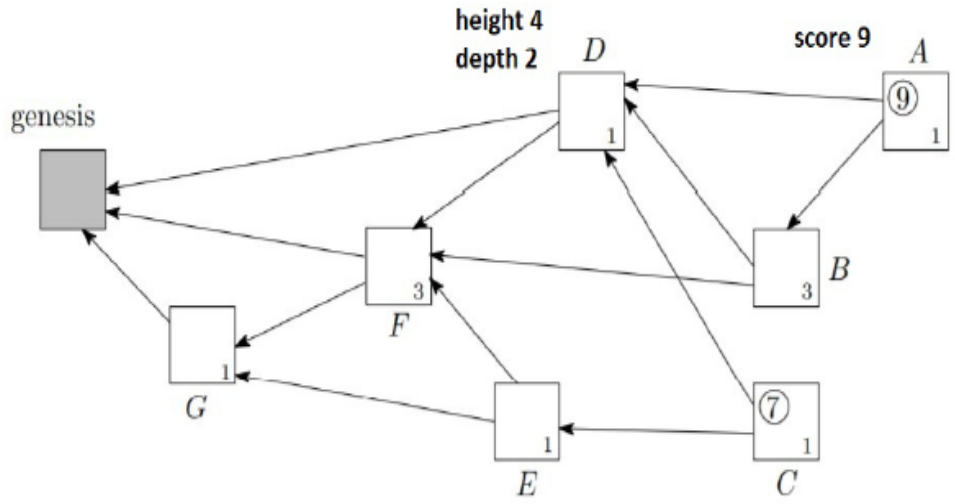

Figura 1.19: Parametri di una transazione. [14]

Per una transazione si possono considerare altri tre parametri: *height*, *depth* e *score*. Il primo rappresenta il percorso più lungo per arrivare al genesis, mentre il secondo rappresenta il valore contrario, ovvero il percorso a ritroso per raggiungere una tip. Lo score di una transazione invece, rappresenta la somma dei pesi di tutte le transazioni approvate da questa, più il proprio peso.

1.8.4 Il sistema ternario

In IOTA, i dati sono rappresentati seguendo il sistema numerale ternario. L'aritmetica ternaria, a differenza di quella binaria implementata in bit, prevede come unità atomica il trit. Questa è l'unità più piccola nel sistema e può assumere valori di -1, 0, 1.
È possibile raggruppare trits in tryte, analogamente a quanto si fa con bit e byte. Un

tryte è composto da 3 trits, da qui si evince che un tryte può assumere 3^3 possibili valori, ovvero 27. Per rendere i trytes di più semplice lettura, essi sono rappresentati da 27 caratteri, composti dal numero 9 e da tutte le lettere dell'alfabeto inglese in maiuscolo, dalla A alla Z. IOTA afferma che il sistema ternario è di maggiore efficienza perché permette di rappresentare i dati in 3 stati piuttosto che 2.

Capitolo 2

Lavori correlati

2.1 Sistemi di Trasporto Intelligenti (ITS)

2.1.1 Introduzione

Riuscire ad avere accesso a grandi quantità di dati e nello stesso tempo in maniera agevole, può avere applicazioni in diversi settori e differenti casi d'uso. Essendo alta e sempre più specifica la richiesta, si stanno consolidando dei mercati nei quali i dati rappresentano la merce più desiderata. Questi, ormai, se ne possono trovare di tutti i tipi grazie alle numerose piattaforme web e ai molteplici social che dominano l'internet in questi tempi. La possibilità di raccogliere, organizzare e poi vendere pacchetti di dati al miglior offerente risulta un'attività in espansione grazie anche all'aumento delle richieste. Risulta però importante in questo mercato la capacità di entrare in possesso di dati di cui le fonti sono attendibili e di sicuro affidamento. Ecco perché è importante utilizzare tecnologie come le DLT che garantiscono che i dati raccolti non possono essere soggetti a modifiche di nessun tipo.
Il processo che permette di entrare in possesso di dati specifici può essere facilitato utilizzando delle DHT che grazie all'ausilio di ricerche basate su keywords associate al dato ricercato rende il tutto più immediato. Un caso d'uso che di seguito viene approfondito e che utilizza diverse tecnologie e diverse tecniche per garantire sicurezza e immutabilità ai dati è quello relativo allo sviluppo di sistemi di trasporto intelligenti (ITS).

2.1.2 Architettura del sistema

Come già accennato, il lavoro che è stato elaborato in questa tesi è un tassello che rientra in una ricerca [1] più ampia. L'idea è quella di realizzare un'architettura di sistema per promuovere lo sviluppo di sistemi di trasporto intelligenti (ITS) utilizzando registri distribuiti e tecnologie correlate. Grazie a questi sistemi diventa possibile creare, archiviare e condividere i dati generati dagli utenti attraverso i sensori sui propri dispositivi o veicoli mentre sono in movimento.

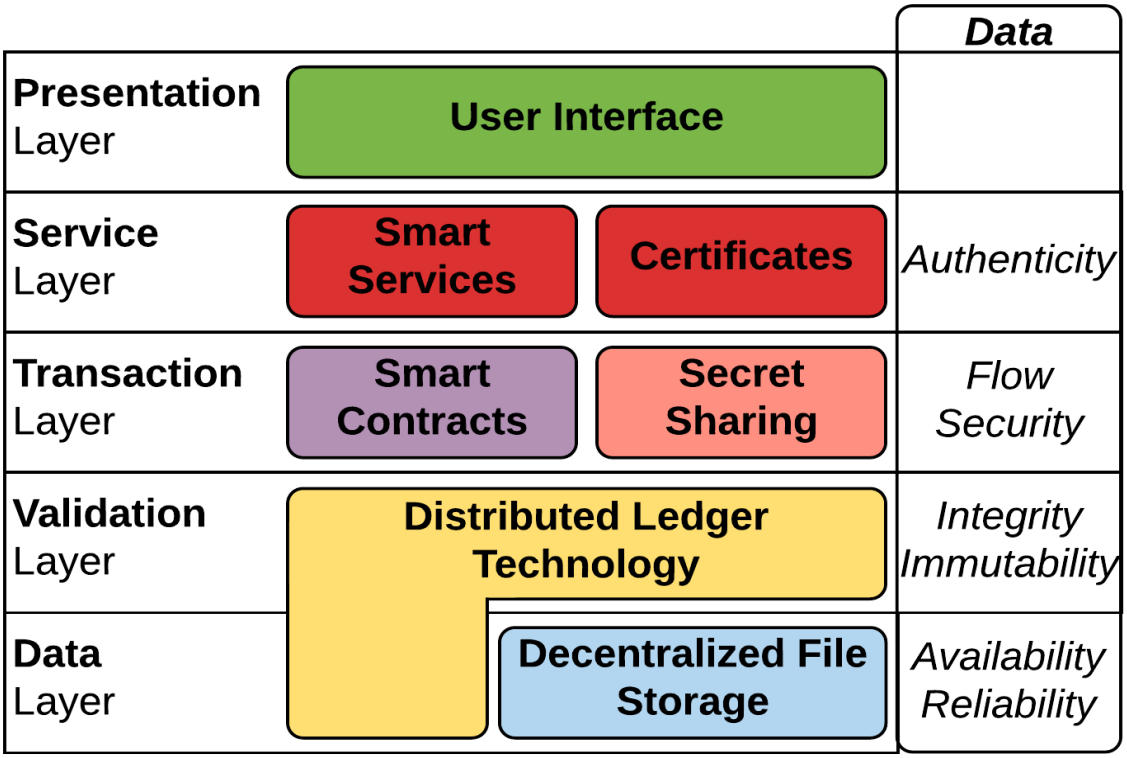

Figura 2.1: Architettura a strati del sistema ITS. [1]

L'architettura di sistema che viene proposta si basa sull'aggregazione di diverse tecnologie distribuite che consentono la raccolta e la condivisione di dati crowd-sensed e gestiti dall'utente.
Gli smart services rafforzano la capacità dell'architettura di ottenere dati corretti e veri-

ficati su cui poi si possono creare applicazioni. Una rappresentazione a strati dell'architettura del sistema è fornita nella Figura 2.1. Nello specifico:

- **Data Layer**: il primo livello (il più basso) comprende quelle tecnologie che forniscono archiviazione, disponibilità e affidabilità dei dati attraverso la replica, ovvero DLT e DFS.

- **Validation Layer**: l'integrità dei dati, rilevati dagli utenti e archiviati nel data layer, deve essere garantita e verificata. A tal fine, il livello di convalida impiega DLT.

- **Transaction Layer**: l'utilizzo dei dati è autorizzato solo agli utenti che hanno avuto delle autorizzazioni. Il controllo degli accessi viene eseguito tramite smart contracts e tecniche di condivisione segreta.

- **Service Layer**: l'accesso ai dati condivisi consente di creare smart services, utili ad altri utenti. È importante garantire ai consumatori di dati che questi siano affidabili e veritieri, ad es. i dati sono stati generati in una determinata posizione e misurati in condizioni adeguate. I certificati possono garantire questa fiducia. Utilizzo di DHT per la ricerca, basata su keywords, di dati.

- **Presentation Layer**: gli utenti possono interagire con i servizi in diversi modi, in base alla specifica applicazione. Questo livello è dedicato all'implementazione delle interfacce tra l'architettura del sistema e il mondo esterno.

L'architettura si basa fondamentalmente sui dati raccolti dai sensori posti nei veicoli o nelle Application Unit degli utenti. L'AU dell'utente, infatti, rappresenta un nodo ITS controllato dall'utente stesso e questo gli permette di condividere i suoi dati e di utilizzare i vari smart services: dato il suo ruolo principale, è posto al centro del diagramma di Figura 2.2.

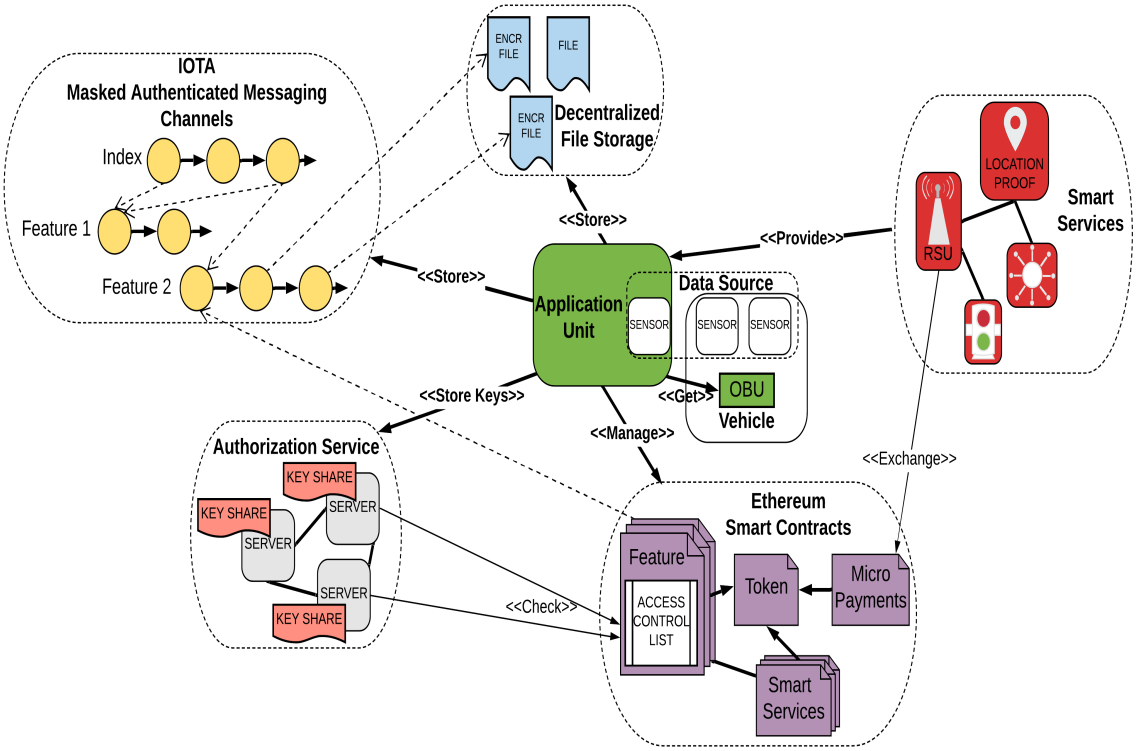

Figura 2.2: Architettura di sistema ITS. [1]

2.1.2.1 Data Layer

In questo primo livello i dati vengono raccolti dai sensori interni del veicolo e rappresentano le principali fonti di dati rilevati nell'ITS. I sensori sono controllati attraverso l'AU (Application Unit) dell'utente o tramite l'OBU (On Board Unit) e i dati raccolti vengono successivamente organizzati e raffinati per essere archiviati nel sistema distribuito.
I dati possono essere archiviati in due diverse tecnologie: un DFS o un DLT. Il fattore discriminante per la selezione della tecnologia specifica è la dimensione dei dati. L'archiviazione dei dati in un Decentralized File System in genere richiede latenze inferiori rispetto a quelle ottenibili tramite DLT.
Tuttavia, la convalida si ottiene tramite la pubblicazione dei dati (o del digest dei dati) in una DLT (vedere Validation Layer). Pertanto, con l'obiettivo di velocizzare il processo di pubblicazione, adottiamo le seguenti euristiche:

- I dati che hanno una dimensione grande, superiore a un singolo valore rilevato, vengono memorizzati in un DFS e referenziati nella DLT tramite il loro ID, ad es. digest.

- I file di piccole dimensioni (la cui dimensione è paragonabile alla dimensione del loro digest) vengono memorizzati direttamente nelle DLT.

Dopo che un file viene pubblicato nel DFS, il riferimento restituito può essere utilizzato per recuperarlo nella rete.

2.1.2.2 Validation Layer

Uno degli obiettivi dell'architettura proposta è quello di dare la proprietà dei dati agli utenti che li producono, senza che l'intera raccolta dei dati rilevati sia archiviata in un servizio di archiviazione dati centralizzato.
Le DLT consentono di evitare tutti gli inconvenienti tipici degli approcci basati su server (ad esempio censura, single point of failure) e offrono funzionalità come l'immutabilità dei dati, la verificabilità e, soprattutto, la tracciabilità. Durante l'implementazione dell'architettura di sistema, è stato deciso di utilizzare IOTA come DLT. La logica alla base di questa scelta è data dalle sue promesse in termini di prestazioni e alla presenza del Masked Authenticated Messaging (MAM), un metodo di gestione dei dati che si occupa del requisito principale della privacy dei dati.
Come già accennato, in IOTA il Tangle memorizza informazioni immutabili che non possono essere censurate / rimosse. Il Tangle è un registro dati pubblico, accessibile a chiunque. Pertanto, al fine di offuscare i dati e renderli accessibili solo alle entità autorizzate, i canali MAM vengono utilizzati per archiviare dati crittografati, fornendo l'accesso solo agli utenti idonei.
I dati raccolti dai sensori sono organizzati in feature, cioè un particolare tipo di dati (ad esempio una temperatura rilevata) o un punto dati come la geolocalizzazione (ad esempio i dati sono stati prodotti a Copacabana, Rio De Janeiro, Brasile) o la velocità del veicolo.

2.1.2.3 Transaction Layer

Il livello delle transazioni è dedicato a consentire agli utenti di condividere e scambiare i propri dati. È composto da un insieme di smart contract Ethereum e da un servizio di autorizzazione, che funge da intermediario per l'utente. Gli smart contracts sono responsabili della concessione dell'accesso a determinati dati.
L'accesso ad un dato specifico, infatti, è acquistabile tramite modalità di smart contract dedicato. In Ethereum, tali metodi sono funzioni pagabili che attuano transazioni monetarie. A causa della presenza di smart contracts, non sono necessarie interazioni dirette tra il proprietario dei dati e gli utenti interessati ai suoi dati. Il servizio di autorizzazione è incaricato di far rispettare i diritti di accesso e di rilasciare le chiavi per accedere ai dati.
Un consumatore di dati può inviare una richiesta al servizio di autorizzazione, al fine di ottenere le chiavi segrete necessarie per decrittografare i messaggi nei canali MAM e negli oggetti su piattaforme DFS come IPFS. Su richiesta dell'utente, il servizio di autorizzazione può verificare l'idoneità, attraverso l'interazione con lo smart contract e successivamente fornisce all'utente le relative chiavi di accesso. In particolare, per ogni messaggio MAM e il relativo oggetto IPFS (se disponibile), è presente una chiave, che viene utilizzata per crittografare i dati prodotti.
La presenza di un servizio di autenticazione è necessaria per due motivi:
i) non è possibile presumere che ogni utente sia sempre attivo per ricevere richieste di accesso; ii) non è possibile che i contratti intelligenti possano essere autonomi nel rilasciare chiavi di decrittazione o decrittografare messaggi, a causa del fatto che il loro calcolo è pubblico.

2.1.2.4 Service Layer

Questo livello include tutti i servizi ITS decentralizzati messi a disposizione degli utenti. Sfruttano tutte le funzionalità e i dati prodotti nei livelli sottostanti. Gli smart services si basano sulle funzionalità affidabili fornite dagli smart contracts.
I servizi utilizzano dati rilevati dalla folla, la cui affidabilità si basa sull'utilizzo di certi-

ficati, ovvero prove sulla validità dei dati. Una delle principali preoccupazioni, relativa al recupero dei dati crowdsensed, si riferisce alla veridicità e accuratezza dei dati. È un dato di fatto che i dati sono spesso incerti, imprecisi e di difficile fiducia. Esistono una serie di problemi correlati, che vanno dai problemi relativi alla precisione, alla presenza di rumore nel rilevamento, fino al comportamento volontariamente dannoso dell'utente, come la falsificazione e così via.
L'importante è garantire che i dati rilevati siano aggiunti in modo sicuro a un registro decentralizzato, evitando così che un'entità esterna possa manometterli. Infatti, una volta aggiunti a una DLT, i dati non possono essere modificati, a causa delle caratteristiche di immutabilità delle DLT. Pertanto, il momento in cui i dati possono essere modificati è tra la sua generazione e il suo inserimento nel registro distribuito. Per far fronte a questo, in questo contesto la fiducia può essere ottenuta tramite autorità di terze parti e meccanismi di prova.
In particolare, i certificati vengono utilizzati come prove di una determinata proprietà dello stato dell'utente nello spazio e nel tempo. Tali certificati possono essere allegati insieme ai dati.
Una volta che i dati sono stati aggiunti e memorizzati in una DLT, è importante riuscirne a ricavare grandi quantità aventi tra loro delle caratteristiche in comune. È possibile che diversi consumatori di dati vogliano analizzare determinati set aventi determinati target in comune. Il lavoro svolto in questa tesi consiste nella progettazione e implementazione di una Distributed Hash Table (DHT), in cui i dati una volta che sono stati acquisiti e registrati in dei registri distribuiti, vengono caricati nella DHT. Questo strumento facilita la ricerca in quanto utilizza un sistema basato su keywords. In questo modo si semplifica notevolmente il processo di ricerca aiutando chiunque li richiede. Riuscire ad entrare in possesso di grandi quantità di dati in maniera facile utilizzando, appunto, delle keywords specifiche per tipologie specifiche, aiuta i vari utenti nello scambio e porta alla creazione di un mercato che beneficia di questo strumento in quanto ne migliora gli automatismi. Il mercato di dati che questa ricerca si immagina di creare, consente di connettere fornitori e consumatori, garantendo loro alta qualità, coerenza e sicurezza nel servizio.
I fornitori di dati sono riconosciuti come proprietari nel mercato e ricevono vantaggi (per lo più economici) in cambio dei set e dei flussi di dati che forniscono. I consumatori

pagano per i dati che acquisiscono e, in cambio, possono fornire nuove informazioni al mercato.

Gli smart contracts (ad esempio i contratti di funzionalità) possono automatizzare la negoziazione tra fornitori e consumatori, fornendo vantaggi per entrambe le parti. La DHT proposta ha il compito di aiutare nella ricerca di set di dati specifici, aiutando qualsiasi attore che intenda usufruirne. In questo modo si facilita anche lo sviluppo del mercato stesso in quanto si rende più facile l'accesso ai dati.

Capitolo 3

Hypeercube

3.1 Progettazione

3.1.1 Analisi del problema

Hypeercube nasce come una DHT avente come obiettivo quello di riuscire a facilitare la ricerca di grandi quantità di dati utilizzando delle keywords specifiche. Tra le tante tipologie di dati che è possibile condividere grazie all'uso delle DLT, i dati raccolti dai sensori dei veicoli rappresentano le principali fonti rilevate nell'ITS. Una volta raccolti sono archiviati o referenziati in una DLT. La DLT scelta è stata IOTA perché permette di avere elevate prestazioni e ottime capacità di gestione dei dati grazie al protocollo MAM (Masked Authenticated Messaging). Il protocollo MAM permette la creazione e l'iscrizione a canali che contengono dei flussi di dati, i quali, confluiscono nel Tangle. Per ottenere delle informazioni che provengono da un canale specifico, è necessario conoscere esattamente la sua Root, cioè il suo indirizzo.
Conoscere un indirizzo preciso di un canale risulta molto limitante per riuscire ad ottenere quelle informazioni presenti in esso. Una funzionalità messa a disposizione dal registro IOTA è quella di creare transazioni e messaggi MAM arricchiti con un TAG.
Questo parametro permette di accorpare quelle transazioni che hanno, appunto, un TAG in comune. Questo meccanismo aiuta molto nella ricerca di informazioni specifiche ma, d'altra parte, presenta dei problemi.

Nel caso in cui si volessero alterare delle informazioni specifiche, un qualsiasi utente malintenzionato potrebbe creare delle transazioni fasulle andando ad inserire un TAG corrispondente ad un set di dati che si vuole alterare. I risultati che si otterrebbero da una ricerca sarebbero cosi distorti.
È importante quindi riuscire ad ottenere le informazioni dal Tangle andando ad utilizzare uno strumento che permette di applicare dei filtri e che consente di avere una sicurezza riguardo la veridicità in quei dati che si vogliono ricercare. Questo strumento che permette di ottenere informazioni filtrate e sicure è rappresentato da una Distributed Hash Table. Hypeercube è la DHT implementata e ottimizzata per la ricerca keywords-based. La caratteristica peculiare della rete della DHT è quella di avere una struttura ad ipercubo. L'idea di questa struttura ha origine in questi lavori.[2] [3]
La rete della DHT sarà essenzialmente una struttura overlay ad ipercubo, nella quale ogni nodo indicizzerà oggetti che rappresentano specifici indirizzi MAM. Questi indirizzi contengono le informazioni che sono state raccolte, le quali sono raggruppate in base a determinati TAG.
Sviluppando la struttura su un modello ad ipercubo, ogni singolo oggetto sarà indicizzato da un solo nodo e, il nodo in questione, sarà facile determinarlo grazie alle keywords associate. L'idea è quella di mappare ogni oggetto in un vettore a *r-bit*, in accordo con il suo set di keywords, e considerare questi vettori a *r-bit* come punti, nodi, di un ipercubo a *r-dimensioni*.
Questo tipo di struttura garantisce che oggetti contenuti nella rete aventi keywords simili, siano gestiti da nodi vicini tra loro. Aspetto molto importante per questa tipologia di struttura, appositamente progettata per cercare di migliorare i meccanismi di routing in termini di efficienza.

3.1.2 Struttura ad ipercubo

Per ipercubo si vuole intendere quella forma geometrica regolare rappresentata in uno spazio di quattro o più dimensioni. L'ipercubo costituisce la struttura della DHT, può essere rappresentato da un grafo composto da 2^r nodi.

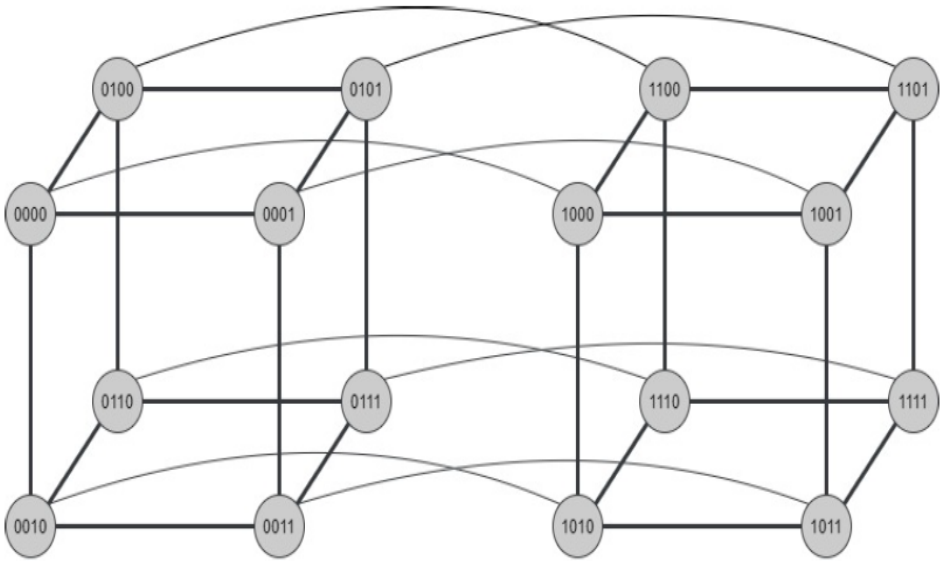

Figura 3.1: Rappresentazione di un ipercubo. [2]

Con r si intende la dimensione dell'ipercubo e anche la dimensione del vettore binario con il quale viene rappresentato ogni nodo in questo tipo di struttura.
I nodi all'interno della rete vengono identificati attraverso la loro numerazione in formato binario in base alla dimensione dell'ipercubo.
Per esempio, data una dimensione pari a quattro, ogni nodo sarà costituito da un vettore di quattro bit. Si definisce $H_r(V,E)$ un ipercubo di r dimensioni, dove V rappresenta l'insieme dei nodi ed E l'insieme degli archi che li collegano. All'interno di questa struttura i nodi sono collegati tra di loro seguendo una regola, cioè quella che tra un nodo ed un altro ci deve essere una differenza di uno e un solo bit. Infatti, un arco *edge(u,v)* collega u con v se, e solo se, u differisce da v di uno e un solo bit.
Prendendo come esempio l'ipercubo a quattro dimensioni rappresentato nella Figura 3.1, ogni nodo è rappresentato da una stringa di 4 bit ed è presente un collegamento fra due nodi solamente se questi differiscono di un solo bit. Se prendiamo il nodo u = 1010 e il nodo v = 1011, si nota che entrambi fanno parte del vicinato dell'altro in quanto distanti di un solo bit all'interno dell'ipercubo. Tra di loro c'è, quindi, un collegamento che li lega. Si definiscono inoltre i due insiemi *One*(u) e *Zero*(u) riguardanti il nodo u.

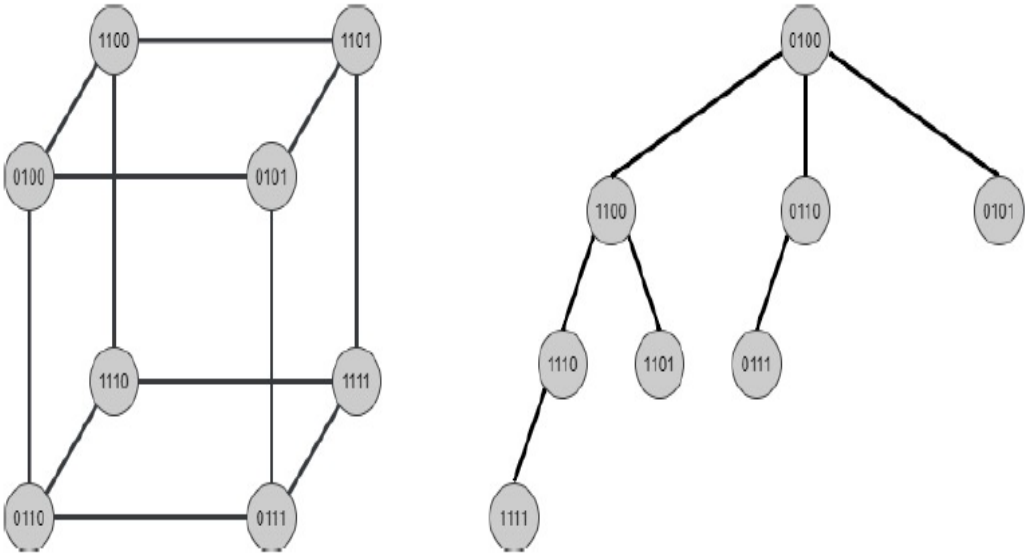

Figura 3.2: Rappresentazione di un sub-ipercubo e il relativo sottografo. [2]

One(u) = { i | u[i] = 1, 0≤i≤r-1} e Zero(u) = { i | u[i] = 0, 0≤i≤r-1}. Rispettivamente One(u) rappresenta le posizioni dei bit con valore 1 del nodo u, mentre Zero(u) le posizioni dei bit con valore 0. Con riferimento ai valori di u e v, il nodo u = 1010 ha un valore di One(u) = {1,3} e Zero(u) = {0,2} mentre il nodo v = 1011 ha un valore di One(u) = {0,1,3} e Zero(u) = {2}.

Se si prende in considerazione un singolo nodo, si può definire un sub-ipercubo indotto dal nodo in questione. Questo sub-ipercubo H_r(u) non è altro che il sottografo composto dai nodi dell'ipercubo principale H_r che sono esclusivamente contenuti in u e da archi che collegano due nodi esclusivamente presenti nel sottografo.
Per capire quanto distanti siano due nodi all'interno della rete, si può utilizzare la distanza di Hamming per calcolarla.
La distanza di Hamming tra due stringhe u = $\{u_i,u_{i-1},\ldots,u_0\}$ e v = $\{v_i,v_{i-1},\ldots,v_0\}$ con $0 \leq i \leq r-1$, è il numero di posizioni i in cui $u_i \neq v_i$.
La distanza di Hamming tra due nodi u e v è data dalla differenza nei valori pari ad 1 in riferimento alle posizioni dell'identificativo binario dei suddetti nodi in seguito ad

un'operazione di XOR. Per fare un esempio, dato il nodo u = 1010 e il nodo v = 1011, calcolando la distanza di Hamming si può notare che la differenza di bit uguali ad 1 dopo l'operazione di XOR è pari ad 1.
Questo significa che i due nodi sono distanti di una sola unità e quindi sono entrambi vicini l'uno con l'altro.
Un aspetto importante da considerare è il legame tra la rete e la struttura ad ipercubo. Il funzionamento di questa particolare struttura prevede che i nodi della rete debbano essere in numero uguale al numero di nodi previsti per l'ipercubo.
La rete di una DHT può essere costituita da un numero di nodi che non per forza sono in numero uguale al numero previsto per l'ipercubo. Questo significa che i nodi logici, quelli dell'ipercubo, potrebbero essere di più dei nodi realmente presenti nella DHT, i cosiddetti nodi fisici. Sotto certi aspetti è possibile avere a che fare con una situazione inversa.
Nella situazione in cui ci siano più nodi logici che fisici, bisognerebbe mappare i nodi dell'ipercubo a quelli della rete della DHT, in uno scenario in cui un nodo fisico si occuperebbe di più nodi logici. Nella situazione inversa in cui i nodi fisici sono di più dei nodi logici, questi si comporteranno alla stessa maniera degli altri nodi andando a creare una situazione di ridondanza delle informazioni archiviate. Infatti, un caso reale consiste in un misto tra le due situazioni; conviene disporre di più nodi logici in modo tale da avere un numero più elevato di set di keywords nella rete su cui basare le ricerche. Non conviene associare esclusivamente un solo nodo fisico ad uno o più di quelli logici perchè se il nodo in questione andasse offline si perderebbero i riferimenti alle informazioni archiviate nei nodi logici associati. La combinazione ideale è quella di associare a un nodo fisico diversi nodi logici e far si che alcuni di questi nodi logici siano mantenuti da altri nodi fisici. Si verrebbero a creare tante intersezioni tra queste due categorie di nodi e in questo modo si avrebbe una ridondanza maggiore di informazioni che permetterebbe di evitare, nel momento in cui un nodo andasse offline, di non perdere le informazioni associate ad esso.

3.1.3 Operazioni sull'ipercubo

L'importanza di questa DHT è data dal fatto di poter cercare, in maniera semplificata, dati archiviati su una DLT come IOTA utilizzando delle keywords specifiche.
Come sappiamo, le operazioni che si possono effettuare, oltre alla ricerca, sono quelle relative all'inserimento e all'eliminazione di oggetti nella rete. Essendo uno strumento che rientra in una architettura più ampia, gli oggetti presenti all'interno della rete provengono dalla DLT nella quale sono archiviati. Le operazioni di inserimento e di cancellazione, quindi, non sono operazioni possibili da effettuare direttamente su questo strumento in quanto i dati sono strettamente collegati alla DLT che li raccoglie, li archivia e li gestisce. Il focus principale di questo strumento è la ricerca.
Innanzitutto, i dati sono rappresentati sottoforma di coppie chiave-valore in cui per valore si intende l'hash dell'oggetto che è stato inserito nella DHT mentre la chiave è rappresentata dalle keywords associate all'oggetto in questione. Gli oggetti in questione non sono altro che gli indirizzi dei canali MAM in cui le informazioni presenti sono raggruppate per argomento comune, oppure, indirizzi dei singoli messaggi nei canali MAM. Ogni nodo della rete gestisce un insieme di indirizzi che trattano di informazioni che possono essere descritte dal keywords set corrispondente. Gli oggetti prima di essere inseriti vengono sottoposti ad una funzione di hashing. Uno degli aspetti di questa funzione è quella di "rappresentare" i dati con il digest derivante, cioè, la stringa di lunghezza prefissata derivante dall'uso della funzione hash. Una volta che si ottengono queste stringhe derivanti dall'hashing, si può contattare un nodo IOTA per ottenere i dati relativi a quell'indirizzo oppure cercare lo smart contract associato per acquisire i dati.
In una ricerca keywords-search, quindi per parole chiave, si assume che ogni oggetto σ è associato a un keywords set K_σ.
Per ogni oggetto σ si dice che un keywords set descrive σ se $\mathrm{K} \subseteq \mathrm{K}_o$.
Nel momento in cui un oggetto viene inserito nella DHT, bisogna specificare le keywords con le quali associarlo. Risulta importante, ai fini di una ricerca, associare correttamente ai vari oggetti le giuste keywords. Descrivere correttamente il dato che si inserisce, permette una ricerca più facile dello stesso. Per fare un esempio, si potrebbero registrare i dati di temperatura e umidità dell'aria dei vari quartieri di Bologna e archiviarli su differenti indirizzi MAM. L'associazione con le giuste keywords permette di distinguere

i dati provenienti da un quartiere rispetto ad un altro.
Se si vogliono registrare i dati provenienti dal quartiere di San Donato si potrebbero utilizzare le seguenti parole chiave: "temperatura", "umidità", "San Donato", "Bologna", "Italia". Se si vogliono registrare i dati provenienti dal quartiere di Savena si potrebbero utilizzare le seguenti parole chiave: "temperatura", "umidità", "Savena", "Bologna", "Italia".

Sono presenti due tipi di ricerca associate al set di keywords:

- Ricerca **Pin Search**
- Ricerca **SuperSet Search**

La ricerca Pin Search consiste in una ricerca precisa dei dati corrispondenti al set di keywords K specificato. Il risultato sarà dato da quell'insieme di oggetti (σ_1, ..., σ_n) che sono descritti esattamente dalle parole chiave inserite. Il nodo che gestisce quelle keywords, una volta ricevuta la richiesta di accesso agli oggetti, restituirà all'utente tutti gli oggetti corrispondenti.
La ricerca Superset Search è simile alla precedente ma in più consente di stabilire il numero massimo di oggetti che si vogliono ricercare. È possibile stabilire una soglia massima c in cui il numero restituito sarà al massimo c. I risultati saranno composti, quindi, da tutti quegli oggetti che possono essere descritti dal keywords set K senza esserlo strettamente.
La peculiarità di questa ricerca è la possibilità di cercare oggetti che sono contenuti in un nodo specifico, quindi con una keyword specifica inserita per la ricerca, e poi cercarne altri contenuti nei nodi vicini al nodo in questione finché non si arriva alla soglia massima c specificata dall'utente.

3.2 Implementazione

3.2.1 Sistemi di simulazione di reti P2P

Per poter realizzare il progetto di tesi, cioè l'implementazione di una Distributed Hash Table, è stato necessario utilizzare un simulatore di reti peer-to-peer su cui appoggiare la progettazione.
PeerSim [16] è il simulatore scelto per realizzare il progetto. È uno strumento sviluppato in Java che permette un'elevata scalabilità e dinamicità per i vari componenti. La simulazione è supportata da molti componenti semplici, estendibili e collegabili, con un meccanismo di configurazione flessibile.

3.2.2 PeerSim

I sistemi P2P possono essere estremamente grandi (milioni di nodi). I nodi della rete si uniscono ed escono continuamente. Sperimentare con un protocollo in un ambiente del genere non è affatto un compito facile.
Per far fronte a queste proprietà e quindi per raggiungere un'estrema scalabilità in modo da supportare un certo dinamismo è stato utilizzato PeerSim [17]. La struttura del simulatore è basata su diversi componenti. Questi semplificano la prototipazione rapida di un protocollo, combinando diversi elementi costitutivi collegabili, rappresentati come oggetti Java.
L'idea generale del modello di simulazione prevede queste fasi:

1. scegliere la dimensione della rete (numero di nodi).
2. scegliere uno o più protocolli da sperimentare e inizializzarli.
3. scegliere uno o più oggetti *Control* per monitorare le proprietà di interesse e per modificare alcuni parametri durante la simulazione (es. La dimensione della rete, lo stato interno dei protocolli, ecc.).
4. eseguire la simulazione invocando la classe Simulator con un file di configurazione.

Tutti gli oggetti creati durante la simulazione sono istanze di classi che implementano una o più interfacce.
Quindi il simulatore imposta la rete inizializzando i nodi nella rete e i protocolli in essi contenuti. Ogni nodo ha gli stessi tipi di protocolli; ovvero, le istanze di un protocollo formano un array nella rete, con un'istanza in ogni nodo. Le istanze dei nodi e dei protocolli vengono create mediante clonazione. Cioè, solo un'istanza viene costruita utilizzando il costruttore dell'oggetto, che funge da prototipo, e tutti i nodi nella rete vengono clonati da questo prototipo. Per questo motivo è molto importante prestare attenzione all'implementazione del metodo di clonazione dei protocolli.
A questo punto, è necessario eseguire l'inizializzazione, che imposta gli stati iniziali di ogni protocollo. La fase di inizializzazione viene eseguita da oggetti Control pianificati per essere eseguiti solo all'inizio di ogni esperimento.
Dopo l'inizializzazione, il motore a ciclo chiama tutti i componenti (protocolli e controlli) una volta per ciclo, fino a un determinato numero di cicli o fino a quando un componente decide di terminare la simulazione. Ad ogni oggetto in PeerSim (controlli e protocolli) viene assegnato un oggetto Scheduler che definisce esattamente quando vengono eseguiti. Per impostazione predefinita, tutti gli oggetti vengono eseguiti in ogni ciclo.
Tuttavia, è possibile configurare un protocollo o un controllo per eseguirli solo in determinati cicli, ed è anche possibile controllare l'ordine di esecuzione dei componenti all'interno di ciascun ciclo.
Per la quasi totalità di simulazioni personalizzate sarà importante imparare a progettare due tipologie di componenti:

- **Protocolli**: classi che implementano l'interfaccia *Protocol*.
- **Controlli**: classi che implementano l'interfaccia *Control*.

La prima tipologia chiaramente tratta di protocolli, quello/i su cui si vuole sperimentare e quelli ausiliari ad esso, mentre la seconda di controllori che osservano i protocolli o/e agiscono modificando lo stato di questi ultimi.
La differenza fondamentale sta nel modo in cui essi vedono la rete; i Control vedono tutti i nodi e la rete nella sua totalità, quindi hanno accesso a tutte le informazioni di quest'ultima; i Protocol sono contenuti in ogni nodo, possono agire sul nodo stesso e sui

soli nodi vicini di cui hanno visione.
I nodi comunicano tra loro direttamente e viene dato loro il controllo periodicamente, in un certo ordine sequenziale, quando possono eseguire azioni arbitrarie, come chiamare metodi di altri oggetti ed eseguire alcuni calcoli. I Protocol inoltre, dentro il singolo nodo, possono essere disposti su più livelli, formando una pila, in modo del tutto analogo a quella della rappresentazione ISO/OSI o TCP/IP per esempio. Vi è anche la possibilità di mettere più protocolli, in parallelo, nello stesso livello.

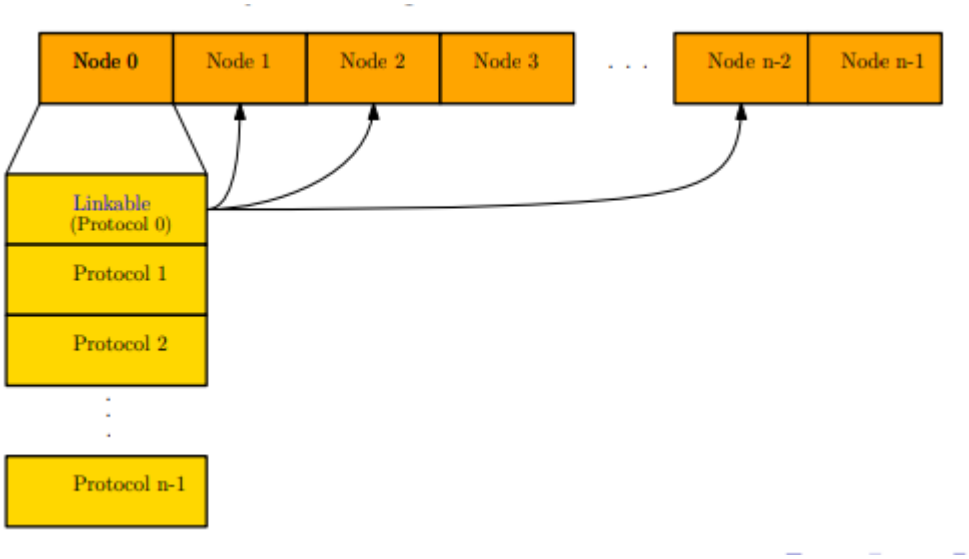

Figura 3.3: Istanze di protocolli contenuti in ogni nodo. [28]

Ogni istanza di un protocollo contenuta in un nodo avrà accesso ai dati e alle risorse delle istanze degli altri protocolli di livello minore o uguale, contenuti nel nodo stesso, e delle istanze dello stesso protocollo contenute negli altri nodi. Entrambe le tipologie di componenti hanno due caratteristiche importanti. La prima, fondamentale, è la periodicità con cui si attivano, ad ogni componente ne può essere assegnata una differente. La seconda riguarda con quali componenti sono collegati (quindi per i Protocol la scelta del livello).
Implica a quali dati e risorse di altri componenti hanno accesso, quindi cosa possono modificare e cosa possono sfruttare. Il collegamento è unidirezionale.

Il poter gestire velocemente e semplicemente queste due caratteristiche, a livello di codice, è uno dei punti di forza di Peersim, che ne evidenzia la sua versatilità. Per osservare l'andamento della simulazione si utilizzano Control specifici per questo scopo, che hanno quindi la caratteristica di *observer*. Essi possono essere usati per stampare dati anche su file.

PeerSim supporta due modelli di simulazione:

- il modello basato sul ciclo.
- il modello basato su eventi.

La differenza fondamentale tra i due modelli è la diversa visione del tempo. Nel modello basato sul ciclo viene dato in "cicli di esecuzione" mentre nel modello basato ad eventi viene dato in valori di tempo, quantità di "unità di tempo". Una volta scelta l'unità di misura del tempo si dovrà rimanere coerenti con essa ogni volta che si avrà necessità di inserire o manipolare un valore di tempo.

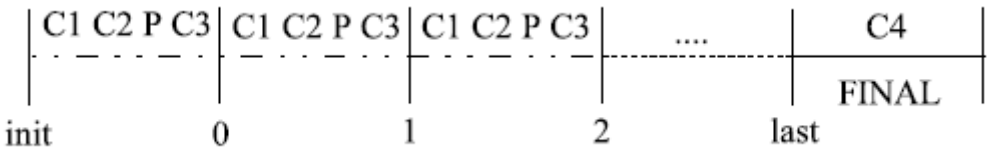

Figura 3.4: Pianificazione di controlli e protocolli. [17]

Nella Figura 3.4, le lettere "C" indicano un componente di controllo, mentre la lettera "P" indica un protocollo. I numeri nella parte inferiore dell'immagine indicano i cicli PeerSim. Dopo l'ultimo ciclo, è possibile eseguire un controllo finale per recuperare un'istantanea finale.

3.2.3 Inizializzazione della rete

La struttura ad ipercubo, come già specificato, può essere rappresentata da un grafo composto da 2^r nodi in cui r rappresenta la dimensione dell'ipercubo e anche la dimensione del vettore binario con il quale viene rappresentato ogni nodo in questo tipo di struttura. Per andare a realizzare questo tipo di struttura, innanzitutto bisogna inizializzare i vari parametri che andranno a costituire le proprietà di ogni nodo della rete. Per prima cosa, quindi, bisogna inizializzare la rete di nodi su cui poi si potrà costruire la struttura della DHT.

Nella classe "NetworkPeer.java" si procede con l'inizializzazione dei nodi della rete. Il numero dei nodi che comporranno la rete viene impostato direttamente dal file di configurazione. Questo file è molto importante in quanto rappresenta lo scheletro della simulazione che si andrà a creare. Qui i vari componenti vengono impostati nell'ordine con cui successivamente, durante la simulazione, verranno eseguiti.
Un problema riguardante questo tipo di struttura è quello relativo alla correlazione tra la dimensione dell'ipercubo e il numero dei nodi effettivi presenti nella rete. Per far si che questa tipologia di struttura funzioni correttamente c'è bisogno che entrambi siano uguali.
Se non ci fosse questa corrispondenza si dovrebbe gestire la situazione in cui i nodi logici dell'ipercubo potrebbero essere di più (o di meno) dei nodi fisici realmente presenti nella rete della DHT. Per risolvere questa situazione le strade percorribili sono due: un nodo fisico potrebbe gestire più porzioni dello spazio delle keywords andando a sopperire alla mancanza del numero di nodi necessario, oppure, alcuni nodi fisici non parteciperebbero al processo di indicizzazione e quindi non si occuperebbero di nessuna porzione dello spazio delle keywords.
In un'implementazione reale si potrebbero far gestire a più nodi della rete della DHT più nodi logici e quindi più keywords in modo da garantire il corretto funzionamento. In questa simulazione si è deciso di assegnare ad ogni nodo una porzione specifica dello spazio delle keywords e quindi per sopperire alla mancanza di nodi ne sono stati creati altri in modo da raggiungere il numero necessario per il funzionamento.

La classe "DynamicNode.java" sopperisce a questo problema andando a inizializzare quei nodi derivanti dalla differenza della dimensione dell'ipercubo e il numero dei nodi

realmente esistenti. I nodi inizializzati da queste due classi presentano un'unica differenza ed è quella derivante dal valore di una variabile booleana atta allo scopo di distinguere il nodo reale da quello creato per sopperire alla mancanza. L'idea è quella di andare a sostituire il nodo fittizio con un nodo reale che decide di unirsi alla rete. In questo modo la porzione delle keywords passa dal nodo fittizio al nodo reale entrante e cosi facendo la struttura rimane intatta senza subire conseguenze.

L'ultimo passo prima di completare l'inizializzazione della rete è quello di andare ad assegnare ad ogni nodo i propri vicini, quei nodi con cui il nodo in questione potrà comunicare e scambiare informazioni. Il numero dei vicini è dato da quei nodi che differenziano con il nodo in questione per un solo bit.
I nodi all'interno della rete vengono identificati attraverso la loro numerazione in formato binario in base alla dimensione dell'ipercubo. I vicini sono assegnati andando a calcolare per ogni nodo tutti coloro che si distinguono dal singolo nodo per un solo bit di differenza. Per esempio, se la dimensione dell'ipercubo è uguale a 4 e quindi i nodi sono identificati tramite un vettore a 4 bit, il nodo 1010 avrebbe come vicini i seguenti nodi: 0010 − 1000 – 1011 – 1110.
La classe che conclude questa inizializzazione con l'assegnazione dei vicini per ogni nodo è la classe "Neighbors.java". Queste tre classi estendono l'interfaccia Control. Le classi di tipo Control hanno il compito di osservare i protocolli e agire modificando lo stato di questi ultimi. Queste, però, vengono eseguite una volta sola in quanto servono esclusivamente per inizializzare i nodi della rete su cui poi operare.

3.2.4 Routing delle informazioni

Una volta che è stata inizializzata la rete di nodi bisogna costruire la struttura ad ipercubo con la quale i nodi possono scambiare messaggi e quindi permettere un routing delle informazioni senza incorrere in problemi di comunicazione.
Per fare questo, bisogna creare il protocollo di rete che andrà a regolare questa comunicazione su questa struttura ad ipercubo.
La classe "EProtocol.java" estende l'interfaccia Protocol e gestisce proprio questa comu-

nicazione tra i nodi. Una volta definito il protocollo esso sarà valido per tutti i nodi. PeerSim supporta due modelli di simulazione: modello a cicli e modello ad eventi. Sono state valutati entrambi i modelli di simulazione e si è constatato che la simulazione ad eventi è quella che meglio supporta questo tipo di comunicazione. Nella classe EProtocol è presente il metodo *processEvent()* che ha il compito di gestire gli eventi che intercorrono nella rete. Questo è un metodo del protocollo che è presente in ogni nodo. Sono state, quindi, implementate tutte le istruzioni tali da gestire tutti gli eventi che possono incorrere.

Per poter capire e regolare al meglio il funzionamento di questo protocollo, è stato necessario inserire manualmente i dati nella DHT. Come detto, i dati in questa DHT provengono da dei registri distribuiti ma, visto che non è stata ancora collegata ad una DLT, si è deciso di inserire una quantità di dati tali su cui poi effettuare delle operazioni di ricerca.

Per fare questo è stata realizzata la classe "Uxer.java" che svolge il ruolo di "interfaccia" tra la DHT e le varie operazioni che si possono effettuare. Questa classe per prima cosa consente di poter inserire il numero di oggetti su cui si vuole lavorare e associare ad ognuno una keyword con cui poi poterlo identificare.

Questi oggetti, in una implementazione reale, corrisponderebbero a degli indirizzi MAM della DLT IOTA ma non avendo collegato ancora la DHT alla DLT si è deciso di utilizzare degli indirizzi fittizi al posto di quelli reali. Vengono cosi instradati nella rete i vari oggetti con le relative keywords sottoforma di coppie chiave-valore. Nello specifico la chiave corrisponde, per l'appunto, alla keyword associata all'oggetto e il valore di questa coppia corrisponde all'indirizzo fittizio. Questo indirizzo prima di essere inserito viene sottoposto a SHA-256.

SHA è l'acronimo di Secure Hash Algorithm, è un algoritmo di hashing che prende in ingresso dei dati, ad esempio una stringa di testo di lunghezza arbitraria e restituisce un'altra stringa lunga 256 bit, dato che si utilizza SHA-256. Il metodo di inserimento nella rete di una coppia di dati <*Keyword, hashObject*> è effettuata andando ad attivare il metodo *processEvent()* del protocollo della classe EProtocol. L'inserimento rappresenta un evento che la rete deve gestire. Il nodo, nel momento in cui riceve un messaggio contenente questi dati si comporta in questo modo: controlla che la keyword presente

nella coppia di dati coincida o meno con il set di keywords che gestisce.
Nel caso in cui corrisponda, allora inserisce nella propria tabella di riferimento la coppia di dati, altrimenti la inoltra nella rete facendo passare il messaggio tramite i nodi vicini.

Qual è il modo in cui un nodo sceglie uno dei vicini a cui trasmettere il messaggio?
Il nodo sceglie in maniera accurata quale dei suoi vicini sia meno distante dal nodo che gestisce quel set di keywords a cui corrisponde la keyword della coppia di dati inseriti. Per fare questo calcola tra tutti i suoi nodi vicini, quel nodo che ha una distanza di Hamming minore rispetto agli altri verso l'obiettivo. Perciò, il nodo trasmette il messaggio al nodo vicino più vicino all'obiettivo. Questo processo viene iterato finché il messaggio non raggiunge destinazione.

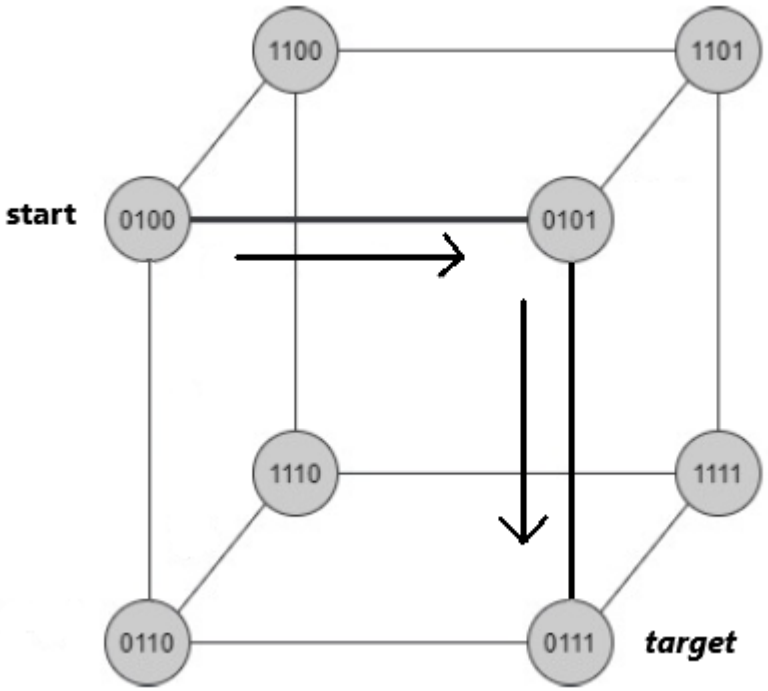

Figura 3.5: Meccanismo di routing tra i nodi.

Prendendo in considerazione la Figura 3.5, si ha il nodo di partenza p = 0100 e il nodo di destinazione q = 0111. Il nodo p attraverso l'algoritmo di routing contatta i suoi vicini e scambia il messaggio con il nodo più vicino al target. Il nodo più vicino è rappresentato dal nodo che ha più bit in comune con il nodo target, ovvero quello che in confronto a tutti gli altri vicini ha distanza di Hamming minore. È possibile che ci siano più percorsi ottimali per raggiungere l'obiettivo. Questi sono equivalenti se vengono portati a termine con lo stesso numero di hop tra i vari nodi. L'algoritmo sceglie sempre il primo che incontra.

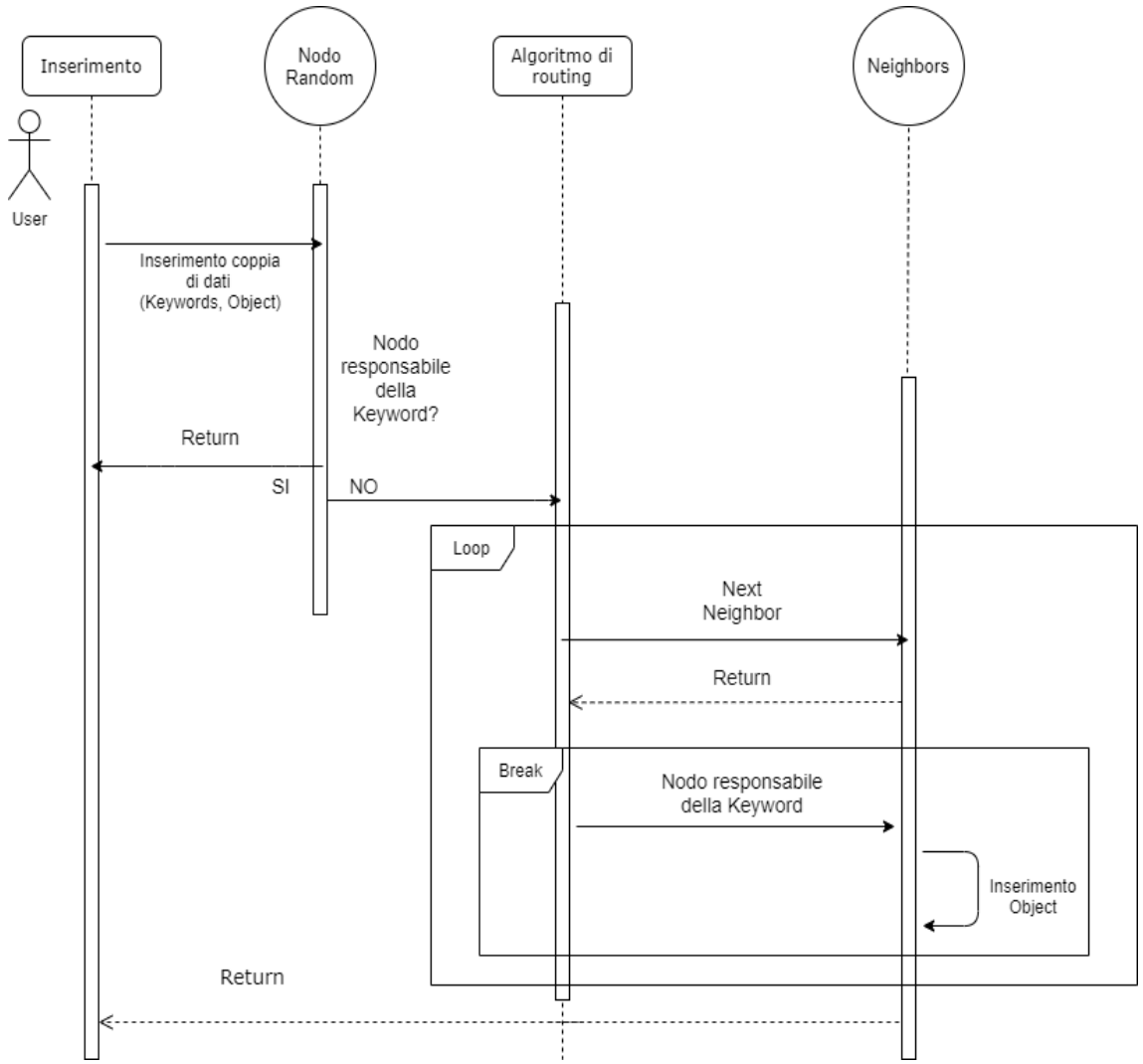

Figura 3.6: Diagramma di inserimento oggetti nella rete.

3.2.5 Operazioni di ricerca

Le operazioni di ricerca sono il fulcro di questo lavoro. Riuscire ad ottenere degli oggetti corrispondenti a delle keywords inserite per la ricerca, rappresenta lo scopo di questa DHT. La DHT in questione permette all'utente di effettuare due tipi di ricerca:

- Pin Search;
- SuperSet Search.

3.2.5.1 Pin Search

La ricerca comincia con l'inserimento da parte dell'utente di un set di chiavi K = (k_1,. . .,k_n) associato all'oggetto che vuole ottenere. Tramite la classe Uxer.java, è possibile interfacciarsi con la DHT ed effettuare le proprie ricerche. Una volta inserita la keyword il sistema sceglierà in maniera random un nodo da cui farla partire.
A questo punto si attiverà il metodo *processEvent()*, il quale gestirà questo evento.
Controlla innanzitutto se il nodo in questione è il nodo che possiede gli oggetti ricercati. Se non è lui il nodo, inoltra il messaggio di ricerca agli altri nodi secondo l'algoritmo di routing. Una volta che sarà raggiunto il nodo target, cioè quello che gestisce il sottoinsieme delle keywords in cui è presente la keyword inserita, il nodo restituirà tutti gli oggetti associati che possiede. A questo punto si otterrà una lista di oggetti che sono stati sottoposti ad hashing prima di essere inseriti nella rete. Si potrà poi controllare la corrispondenza degli indirizzi MAM relativi contattando un nodo IOTA per ottenere i dati a quell'indirizzo oppure cercare lo smart contract associato per acquisire i dati.

3.2.5.2 SuperSet Search

Per questo tipo di ricerca, la differenza che si ha rispetto alla Pin Search è relativa al numero di oggetti che si possono ottenere. La ricerca SuperSet estende la ricerca Pin

Search andando a recuperare sia gli oggetti archiviati dal nodo che gestisce la keyword inserita dall'utente ma anche gli oggetti gestiti da altri nodi e quindi rappresentanti di altri set di keywords. Nella fase di input, oltre a specificare la keyword, viene definita anche una variabile soglia c che rappresenta un valore massimo di risultati di risposta. La ricerca quindi risponderà con massimo c valori che possono essere descritti dal set K inserito dall'utente.
I nodi vicini tra di loro gestiscono set di keywords che possono descrivere oggetti che hanno delle caratteristiche in comune. Per questa ricerca il sistema agisce in questo modo: il messaggio di ricerca che viene inviato arriva fino al nodo di destinazione e vengono restituiti tutti gli oggetti contenuti da quest'ultimo e se il numero soglia c inserito dall'utente non viene raggiunto, si continua con la ricerca. A questo punto bisogna tenere conto del vettore r-bit del nodo. L'ID binario del nodo sarà costituito da valori pari ad 1 e valori pari a 0. La ricerca continua su quei nodi il cui ID presenta sulle stesse posizioni uguali ad 1 gli stessi valori pari ad 1 ma, valori differenti, sia 0 che 1, sulle posizioni dell'ID aventi valore 0. Facendo un esempio, se il nodo u ha un ID pari a 1010, verranno visitati i seguenti nodi: 1011 – 1110 – 1111.
La ricerca procede finchè non si raggiunge la soglia c richiesta dall'utente o non si ottengono tutti gli oggetti presenti nei nodi visitati.

Capitolo 4

Valutazione sperimentale

Quest'ultima parte si concentra sull'andare ad effettuare e successivamente analizzare dei test che mirano a valutare la bontà del lavoro svolto. Dopo aver progettato e realizzato la DHT Hypeercube, è importante capire quanto è efficiente il meccanismo di routing implementato.
Riuscire ad avere i dati richiesti inserendo le giuste keywords associate ed ottenere dei risultati in tempi brevi, rappresenta l'obiettivo che si prefigge questa DHT.
Efficienza e accuratezza costituiscono il monito di Hypeercube. Sono stati effettuati diversi test ipotizzando uno scenario in cui la rete della DHT fosse composta da un numero di nodi e da oggetti archiviati in essi di volta in volta differente in modo da valutare come il sistema reagisse con questi parametri diversi ogni volta che veniva interrogato.
Le simulazioni effettuate hanno preso in esame differenti dimensioni dell'ipercubo e per ognuna si è analizzata l'efficienza dello scambio di messaggi tra i nodi presenti nella rete. Si è valutato il numero di hop (scambi) che sono stati compiuti da un nodo di partenza e il nodo destinatario del messaggio. Nello specifico, ogni nodo è individuato all'interno della rete tramite un ID, il quale corrisponde ad un numero in formato binario. Ogni nodo ha un vicinato composto da nodi che differenziano da quello in questione per un solo bit. Per fare un esempio si può pensare alla rete della DHT composta da 256 nodi in cui ogni nodo ha un ID corrispondente ad otto cifre in binario. Se si considera il nodo 10101101 si può calcolare che il numero dei vicini è uguale al numero di variazioni di un bit rispetto allo stesso.

Più precisamente si hanno come vicini i seguenti nodi: [00101101 – 10001101 – 10100101 10101001 – 10101100 – 10101111 – 10111101 – 11101101].
Si vuole specificare che per ogni ricerca effettuata, che sia di tipo Pin Search o SuperSet Search, il client che la effettua è scelto in maniera random tra tutti gli altri.
Visto che la DHT non è stata collegata a nessuna DLT e quindi non è stato possibile basare le proprie ricerche su dati reali, il numero di oggetti viene impostato manualmente all'inizio di ogni simulazione. Per queste simulazioni gli oggetti corrispondono a dei dati che non hanno delle corrispondenze reali e quindi fondamentalmente dei dati inventati. Questi dati potrebbero rappresentare misurazioni di temperatura, umidità, qualità dell'aria di alcune città metropolitane, oppure dati riguardanti i livelli di traffico in determinati orari della giornata. Volendo continuare, questi dati potrebbero mostrare i livelli di affollamento di alcune città italiane durante i giorni festivi. Gli oggetti in questione una volta creati, vengono inseriti nella rete in maniera casuale. Questo significa che dei nodi possono contenere e quindi memorizzare oggetti mentre altri no. Le keywords associate ad ogni oggetto sono rappresentate dalle stringhe in binario degli identificativi dei nodi. Queste keywords vengono create e associate ai vari oggetti in maniera casuale.
"10111010" potrebbe essere una keyword associata all'oggetto "temperatura città di Bologna" e inserito nella DHT verso il nodo il cui ID corrisponde alla keyword menzionata. Ogni oggetto, quindi, ha associato una keyword che corrisponde all'ID binario del nodo e quindi quel nodo ha il compito di archiviare l'oggetto in questione.

4.1 Tipologia di test svolti

Per valutare le ricerca Pin Search e SuperSet Search, sono stati svolti dei test su dimensioni dell'ipercubo differenti. Nello specifico le dimensioni prese in esame variano da un numero di nodi presenti nella rete pari a 128 (r = 7) fino ad un numero di nodi pari a 8192 (r = 13). Sono stati effettuati test su queste dimensioni, da r = 7 fino a r = 13, e per ognuna si è inserito nella rete un numero di oggetti differente. Il numero di oggetti preso in esame varia da 100, 1000 e infine 10000.

Per la ricerca SuperSet Search, la soglia massima c di oggetti che vengono restituiti all'utente è stata impostata a 10. Le dimensioni dell'ipercubo scelte per l'analisi sono state quelle relative ad un numero di nodi pari a 128 (r = 7), 1024 (r = 10) e 8192 (r = 13). Sono state valutate dieci ricerche con dieci keyword differenti per ognuna.
In seguito, per valutare in maniera ancora più precisa la ricerca Pin Search e la ricerca SuperSet Search, è stata calcolata la media di 50 ricerche prendendo in esame le dimensioni da r = 7 fino a r = 13. Il numero di oggetti preso in considerazione e quelli restituiti nella ricerca SuperSet Search rimane lo stesso.
Inoltre, sono state calcolate per entrambe le ricerche, i valori corrispondenti alla deviazione standard e l'intervallo di confidenza. Con la deviazione standard si vuole rappresentare una distanza media, ovvero una distanza "tipica" di ogni singola osservazione dalla media. Più c'è variabilità tra le osservazioni, più grandi sono gli scostamenti dalla media, maggiore è la somma dei quadrati e, quindi, più elevato è il valore dalla varianza e di conseguenza anche dallo scarto quadratico medio.
Una forte asimmetria o la presenza di outliers possono far aumentare di molto il valore dello scarto quadratico medio e di conseguenza rendere anomali alcuni risultati dell'analisi.
L'intervallo di confidenza, calcolato al 95%, mira a stabile l'ampiezza dell'intervallo nella quale ci si aspetta di trovare il valore della media calcolato sul campione di 50 ricerche. Serve per dare una misura su quanto è possibile essere sicuri che il valore della media delle ricerche effettuate rientri in un certo intervallo.

4.2 Pin Search Test

La ricerca Pin Search permette di trovare tutti gli oggetti associati alla keyword inserita per la ricerca. Il nodo che gestisce il set di keywords in cui rientra la keyword caricata, restituirà tutti gli oggetti associati ad essa.

Di seguito vengono riportati i test effettuati sulle tre diverse dimensioni dell'ipercubo.

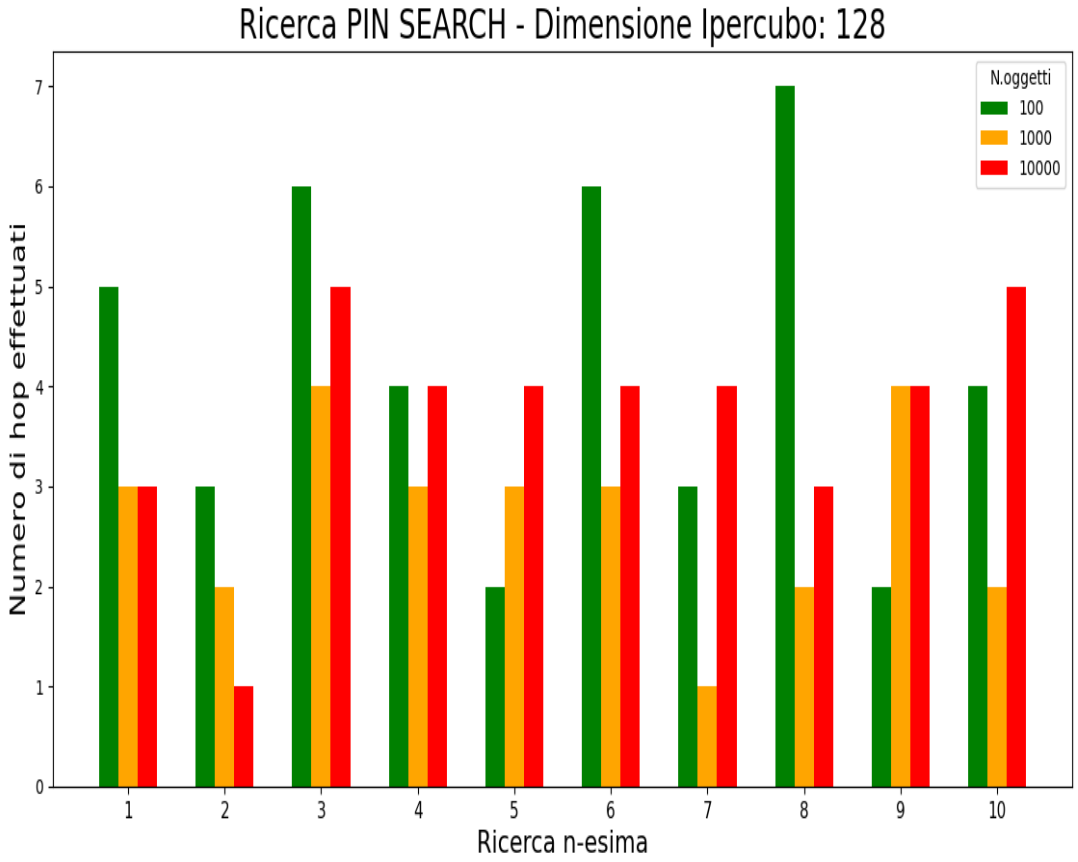

Figura 4.1: Ricerca con dimensione ipercubo pari a 128 nodi.

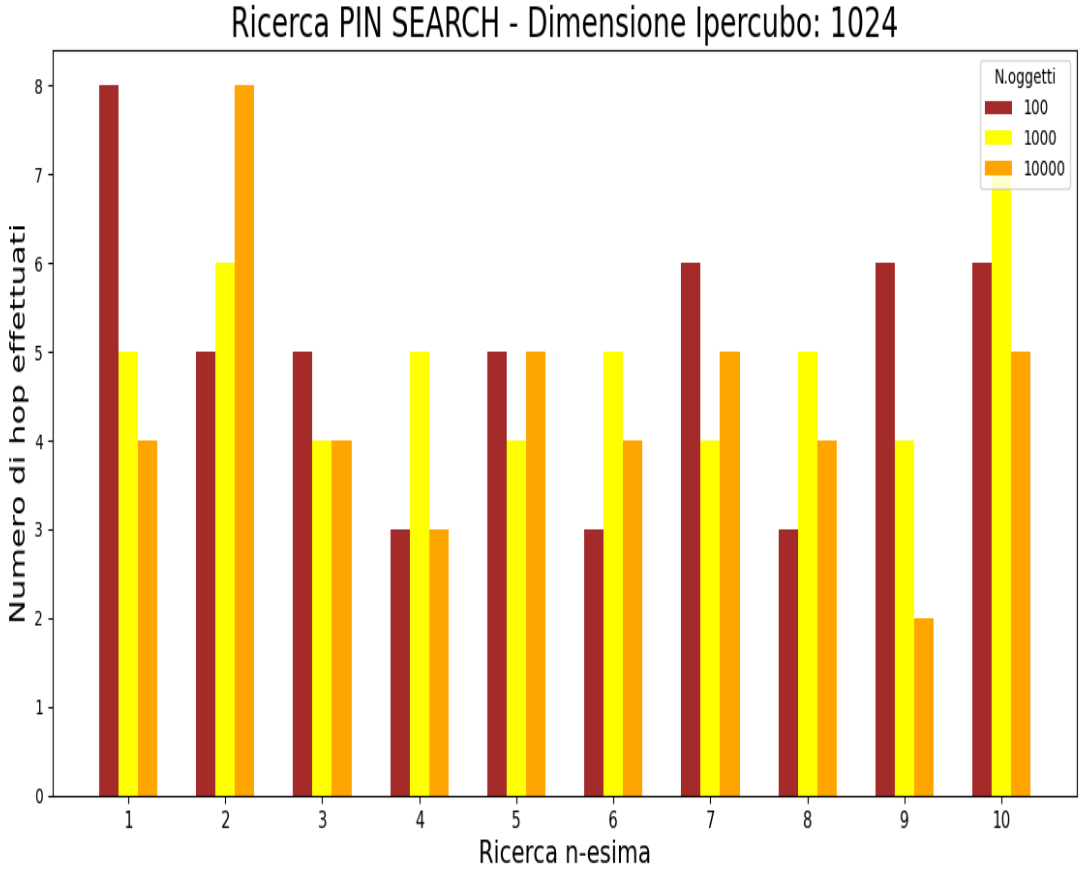

Figura 4.2: Ricerca con dimensione ipercubo pari a 1024 nodi.

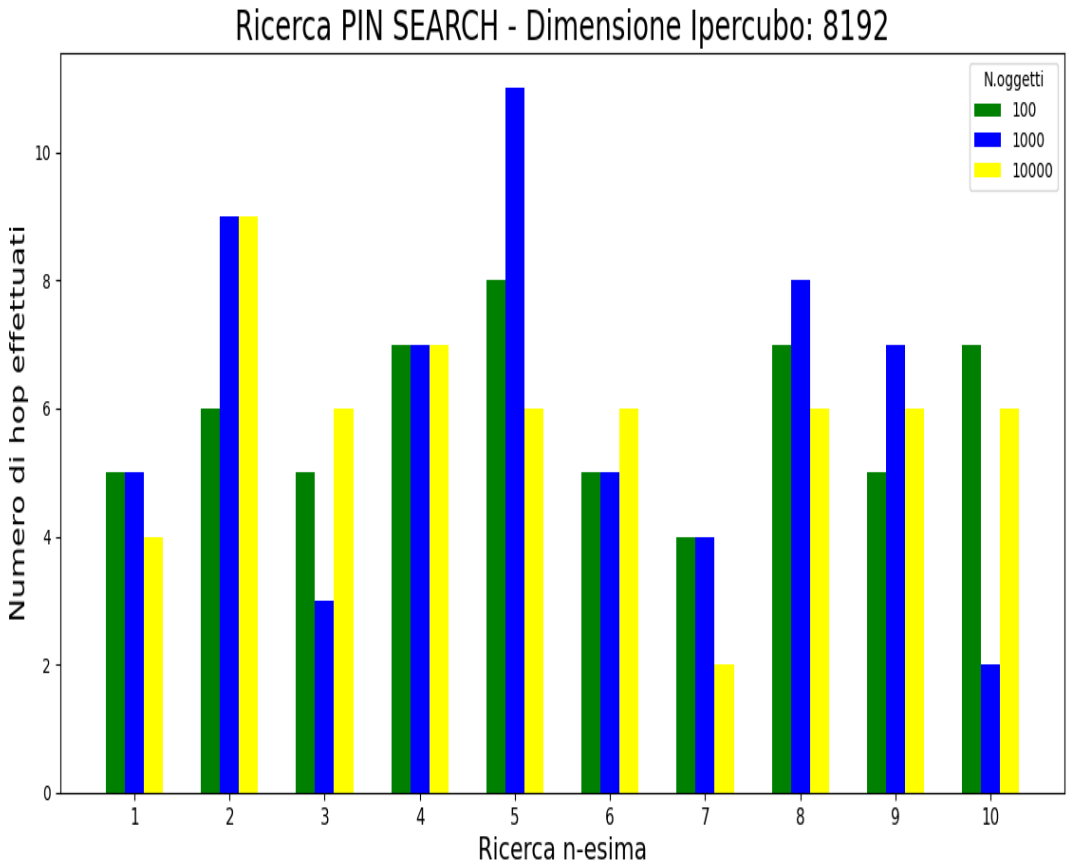

Figura 4.3: Ricerca con dimensione ipercubo pari a 8192 nodi.

Come si evidenzia facilmente da questi tre grafici, il numero di hop necessari per trasmettere un messaggio dal nodo di partenza fino al nodo di destinazione cresce quando si incrementa la dimensione dell'ipercubo. Questo comportamento è spiegabile dal fatto che aumentando la dimensione dell'ipercubo e quindi il numero dei nodi, automaticamente si allarga il percorso che un messaggio deve fare prima di arrivare a destinazione.

Con più precisione si può notare che incrementando il numero di oggetti archiviati nella DHT, il numero di hop necessari diminuisce. Essendo memorizzati più oggetti, si ha un ventaglio maggiore di keywords associate e di conseguenza si ha più probabilità che la keyword inserita sia legata ad un oggetto memorizzato da uno dei nodi.

Di seguito il test svolto prendendo in considerazione la media di 50 ricerche con le differenti dimensioni dell'ipercubo.

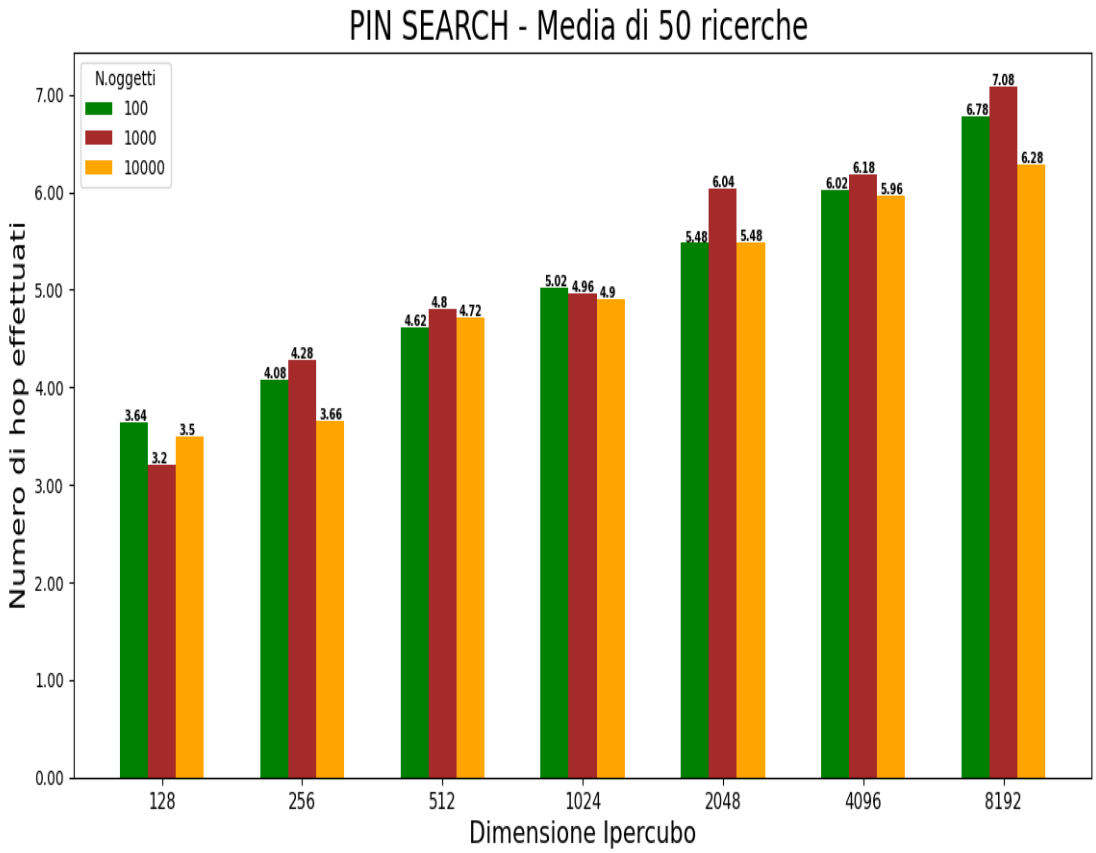

Figura 4.4: Media di 50 ricerche di tipo Pin Search.

Da questo grafico si può notare meglio quanto detto in precedenza. Il numero medio di hop aumenta all'aumentare della dimensione dell'ipercubo e mediamente il numero di scambi tra i nodi diminuisce con un numero di oggetti maggiore presenti nella DHT.
Il numero di hop medio passa da circa 3.5 per una dimensione pari a 128 (r = 7) a circa 6.72 per una dimensione pari a 8192 (r = 13). Si può dire che, all'incirca, ogni due dimensioni incrementate dell'ipercubo, il numero di hop cresce in media di una unità.

Di seguito la tabella con i valori della media, deviazione standard e intervallo di confidenza al 95% calcolati per ogni dimensione dell'ipercubo e divisi per numeri di oggetti archiviati nella DHT.

Dim. Ipercubo	Media	Deviazione Standard	Intervallo di Confidenza
128	3.64	1.3365	(3.27,4.01)
256	4.08	1.4546	(3.68,4.48)
512	4.62	1.5765	(4.18,5.06)
1024	5.02	1.6841	(4.55,5.49)
2048	5.48	1.7640	(4.99,5.97)
4096	6.02	1.5582	(5.59,6.45)
8192	6.78	1.6324	(6.33,7.23)

Tabella 4.1: Oggetti archiviati:100

Dim. Ipercubo	Media	Deviazione Standard	Intervallo di Confidenza
128	3.2	1.3248	(2.83,3.57)
256	4.28	1.4852	(3.87,4.69)
512	4.8	1.7023	(4.33,5.27)
1024	4.96	1.6777	(4.49,5.43)
2048	6.04	1.8512	(5.53,6.55)
4096	6.18	1.6123	(5.73,6.63)
8192	7.08	1.6015	(6.64,7.52)

Tabella 4.2: Oggetti archiviati:1000

Dim. Ipercubo	Media	Deviazione Standard	Intervallo di Confidenza
128	3.5	1.1294	(3.19,3.81)
256	3.66	1.3188	(3.29,4.03)
512	4.72	1.2460	(4.37,5.07)
1024	4.9	1.6933	(4.43,5.37)
2048	5.48	1.6932	(5.01,5.95)
4096	5.96	1.6283	(5.51,6.41)
8192	6.28	1.6418	(5.82,6.74)

Tabella 4.3: Oggetti archiviati:10000

Confrontando la deviazione standard con la media, si può notare che non è presente molta variabilità tra le osservazioni in quanto il valore dello scarto quadratico medio (deviazione standard) non è elevato. Il parametro in questione tende ad aumentare leggermente con l'incremento della dimensione dell'ipercubo ma rimane pressoché stabile e compreso tra i valori 1 e 2 in tutte e tre le analisi effettuate prendendo in considerazione differenti oggetti archiviati nella DHT. L'intervallo di confidenza serve per dare una misura su quanto il valore della media osservata sia in un certo intervallo. L'intervallo di confidenza presenta delle soglie poco distanti tra loro. Il valore medio reale quindi si avvicina molto alla media calcolata, nello specifico, si può affermare con una confidenza al 95% che la vera media sia inclusa in quest'intervallo e che si avvicini molto al valore calcolato.

4.3 SuperSet Search Test

La ricerca SuperSet permette di recuperare, oltre agli oggetti archiviati nel nodo che gestisce la keyword inserita, anche altri oggetti memorizzati nei nodi vicini e quindi rappresentanti di altri set di keywords. Viene definita anche una variabile soglia c che rappresenta un valore massimo di risultati di risposta. La ricerca quindi risponderà con massimo c valori che possono essere descritti dal set K inserito dall'utente.

Di seguito vengono riportati i test effettuati sulle tre diverse dimensioni dell'ipercubo.

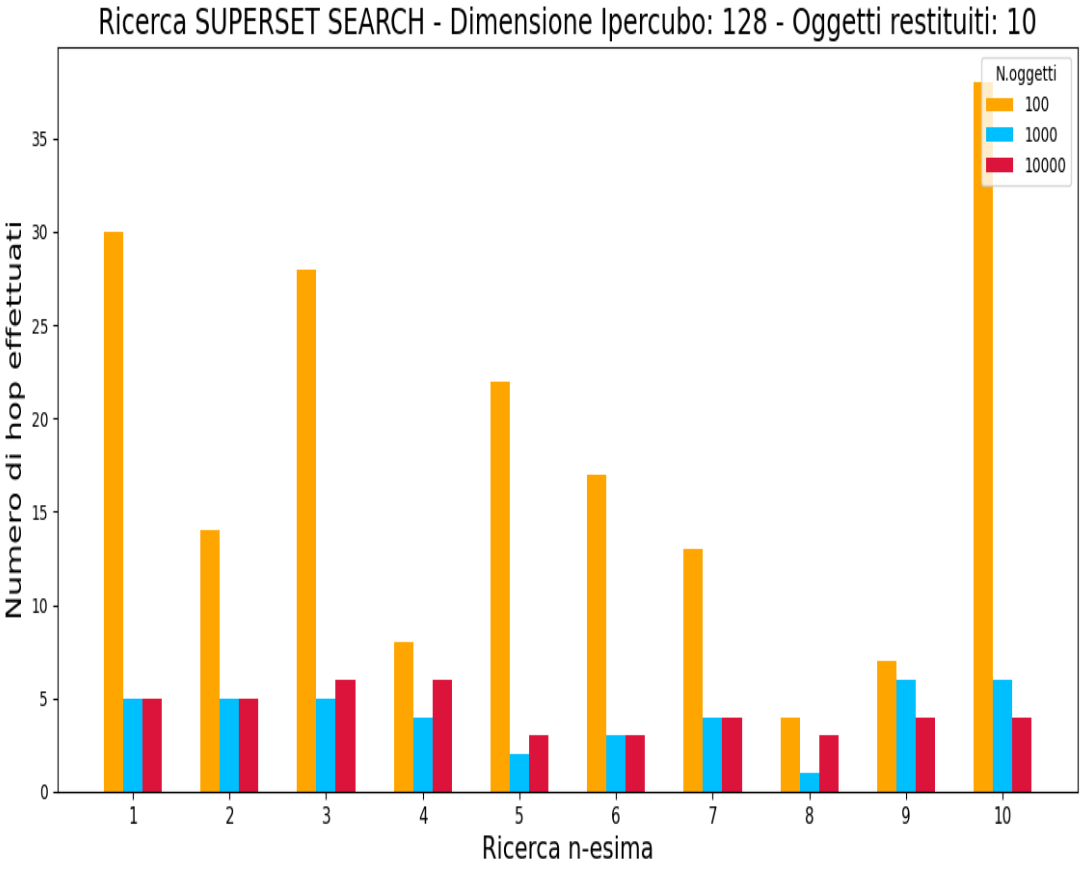

Figura 4.5: Ricerca con dimensione ipercubo pari a 128 nodi.

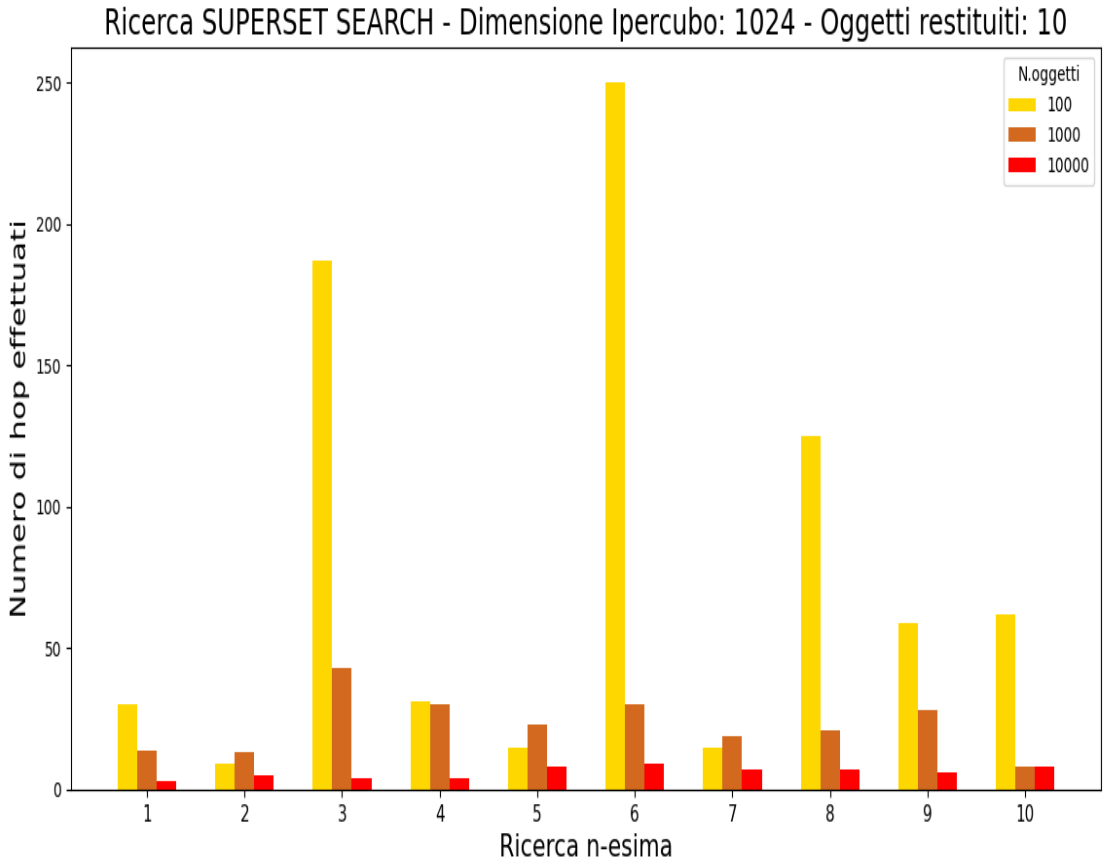

Figura 4.6: Ricerca con dimensione ipercubo pari a 1024 nodi.

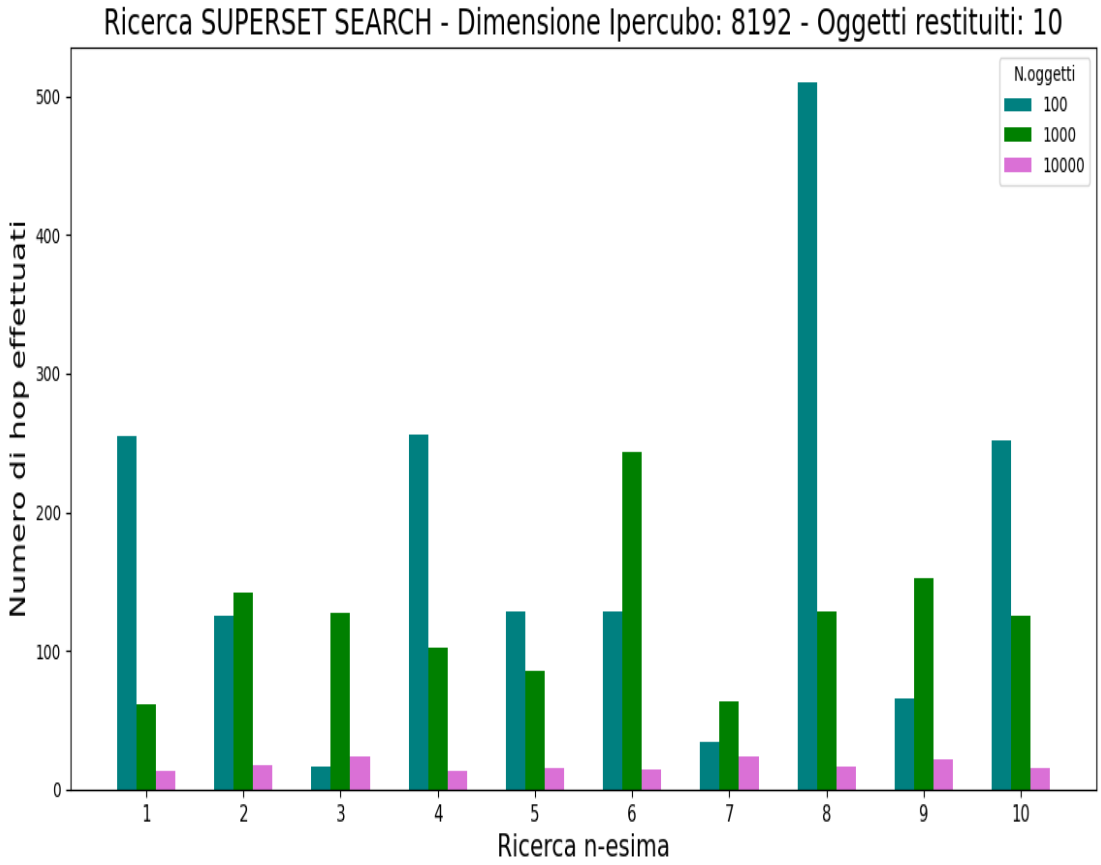

Figura 4.7: Ricerca con dimensione ipercubo pari a 8192 nodi.

Questi test svolti sulla ricerca SuperSet Search presentano dei risultati con dei valori dissimili rispetto alla precedente ricerca. Ad una prima analisi sicuramente risaltano all'occhio quei valori apparentemente anomali, corrispondenti ad un numero elevato di hop (scambi) tra i nodi, che aumentano all'aumentare della dimensione dell'ipercubo.
In ogni test svolto è possibile riscontrare queste "anomalie", valori eccessivamente elevati rispetto alla media, nel numero di hop necessari per completare la ricerca. Questi outliers si hanno in corrispondenza di un numero minore di oggetti presenti nella DHT. Questo fenomeno è spiegabile dal fatto che in una rete con molti nodi e pochi oggetti archiviati da quest'ultimi, è necessario un numero di hop maggiore per soddisfare la richiesta dell'utente.
I vari oggetti possono essere archiviati in nodi distanti tra loro e in una rete che presenta un numero elevato di partecipanti, c'è il rischio che il numero di hop tra i vari nodi risulta avere dei picchi elevati come nella Figura 4.7. A maggior ragione se il numero di oggetti richiesti dall'utente aumenta. Invece, più aumentano gli oggetti più il numero di hop tende a stabilizzarsi.

Di seguito il test svolto prendendo in considerazione la media di 50 ricerche con le differenti dimensioni dell'ipercubo.

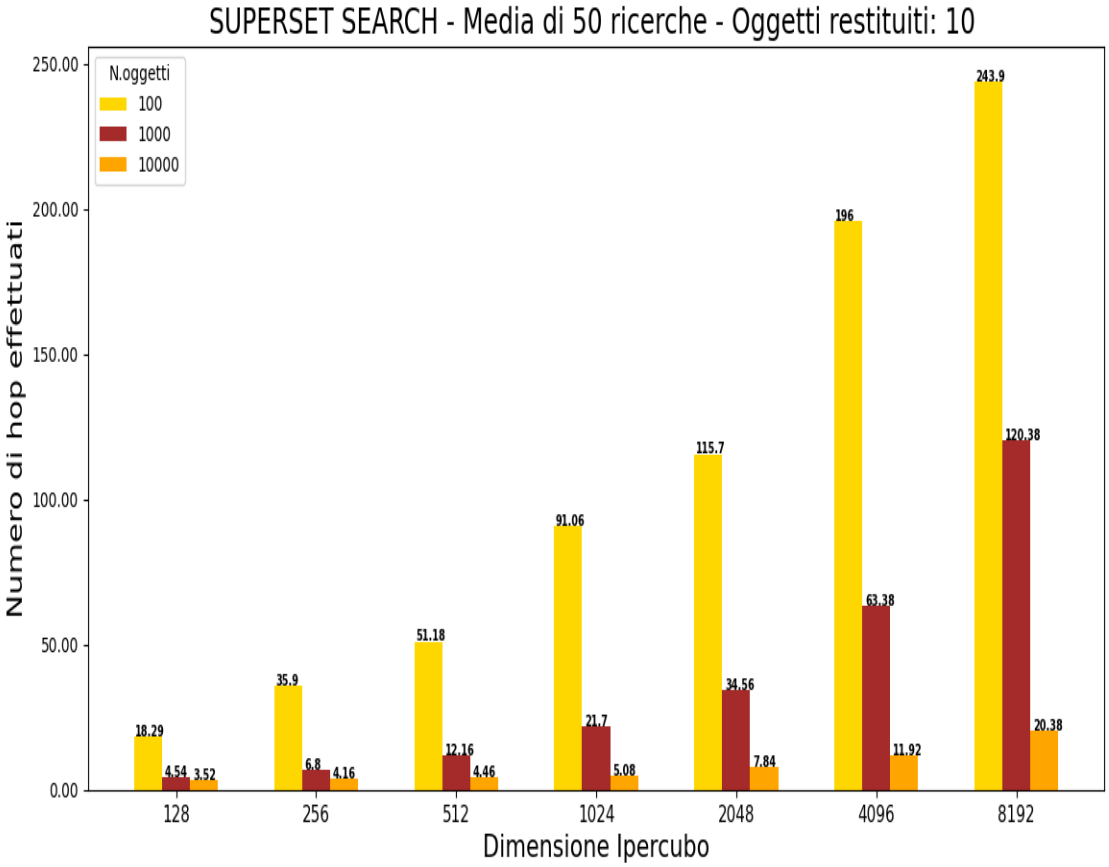

Figura 4.8: Media di 50 ricerche di tipo SuperSet Search.

Da questo grafico si può notare meglio quanto detto in precedenza. Il numero medio di hop aumenta all'aumentare della dimensione dell'ipercubo. Si nota anche qui che in corrispondenza di un numero basso di oggetti archiviati nella DHT, si hanno dei valori medi elevati riguardo al numero di hop da effettuare per soddisfare la ricerca. Tuttavia, il numero di scambi medi tra i nodi diminuisce con un numero maggiore di oggetti archiviati nella DHT.
Meno oggetti sono archiviati nella DHT e più aumenta in maniera considerevole, a tratti esponenziale, il numero di hop da effettuare con l'incremento della dimensione dell'ipercubo. Invece, più oggetti sono archiviati nella DHT e meno accentuato e più graduale è l'incremento del numero degli hop all'aumentare della dimensione dell'ipercubo.
Per questa ricerca è difficile ipotizzare una proporzione numerica tra l'aumento del numero di hop effettuati con l'incremento della dimensione dell'ipercubo perchè questi valori sono soggetti ad oscillazioni difficilmente prevedibili.

Di seguito la tabella con i valori della media, deviazione standard e intervallo di confidenza al 95% calcolati per ogni dimensione dell'ipercubo e divisi per numeri di oggetti archiviati nella DHT.

Dim. Ipercubo	Media	Deviazione Standard	Intervallo di Confidenza
128	18.28	8.4420	(15.94,20.62)
256	35.9	17.8934	(30.94,40.86)
512	51.18	37.8556	(40.69,61.67)
1024	91.06	72.4469	(70.98,111.14)
2048	115.7	98.3933	(88.43,142.97)
4096	196	186.8865	(144.20,247.80)
8192	243.9	253.5996	(173.61,314.19)

Tabella 4.4: Oggetti archiviati:100

Dim. Ipercubo	Media	Deviazione Standard	Intervallo di Confidenza
128	4.54	1.5414	(4.11,4.97)
256	6.8	2.2588	(6.17,7.43)
512	12.16	3.2972	(11.25,13.07)
1024	21.7	6.2344	(19.97,23.43)
2048	34.56	13.0010	(30.96,38.16)
4096	63.38	25.3714	(56.35,70.41)
8192	120.38	68.6529	(101.35,139.41)

Tabella 4.5: Oggetti archiviati:1000

Dim. Ipercubo	Media	Deviazione Standard	Intervallo di Confidenza
128	3.52	1.1993	(3.19,3.85)
256	4.16	1.4337	(3.76,4.56)
512	4.46	1.3126	(4.10,4.82)
1024	5.08	1.6884	(4.61,5.55)
2048	7.84	1.9832	(7.29,8.39)
4096	11.92	2.6404	(11.19,12.65)
8192	20.38	6.2821	(18.64,22.12)

Tabella 4.6: Oggetti archiviati:10000

Analizzando la deviazione standard e l'intervallo di confidenza delle ricerche di tipo SuperSet Search, si possono notare dei valori molto differenti rispetto alla precedente ricerca. Tra le varie osservazioni è presente una variabilità notevole.
Infatti, più c'è variabilità tra le osservazioni, più grandi sono gli scostamenti dalla media, e di conseguenza anche dallo scarto quadratico medio. In queste ricerche sono presenti degli outliers che possono far aumentare di molto il valore dello scarto quadratico medio. Questo fenomeno si ha, maggiormente, in corrispondenza di meno oggetti archiviati nella DHT.
All'aumentare di quest'ultimi, i valori tendono a diminuire notevolmente. Questo perchè sono presenti meno outliers e c'è meno asimmetria e di conseguenza i risultati presentano meno anomalie.
L'intervallo di confidenza serve per dare una misura su quanto il valore della media osservata sia in un certo intervallo. L'intervallo di confidenza al 95% presenta delle soglie molto distanti tra loro. Questa distanza diminuisce all'aumentare del numero di oggetti archiviati. Risulta difficile affermare con certezza che il valore medio reale si avvicini alla media calcolata in quanto il range risulta ampio. Più aumentano gli oggetti e più questa distanza diminuisce e in questo modo il valore medio reale tende ad avvicinarsi di più alla media calcolata.

Capitolo 5

Discussione

5.1 Analisi

Hypeercube è una DHT la cui rete è composta da un numero arbitrario di nodi, la quale permette l'archiviazione di oggetti e la relativa ricerca. Questa DHT presenta una particolare caratteristica che è quella di avere una struttura ad ipercubo. Le comunicazioni e gli scambi di messaggi avvengono tra quei nodi che differenziano di un solo bit l'uno con l'altro.
L'inserimento di informazioni all'interno della rete prevede l'instradamento di un messaggio formato da una coppia di dati chiave-valore in cui la chiave è una keyword che identifica l'oggetto, e come valore si ha l'hash dell'oggetto stesso. Sia per l'inserimento che per la ricerca di informazioni nella DHT, i messaggi vengono scambiati di nodo in nodo fino a che il messaggio non arriva a colui che gestisce la keyword inserita.
Per valutare l'efficienza della DHT Hypeercube, si è calcolato il numero di hop medio che occorre per far arrivare un messaggio a destinazione partendo da un nodo arbitrario della rete. Nei test effettuati si può notare che sia nella ricerca di tipo Pin Search e sia nella ricerca di tipo SuperSet Search, il numero di hop aumenta all'aumentare della dimensione dell'ipercubo e più sono presenti oggetti archiviati nella DHT più il numero di hop diminuisce.
Più precisamente questo numero tende a stabilizzarsi e non avere dei valori considerati eccessivi rispetto alla media. Volendo trarre una conclusione su questi test svolti, si è

visto che dopo aver effettuato 50 ricerche di tipo Pin Search per calcolare la media degli hop tra i nodi, il numero medio di questi hop è circa 3 per una dimensione dell'ipercubo pari a 128 e circa 7 per una dimensione dell'ipercubo pari a 8192. Questo numero rimane comunque stabile in proporzione sia all'aumentare che al diminuire della dimensione della rete.

La conclusione che è possibile trarre a seguito di queste ricerche è che la media degli scambi tra i nodi per inviare e far ricevere al nodo destinatario il messaggio presenta un valore relativamente basso. Questo risultato si può considerare ottimo poiché rientra nell'ordine di O(log N) che rappresenta il risultato ideale in questo tipo di ricerca per una DHT.

5.2 Caso d'uso

Una Distributed Hash Table è uno strumento che permette di archiviare in maniera distribuita qualsiasi tipo di dato in una rete formata da migliaia e migliaia di nodi. Il dogma informatico di questo tipo di struttura è la decentralizzazione. Decentralizzare delle informazioni significa non affidarle nelle mani di un'unica entità bensì affidarle ad una rete di persone connesse tra di loro. Creare dei sistemi decentralizzati porta molti vantaggi alla comunità. Hypeercube mira ad offrire un servizio che permetta di aiutare questa rete di persone ad avere un accesso facile e veloce a qualsiasi tipo di informazione venga richiesta.
In base ai test effettuati, Hypeercube risulta avere come tempi di risposta alle richieste, un range di risultati che ci si aspetta di ottenere da una DHT.
Il rapporto che intercorre tra il numero di nodi da contattare per ottenere un risultato e il numero di keywords memorizzate nei vari nodi è un valore significativamente ottimo. Questo risultato, infatti, significa che nel momento in cui viene inviato un messaggio da un qualsiasi nodo della rete verso il nodo di destinazione, il numero di scambi che deve essere effettuato è relativamente basso nonostante che ad ogni nodo sia corrisposta soltanto una keyword in particolare.

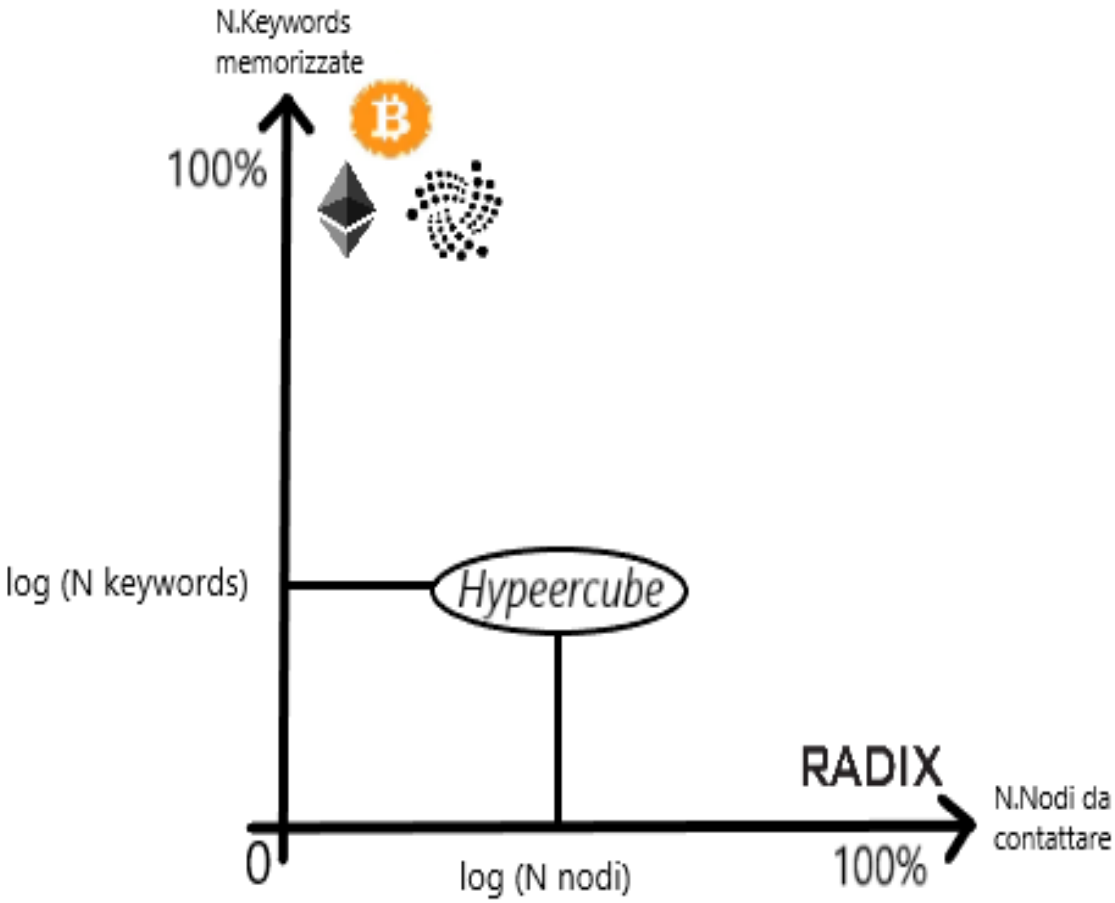

Figura 5.1: Grafico caso d'uso.

Mettendo a confronto la DHT Hypeercube con altri sistemi peer-to-peer come le DLT o le Blockchain, si possono notare delle differenze sostanziali sia sul come operano questi sistemi P2P sia sul ruolo dei nodi all'interno di questi sistemi. Blockchain come quella di Bitcoin, Ethereum o DLT come quella di IOTA sono composte da migliaia di nodi. Questi nodi si differenziano in base alla tipologia di compito che svolgono all'interno della rete. I masternode o full node sono quei nodi che mantengono per intero il registro delle transazioni avvenute e con tutti i relativi dati annessi. Il numero di questi particolari nodi varia da sistema a sistema e visto che richiedono dei requisiti specifici come essere operativi ogni giorno tutti i giorni, gran consumo di energia elettrica, vasto spazio di memoria per immagazzinare l'intero registro, non sono nodi che si trovano in un numero elevato nella rete.
Per richiedere delle informazioni specifiche c'è bisogno di interrogare uno di questi nodi in quanto essi sono coloro che mantengono il registro con tutte le informazioni annesse. Interrogando quindi uno di questi nodi si ha una risposta sicura perché se quello che cerchiamo esiste allora sarà sicuramente presente e quel nodo potrà fornire la risposta attesa. D'altro canto, si potrebbero creare degli intasamenti, dei colli di bottiglia, dati dal fatto che il numero di questi nodi è basso e che il numero di richieste può essere elevato in un determinato momento. Si verrebbero a creare degli intasamenti nel caso venga interrogato continuamente un unico nodo e quindi dei rallentamenti nei tempi di risposta.
Ci sono poi altre DLT come Radix che presentano delle caratteristiche differenti. I nodi che formano la rete ottengono un'identità nel momento in cui ne entrano a far parte. Radix è una DLT che utilizza il sistema dello sharding per la gestione delle proprie informazioni. L'identità di un nodo viene usata per identificare il root shard di quel nodo, ovvero lo shard che quel nodo dovrà sempre mantenere. In base alla capacità di calcolo del nodo, esso può gestire un numero di shard tale da permettergli di operare senza compromettere il proprio lavoro. Dunque, ciascun nodo cerca di mantenere più shard possibili, diminuendoli se non riesce a restare sincronizzato. Le informazioni vanno ricercate, quindi, nel nodo che gestisce lo shard nel quale è presente. Nel caso peggiore

potrebbero essere interrogati tutti i nodi della rete o in caso di ridondanza dei dati presenti, potrebbero essere comunque molti i nodi da interrogare.
Hypeercube si pone come una soluzione intermedia rispetto ad altri sistemi P2P per quanto riguarda le modalità di propagazione delle informazioni tra i vari nodi. Infatti, secondo i test effettuati, il sistema progettato risponde in maniera corretta e veloce a qualsiasi interrogazione gli viene posta. Sia l'inserimento che la ricerca di oggetti all'interno della rete richiede un numero medio di scambi tra i nodi che rientra nell'ordine di O(log N), dove N rappresenta il numero totale di nodi nella rete.
Il risultato ottenuto è sicuramente soddisfacente per la DHT Hypeercube.

Conclusioni

Hypeercube è una DHT progettata con lo scopo di semplificare la ricerca e l'acquisizione di dati. Questo strumento rientra in una architettura più grande [1], la quale, assicura nel suo insieme di poter garantire la veridicità dei dati forniti. È molto importante dare fiducia a chiunque utilizzi questo strumento, mostrando qualità ed efficienza nel servizio. Il processo di acquisizione è reso più semplice e veloce grazie al meccanismo implementato che permette la ricerca utilizzando delle keywords specifiche associate a dati specifici. Questo funzionamento rapido è garantito dalla struttura ad ipercubo con la quale è stata implementata la DHT Hypeercube. Utilizzando le due tipologie di ricerche presenti all'interno della DHT, l'utente può entrare in possesso di grandi quantità di dati su cui poi basare le proprie analisi.
Sui test effettuati si può concludere che dopo 50 ricerche di tipo Pin Search, il numero medio degli hop tra i nodi è circa 3 per una dimensione dell'ipercubo pari a 128 e circa 7 per una dimensione dell'ipercubo pari a 8192. Questo numero rimane comunque stabile in proporzione sia all'aumentare che al diminuire della dimensione della rete.
I test effettuati su entrambe le tipologie di ricerca ci consentono di affermare che la DHT permette di rispondere alle varie interrogazioni in tempi brevi. Il risultato ottenuto è da considerare ottimo poiché il numero medio di scambi tra i nodi rientra nell'ordine di O(log N), dove N rappresenta il numero totale di nodi nella rete.
Le Distributed Hash Tables forniscono un efficiente livello di astrazione per l'instradamento e di gestione dei dati nei sistemi distribuiti. Rispetto ai sistemi centralizzati, consentono di avere una maggiore scalabilità, minor latenza di instradamento e una più efficiente tolleranza ai guasti e adattabilità alle varie situazioni.
La caratteristica principale di Hypeercube è quella di essere, per l'appunto, un sistema

decentralizzato. La decentralizzazione sta diventando sempre più un modello di riferimento, lo si riscontra ancora di più in quelle realtà che fanno un uso maggiore di registri distribuiti come per esempio la blockchain.
Questi sistemi garantiscono affidabilità, immutabilità e trasparenza nei dati, qualità sempre più richieste dalle persone. Con questi strumenti è possibile creare un mercato di dati nel quale il singolo consumatore diviene nello stesso tempo compratore e venditore. Con questi strumenti è possibile diventare padroni dei propri dati.

Ringraziamenti

Siamo giunti alla fine di questo percorso. Un cammino che mi ha formato professionalmente e ha arricchito molto la mia persona. Ne esco consapevole della mia strada.

I miei ringraziamenti vanno in primis alla mia famiglia, la prima di ogni cosa, per avermi dato la possibilità e il sostegno necessario per affrontare questi studi.

Ringrazio vivamente il professor Stefano Ferretti e il dottor Mirko Zichichi per la professionalità e la disponibilità che mi hanno concesso durante tutto il percorso di tesi.

Ringrazio tutti i miei amici e i miei compagni di corso perché grazie a loro questo percorso è stato più facile da affrontare.

Ma soprattutto, un ringraziamento speciale va alla mia ragazza Elena con cui ho passato insieme questi anni di studio e con cui passerò insieme il resto della mia vita.

Bibliografia

Articoli

[1] Mirko Zichichi, Stefano Ferretti e Gabriele D'Angelo, "A Framework Based on Distributed Ledger Technologies for Data Management and Services in Intelligent Transportation Systems", 2020

[2] Yuh-Jzer Joung, Li-Wey Yang e Chien-Tse Fang, "Keyword Search in DHT-based Peer-to-Peer Networks", 2007

[3] Artur Olszak, "HyCube: A distributed hash table based on a variable metric", 2016

[4] Federico Mazzini, "Query sull'infrastruttura IOTA: un approccio keyword-based", 2019

[6] Maarten van Steen e Andrew S. Tanenbaum, "A brief introduction to distributed systems", 2016

[9] Ion Stoica et al., "Chord: a scalable peer-to-peer lookup protocol for Internet applications", 2003

[10] Petar Maymounkov e David Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric", 2005

[12] Satoshi Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2008.

[13] Vitalik Buterin, "Ethereum white paper," 2014

[14] Serguei Popov, "The tangle", 2015

[16] Alberto Montresor and Márk Jelasity, "PeerSim: A Scalable P2P Simulator", 2009

[17] Gian Paolo Jesi, "PeerSim HOWTO: Build a new protocol for the PeerSim 1.0 simulator", 2005

Libri

[5] R. Steinmetz and K. Wehrle, "P2P Systems and Applications", 2005

[7] Cormen, Thomas H. et al., "Chapter 11: Hash Tables". Introduction to Algorithms, 2001

[8] Klaus Wehrle, Stefan Gotz, Simon Rieche, "Distributed Hash Tables", 2007

Online

[11] Tradeix, The Difference Between Blockchain and Distributed Ledger Technology, URL: https://tradeix.com/distributed-ledger-technology/

[15] IOTA Foundation, IOTA Documentation, URL: https://www.iota.org

[18] Connessioni peer-to-peer nella rete, URL: https://www.pandasecurity.com/it/mediacenter/mobile-news/peer-to-peer/

[19] La nuova via della decentralizzazione, URL: https://vitolavecchia.altervista.org/definizione-e-caratteristiche-di-un-sistema-distribuito-distributed-system/

[20] Modelli di rete a confronto, URL: https://guardacome.com/reti-p2p-quali-sono-quali-sono-i-loro-vantaggi-e-svantaggi-e-quali-tipi-ci-sono/

[21] Indicizzazione delle chiavi con i rispettivi valori, Funzione di hashing applicata alle chiavi, Concatenamento degli elementi nella tabella, URL: https://www.programiz.com/dsa/hash-table

[22] Dati sottoposti ad hashing prima di essere instradati nella rete, URL: https://www.educative.io/edpresso/what-is-a-distributed-hash-table

[23] Vista dell'overlay e underlay di una Distributed Hash Table, Due metodi per l'archiviazione dei dati nelle Distributed Hash Table, URL: http://pages.di.unipi.it/ricci/14-03-2006-DHT.pdf

[24] Consistent hashing sull'hash ring, URL: https://www.acodersjourney.com/system-design-interview-consistent-hashing/

[25] Centralised vs Distributed Ledger, URL: https://www.newtechnorthwest.com/blockchain-crypto-basics/

[26] Un blocco della blockchain contenente diverse transazioni, La catena di blocchi della blockchain, URL: https://www.blockchain4innovation.it/esperti/blockchain-perche-e-cosi-importante/

[27] Ecosistema Ethereum, URL: https://ethereum.org/it/

[28] Istanze di protocolli contenuti in ogni nodo, URL: https://www.cs.unibo.it/ babaoglu/courses/csns15-16/slides/peerSim.pdf