Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Per Il Management

Query sull'infrastruttura IOTA: un approccio keyword-based

Relatore:
Chiar.mo Prof.
Gabriele D'angelo

Presentata da:
Federico Mazzini

Correlatori:
Chiar.mo Prof.
Stefano Ferretti

Chiar.mo Dott.
Mirko Zichichi

Sessione III
Anno Accademico 2018/2019

Ringraziamenti

Desidero ringraziare chi nel corso di questi tre anni mi è stato vicino, sia dal lato tecnico, sia da quello umano, chi mi ha supportato e chi sopportato.

Un ringraziamento speciale ad Anna, compagna di studio e di giornate, mi hai fatto capire che l'impegno è più meritevole delle capacità.

Grazie alla mia famiglia per avermi dato la possibilità di formarmi e aver creduto in me.

Grazie al Professor Stefano Ferretti e al Dottor Mirko Zichichi per la professionalità e la disponibilità concessami lungo tutto il periodo di tirocinio e di tesi.

Grazie ai miei compagni di viaggio Giovanni, Michele, Andrea e Francesco con cui ho condiviso le ansie e le gioie di questi tre anni accademici. Un ringraziamento speciale a Giovanni, dal primo giorno un amico, oltre che collega.

Grazie ai miei amici per avermi offerto un appoggio su cui contare, ma anche la leggerezza in cui svagarmi.

A tutti voi.

Abstract

I registri distribuiti o Distributed Ledger Technology (DLT) sono un particolare tipo di architettura che consente la memorizzazione di dati in maniera distribuita e permanente, senza l'ausilio di un entità centrale. Il primo esempio di tecnologia DLT è rappresentato dalla struttura dati Blockchain, sviluppata per la gestione della criptovaluta Bitcoin. Da quella prima implementazione ne sono seguite altre, che spaziano ad altri campi applicativi e che perciò possiedono caratteristiche diverse dalla tipica struttura a blocchi. Tra queste si trova IOTA, una criptovaluta sviluppata in riferimento all'ambiente dell'Internet of Things (IOT), la quale permette di effettuare micro transazioni a costo zero (zero *fee*). Una delle caratteristiche più importanti di IOTA è il protocollo di comunicazione MAM, attraverso il quale vengono creati canali di messaggi di tipo publisher-subscriber. La creazione e la scrittura di questi è funzionale e potente, ma la ricerca è possibile solamente conoscendo un indirizzo denominato *root*. Lo scopo di questo progetto di tesi è presentare una struttura distribuita in grado di eseguire query keyword-based all'interno del registro IOTA. E' stato progettato un overlay basato su Distributed Hash Table (DHT) con topologia a ipercubo. A partire da un set di keyword, la struttura DHT indicizza i messaggi MAM, ma non li replica, garantendo così l'integrità dei dati che tanto contraddistingue i registri distribuiti. Infine, è presentata una riflessione sui Permanode, particolari nodi all'interno di IOTA.

Indice

Elenco delle figure

Introduzione

Nel corso degli ultimi anni si è riscontrato un notevole interesse verso le Distributed Ledger Technology (DLT), un particolare tipo di architettura che consente la memorizzazione di dati in maniera distribuita e permanente. Questi registri sfruttano protocolli di tipo Peer-to-Peer e non si avvalgono di nessuna entità centrale di controllo. Attraverso algoritmi di consenso garantiscono alti livelli di sicurezza, trasparenza e anonimità.

Il primo esempio di tecnologia DLT è rappresentato dalla struttura dati Blockchain, sviluppata nel 2008 per la gestione della criptovaluta Bitcoin. Da quella prima implementazione, ne sono seguite altre, che spaziano ad altri campi applicativi e che perciò possiedono caratteristiche diverse dalla struttura a blocchi.

Tra queste vi è IOTA, anch'essa una criptovaluta, ma sviluppata in riferimento all'ambiente dell'Internet of Things (IOT). Si stima che entro i prossimi 5 anni i dispositivi coinvolti in relazioni *machine-to-machine* arrivino fino a 75 miliardi di unità. Questa enorme mole di macchine che comunica con la rete genera un flusso di dati prezioso, ma complesso da gestire. A questo proposito IOTA cerca di sopperire alle mancanze che presenta la Blockchain, quali scalabilità, portabilità e la presenza di fee, sviluppando un'infrastruttura che permetta micro transanzioni a costo zero.

La struttura di IOTA non si basa su una catena di blocchi, bensì in un Directed Acyclic Graph (DAG), che IOTA chiama Tangle, traduzione di groviglio. Affinché una transazione possa essere salvata all'interno del Tangle, questa de-

ve validare, attraverso un algoritmo di consenso, due transazioni già presenti nel registro, che precedentemente avevano fatto lo stesso. Questa diversa distribuzione del carico di lavoro, anche per quanto riguarda lo svolgimento della *Proof-of-Work*, svolta per ogni transazione ma molto meno costosa, concede all'infrastruttura proprietà scalabili e più alti livelli di transazioni per secondo (TPS) rispetto alla tecnologia Blockchain.

Una delle caratteristiche più importanti di IOTA è il protocollo di comunicazione MAM, acronimo di Masked Authenticated Message. Come si evince dal nome, il modulo MAM permette di pubblicare flussi di messaggi (stream), sotto forma di concatenazione di transazioni verso il Tangle. Questi flussi di messaggi costituiscono delle strutture denominate channel, a cui un utente può sottoscriversi, proprio come ci si iscrive a un canale YouTube. Il protocollo MAM apre le porte a tutta una serie di progetti riguardanti i dati e la loro fruizione, primo fra tutti il Data Marketplace, progetto sperimentale portato avanti dalla IOTA Foundation. In questa piattaforma è possibile, per un dispositivo connesso alla rete, trasmettere periodicamente i dati che lui stesso raccoglie sotto forma di stream, andando a creare un canale MAM.

Questi flussi di dati in entrata al Tangle hanno un enorme potenziale e la loro scrittura è funzionale e potente, ma la fruizione di questi, cioè la possibilità da parte degli interessati di entrare in possesso di dati precisi e inerenti a una ricerca ben definita, non è allo stesso livello.

All'interno di questa tesi si presenta una struttura per la realizzazione di query su DLT, con particolare riferimento e applicazione al registro distribuito IOTA. E' stato progettato un overlay basato su Distributed Hash Table (DHT), i cui nodi mantengono informazioni specifiche riguardo a transazioni contenute nel Tangle. A partire da un set di keyword, la struttura DHT indicizza le informazioni del Tangle, ma non le replica, garantendo così l'integrità dei dati che tanto contraddistingue i registri distribuiti. Sono stati implementati inoltre i moduli per l'invio e la ricezione di transazioni e messaggi MAM e sono stati ef-

fettuati alcuni test sull'effettivo funzionamento e sulle prestazioni. Inoltre, verrà presentato uno schema riguardante l'organizzazione dei Permanode, particolari nodi che mantengono in memoria tutto il Tangle.

Il seguito della tesi è strutturato in 7 capitoli. Il capitolo 1 riprende quello che è lo stato dell'arte delle tecnologie trattate nel testo, quali Blockchain, registri DLT nella loro generalità, i Directed Acyclic Graph e le tabelle hash distribuite (DHT). Il secondo capitolo introduce l'ambiente IOTA, con particolare riferimento alle componenti principali che lo compongono e al funzionamento delle varie parti. Il capitolo 3 tratta una delle funzioni principali di IOTA, il protocollo di comunicazione MAM. Il quarto capitolo invece si occupa di analizzare il problema e introdurre le scelte implementative prese per arrivare a una possibile soluzione. Il capitolo 5 tratta l'implementazione vera e propria del sistema, sia a livello di DHT, che di utilizzo delle API IOTA per la creazione di transazioni, canali MAM e comunicazione generica con il Tangle. Il capitolo 6 analizza quanto implementato ed espone gli sviluppi futuri del progetto. Nel capitolo 7 sono presenti invece le conclusioni finali.

Capitolo 1

Stato dell'arte

1.1 Blockchain

La Blockchain ha radicalmente cambiato l'idea di decentralizzazione. Questa struttura dati è stata introdotta per la prima volta dalla piattaforma Bitcoin nel 2008, quando Satoshi Nakamoto, pseudonimo per una persona o un gruppo di persone tutt'ora ignote, pubblicò il whitepaper *"Bitcoin: A Peer-to-Peer Electronic Cash System"* [1].

La Blockchain è un registro distribuito (DLT) contenente informazioni all'interno di blocchi concatenati, la cui integrità è garantita dalla crittografia. Una volta che un dato è registrato all'interno della blockchain e approvato, questo risulterà immodificabile, o molto difficile da modificare. Questo registro sfrutta i protocolli di comunicazione *Peer-to-Peer* e non si avvale di nessuna entità centrale di controllo.

I termini "blocco" e "catena", che compongono il nome Blockchain si riferiscono al modo in cui questa struttura i dati. Essa infatti è composta da una serie di blocchi legati tra loro tramite puntatori. Ogni blocco è essenzialmente un contenitore di transazioni e possiede un riferimento al blocco che lo precede, andando a creare così una catena di blocchi.

Le transazioni sono l'elemento costituente dei blocchi e rappresentano uno scambio di valuta da un portafoglio a un altro. Per portafoglio si intende l'insieme di due chiavi crittografiche asimmetriche, una privata e una pubblica. La prima permette di firmare una transazione e garantirne la provenienza, mentre la seconda funge da indirizzo a cui è possibile inviare valuta.

Come detto, la rete Peer-to-Peer che contorna una Blockchain non si avvale di nessuna autorità centrale al fine di mantenere una copia di riferimento del registro, infatti ogni nodo della rete può possedere una copia dei dati e può idealmente aggiungere un blocco alla catena. I blocchi sono referenziati l'uno all'altro, in maniera cronologica. Questo riferimento è ottenuto tramite una funzione di hash, per la precisione mediante l'utilizzo dell'algoritmo SHA-256, a partire da tutto il blocco di cui è necessario ottenere il digest. Questa specifica implementazione rende facile verificare l'integrità della catena stessa.

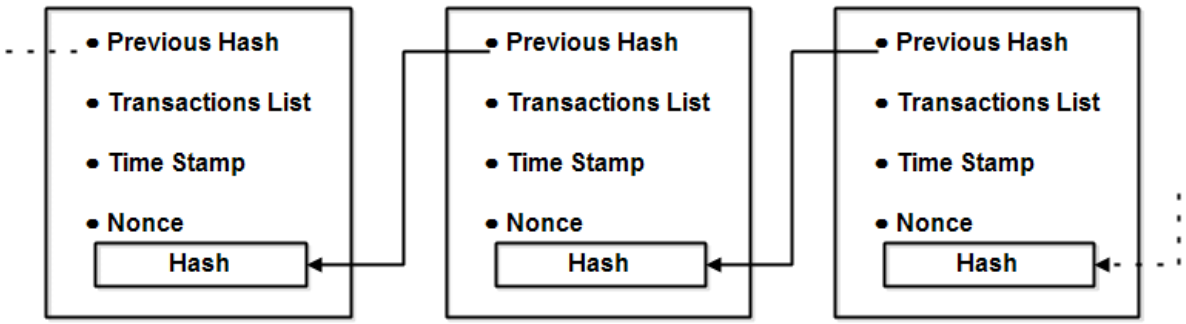

Figura 1.1: Referenziazione dei blocchi [2]

Proof of Work, mining e fees

Per aggiungere un blocco alla catena, un nodo deve risultare "leader" per quel turno di scrittura. In Bitcoin, un nodo viene eletto "leader" se completa un calcolo matematico denominato *Proof-of-Work*(POW), il quale consiste in una prova di forza bruta, richiedente un grande sforzo computazionale e di energia.

A svolgere la Proof-of-Work sono i cosiddetti nodi *miner*. Una volta che un miner completa la PoW, tenterà di aggiungere un nuovo blocco in cima alla catena. Quando questo accade, il blocco viene propagato nella rete e tutti i nodi connessi ne vengono a conoscenza, così da poter confermare o meno la validità dello stesso e delle transazioni contenute al suo interno.

L'operazione matematica ha come obbiettivo il trovare, tramite una funzione di hash, un valore definito come *nonce* che, concatenato all'hash e all'header del blocco precedente, restituisca come output un valore che soddisfi alcuni requisiti. Il requisito fondamentale è che il valore restituito sia minore di un *target*.

Il *target* dipende da un parametro chiamato *difficulty*, il quale è ricalcolato ogni 2016 blocchi per far in modo che il tempo per risolvere il puzzle sia circa costante, nell'ordine di una decina di minuti. In questo modo la quantità di calcolo necessaria, se un nodo malevolo volesse modificare blocchi già istanziati, sarebbe insostenibile. Perchè, ricordando la caratteristica per la quale i blocchi si concatenano l'un l'altro attraverso l'hash, modificare un blocco significherebbe dover modificare tutti i blocchi successivi e per ognuno calcolare di nuovo la relativa Proof-of-Work.

Quanto trattato finora non spiega perchè i miners dovrebbero sostenere operazioni di calcolo così complesse per completare la PoW. Il motivo è un reward costituito da una quota fissa in token e da fees delle transazioni che il nodo sta inserendo nel blocco. Attualmente il reward è pari a 12.5 BTC, ma il valore si dimezza ogni 210.000 blocchi, circa ogni 4 anni.

La fee di una transazione è il contributo dato ai miners per gestire quella transazione. Più alto è il valore della fee e più è appetibile per un miner prendersi carico di questa.

1.1.1 Evoluzione della Blockchain

Il registro blockchain ha introdotto fondamentali novità sul piano della decentralizzazione, infatti grazie alle sue caratteristiche, offre la possibilità di decentraliz-

zare ciò che prima di questa tecnologia doveva avvalersi di un'autorità centrale per garantire sicurezza e integrità. Si può individuare, nel corso degli anni, uno sviluppo crescente e un impiego sempre più trasversale della struttura a blocchi. In particolare, si è osservata la seguente evoluzione:

- Blockchain 1.0: impiego nel campo delle criptovalute. Dal 2008 con la nascita di Bitcoin e successivamente con la creazione di criptovalute alternative denominate Altcoin.

- Blockchain 2.0: introduzione degli smart contracts [3]. La blockchain non è più vista solo come un registro contenente transazioni. Introdotti dalla piattaforma Ethereum nel 2015, si parla di vere e proprie applicazioni finanziarie.

- Blockchain 3.0: registri distribuiti non necessariamente rivolti alle criptovalute. Solo alcuni esempi di possibili impieghi: voto elettronico, health care, gestione dell'identità, sistema notarile decentralizzato, supply chain, IoT.

1.1.2 Limiti della Blockchain

Scalabilità

Questa è forse la limitazione più grande del registro. Attualmente, la dimensione di un blocco in Bitcoin è circa di 1 Megabyte, questo significa che il numero di transazioni all'interno di un blocco è pressoché fisso e limitato. Considerato che, in media, un blocco è minato ogni 10 minuti (600 secondi), il numero di transazioni per secondo (TPS) si aggira tra 4 e 7, variabile a seconda della dimensione delle stesse. Valore molto lontano, ad esempio, da un circuito come Visa, capace di prendere in carico migliaia di transazioni al secondo.

Fee

Le fee sono un costo aggiuntivo da aggiungere a ogni transazione e tecnicamente sono rappresentate dalla differenza tra il valore in input e il valore in output di una transazione. A causa del limite di capienza di un blocco come discusso nella sezione precedente, i nodi *miners* devono decidere quali transazioni aggiungere a un blocco. Essi saranno ovviamente propensi a scegliere le transazioni con fees più elevate. Il rischio è quello per cui, a un eccessivo numero di transazioni effettuate in un determinato momento, si crei un collo di bottiglia per cui solo i trasferimenti con alte fee siano aggiunti ai nuovi blocchi, alzando a dismisura il costo delle stesse.

Storage

La struttura Blockchain, in particolare Bitcoin e implementazioni simili, non mantiene i bilanci degli account. La valuta disponibile a un account è calcolata in base alle transazioni passate, si parla in questo caso di Unspent Transaction Output (UTXO). E' necessario per questo mantenere l'intera catena di blocchi affinchè i nodi siano in grado di validare una transazione. La dimensione della blockchain è quindi destinata a crescere con il tempo e per ora misura circa 250 Gigabyte. Oltre alla grande quantità di spazio richiesta per immagazzinare la blockchain, la struttura richiede anche un grande dispendio di energia elettrica per svolgere operazioni come la Proof-of-Work.

Decentralizzazione centralizzata

La blockchain è nata come registro distribuito, ed effettivamente lo è, ma la grande corsa al mining e l'eccessivo dispendio di energie per completare la Proof-of-Work hanno fatto si che solamente poche entità possano competere nello svolgimento della PoW. Inoltre, queste stesse entità spesso preferiscono collaborare, creando le cosiddette *mining pool*, le quali aggregano la potenza di calcolo di

molte entità. Ecco che quello che era pensato come un sistema decentralizzato si rivela perciò in certi aspetti centralizzato nelle mani di pochi *miners*, non rendendo impossibile il cosiddetto attacco del 51%.

1.2 DAG e il Tangle

Successivamente all'affermazione della blockchain come registro distribuito, altre forme di registri pubblici distribuiti hanno preso forma. Tra questi si può individuare il cosiddetto Tangle [4]. Il Tangle si basa su una struttura dati denominata Directed Acyclic Graph (DAG). Questo tipo di struttura è appunto un grafo, il quale prevede la presenza di nodi collegati l'un l'altro, ma senza risultare di forma ciclica. Inoltre, il collegamento tra di essi ha una direzione, per questo l'aggettivo *directed*.

Il Tangle non prevede la presenza di blocchi. La Blockchain poteva essere vista come una linked list, nel Tangle tutte le transazioni sono contenute in un grafo e ogni transazione è un nodo di esso.

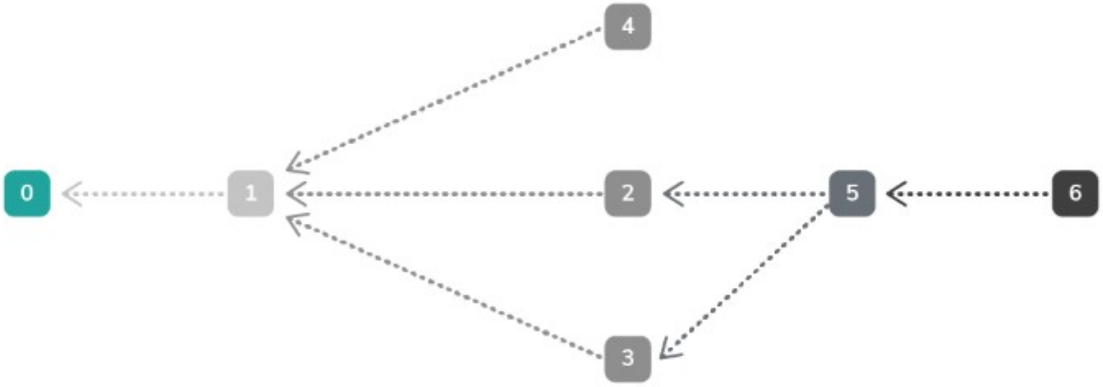

Figura 1.2: Un esempio di DAG [5]

Per rendere questo registro immutabile, ogni nuova transazione sul Tangle, denominata *tip*, è referenziata a due precedenti transazioni già all'interno del

grafo e contribuisce alla loro approvazione. Queste due *tip* sono visualizzabili all'interno dei dati della transazione stessa, sotto il nome di *branch* e *trunch* transaction. 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.

Ad esempio, si nota nell'immagine 1.2 come vi sia referenza diretta tra il nodo 6 e il nodo 5, e indiretta tra il nodo 6 e il numero 2.

Il modo in cui una nuova transazione sceglie due *tip* non è casuale, bensì 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.

Il Tangle e il registro IOTA saranno approfonditi nel dettaglio all'interno dei capitoli 2 e 3.

1.3 DHT: Distributed Hash Table

Una distributed hash table (DHT) è un sistema distribuito, su rete Peer-to-Peer, dal funzionamento simile a quello di un hash table. Come quest'ultima infatti, una DHT gestisce dati sotto forma di coppie *(chiave, valore)*, ma in maniera distribuita tra vari nodi, i quali formano un network. Ognuno di questi riferimenti chiave-valore è immagazzinato nella DHT e ogni nodo partecipante alla rete può ricavare il valore associato a una chiave, dando in input la chiave stessa. La particolare struttura della rete DHT garantisce decentralizzazione, scalabilità e, in alcune implementazioni, bilanciamento del carico e integrità dei dati. Inoltre la struttura è indipendente dalla tipologia di dato gestita.

I primi utilizzi delle tabelle hash distribuite risalgono ai primi anni 2000 all'interno di grandi network Peer-to-Peer quali BitTorrent, Napster e Gnutella. Tra gli esempi di algoritmi DHT si trovano Chord [6], Pastry [7], Tapestry [8].

Di fondamentale importanza è la gestione delle chiavi K. In una hash table tradizionale, le coppie chiave-valore sono gestite tutte nella stessa tabella. In una DHT invece, ogni nodo gestisce in maniera autonoma uno spazio di chiavi, non la totalità di queste. Questa tecnica prende il nome di partizionamento del keyspace, dove per keyspace si intende l'insieme di tutte le possibili chiavi all'interno della rete. Un algoritmo di partizionamento spezza il keyspace in tanti set su tutti i nodi partecipanti alla rete, affidando così la responsabilità di un set a un preciso nodo. Esso si occuperà dell'indicizzazione dei file che hanno chiave contenuta all'interno del suo set di riferimento.

All'interno della rete, ogni nodo mantiene un set di link ad altri nodi detti vicini o neighbors, secondo una particolare topologia del network. L'insieme di tutti i nodi forma il cosiddetto overlay network. Comune a tutte le topologie è la proprietà per cui, per ciascuna chiave k, il nodo la gestisce direttamente oppure ha tra i suoi neighbors un nodo più vicino a k in base al partizionamento delle chiavi. Questa proprietà influisce sul routing quando si è alla ricerca di un determinato nodo nell'overlay, dato che, passando nodo per nodo, si viene reindirizzati sempre più verso il nodo cercato, ottimizzando il routing.

Un esempio sul funzionamento. Per indicizzare un file con un dato *filename* e *data*, viene inizialmente ottenuta la chiave k del file tramite la funzione *hash(filename)*. Successivamente un messaggio *put(k,data)* è inviato lungo il network verso il responsabile della particolare chiave che immagazzinerà la coppia. A questo punto un altro nodo potrà richiedere il file con una richiesta *get(k)*, avendo calcolato in precedenza l'hash del filename. La richiesta verrà inoltrata nell'overlay verso il nodo responsabile per k, il quale risponderà con una copia del dato.

La grande limitazione di questa richiesta è il fatto di dover conoscere a priori il *filename*, per poterne calcolare l'hash e ricavarne di conseguenza k. Spesso questa informazione non la si ha e si preferirebbe una ricerca per parole chiave, le quali permetterebbero di ottenere risultati concernenti la ricerca senza avere

esattamente il nome esatto. Vi sono molti sistemi che cercano di risolvere il problema del keyword search all'interno di una tabella di hash distribuita. Tra le varie soluzioni, nel corso della tesi è stata presa come riferimento una struttura a ipercubo [9]. Questa topologia prevede che i nodi siano rappresentati all'interno di un ipercubo a r-dimensioni e abbiano come id una stringa composta da r-bit. Essi hanno come neighbors solamente i nodi aventi id differenti dal loro di un solo bit. Mappando ogni oggetto in un vettore a r-bit, in relazione al suo set di keyword, è possibile assegnare la responsabilità di un oggetto a un solo nodo. Prima di approfondire la struttura a ipercubo e applicarla a una tabella hash distribuita, verrà trattato nei seguenti capitoli il registro distribuito IOTA, fulcro della dissertazione.

Capitolo 2

IOTA

2.1 Introduzione a IOTA

Figura 2.1: Il logo IOTA

IOTA è un progetto nato a fine 2015 da Sergey Ivancheglo, Dominik Schiener, David Sønstebø e Serguei Popov, allo scopo di fornire una valida alternativa al registro Blockchain nel campo dell'IoT. Come definito nel capitolo 1, IOTA consiste in un registro distribuito permissionless e dalla criptovaluta omonima [5]. 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 denominato Tangle. Sul Tangle vengono inoltrate e raccolte tutte le transazioni del network. Queste vengono ispezionate da una serie di attori per verificarne integrità e 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. Ogni nodo ha dei vicini, detti anche neighbors, proprietà comune a tutti i registri distribuiti. 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.

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 rivelarlo. 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*.

2.1.1 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. E' 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.

Il motivo d'uso dell'aritmetica ternaria ancora sfugge, essendo il sistema binario in uso sulla totalità dei processori presenti sul mercato. IOTA afferma che il sistema ternario è di maggiore efficienza perchè permette di rappresentare i dati in 3 stati piuttosto che 2.

2.1.2 Caratteristiche principali

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.

Scalabile

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.

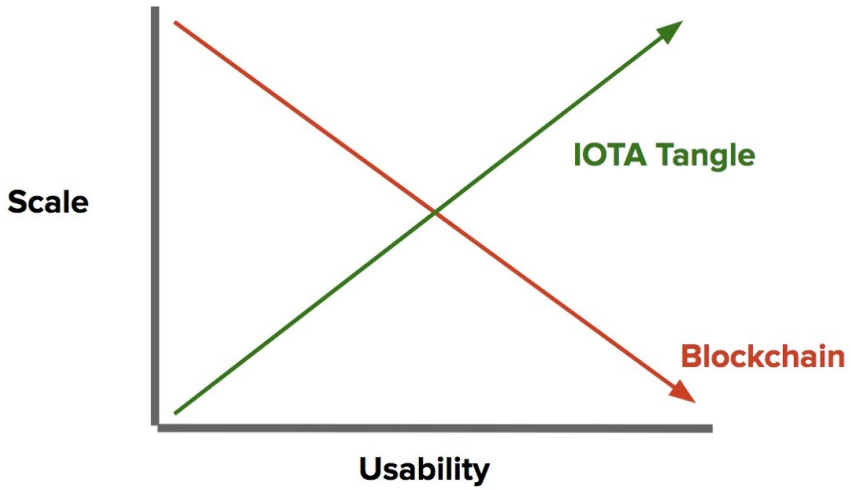

Figura 2.2: Scalabilità di IOTA https://medium.com/coinmonks/tangle-the-missing-link-for-iot-and-web-3-0-6519f22ebc81

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

Zero fee

IOTA non presenta nessun collo di bottiglia per quanto riguarda l'immagazzinamento di transazioni nel Tangle e non è prevista la figura del *miner*. In compenso, al contrario della tecnologia blockchain in cui solamente i *miner* svolgono la Proof-of-Work, in IOTA chiunque carichi una transazione deve svolgere la PoW. Questo è il meccanismo con cui si sostituisce il pagamento delle fee in IOTA, dove non è previsto nessun tipo di pagamento ulteriore.

Micro pagamenti

Data l'assenza di fee nel sistema, IOTA supporta la possibilità di effettuare transazioni contenenti pochissima valuta e addirittura permette la creazione di transazioni zero-value.

Quantum resistance

IOTA utilizza per la firma l'algoritmo Winternitz (W-OTS), denominato One-Time Signature. Questo ha la caratteristica di essere resistente agli attacchi effettuati da computer quantistici, ma ha il problema di rivelare parte della chiave ogni volta che è utilizzato. Per questo motivo è necessario ogni volta che si effettua una transazione cambiare chiave e di conseguenza cambiare indirizzo, come si vedrà in seguito.

2.2 Il Tangle

Tangle [4] è il nome dato alla struttura dati che costituisce il registro distribuito di IOTA. Si basa su un directed acyclic graph (DAG), composto da vertici e da archi. Un vertice, detto anche nodo, rappresenta una singola transazione, mentre un arco rappresenta un collegamento tra due nodi e ha una direzione. Come definito nel paragrafo 1.2, ogni nuova transazione sul Tangle, contribuisce

all'approvazione di due precedenti *tip*. L'approvazione si definisce diretta se esiste un arco che collega le due transazioni, mentre indiretta se esiste un percorso che permette di arrivare da una transazione all'altra.

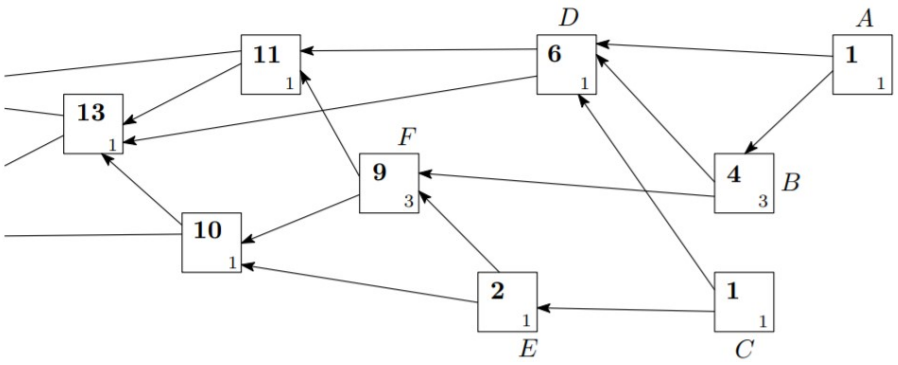

Figura 2.3: Esempio di Tangle [4].

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, svolgere una *Proof-of-Work*, simile a quella di Bitcoin ma molto meno dispendiosa.

2.2.1 Weight e altri valori

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.

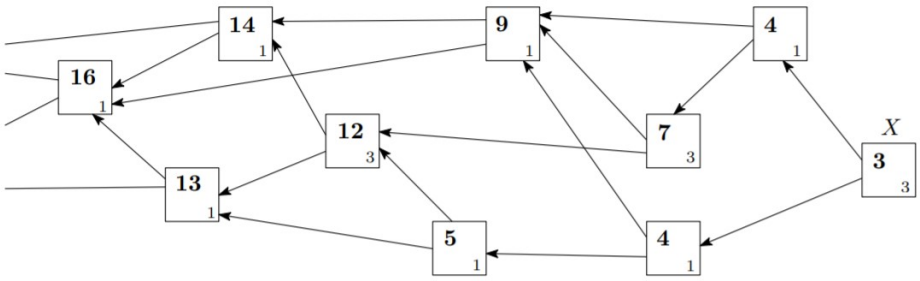

Figura 2.4: Pesi di una transazione nel Tangle[4].

Si nota in figura un esempio di Tangle. Ogni nodo è rappresentato da un quadrato e ha due valori, in basso a destra il parametro *weight* e in alto a sinistra il parametro *cumulative weight*. Confrontando la figura 2.3 con la figura 2.4, si nota come la nuova transazione X approvi due *tip*, causando un aumento del loro *cumulative weight*.

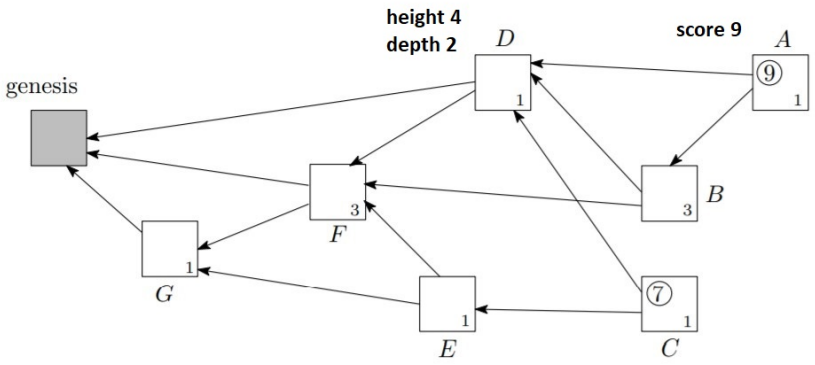

Figura 2.5: Height, depth e score [4].

Si introducono inoltre altri tre parametri per una transazione: *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.

2.3 Nodi

I nodi sono il nucleo di un network IOTA. Come già affermato, permettono l'accesso al Tangle, rendono cioè possibile la lettura e la scrittura nel registro. Sono stati precedentemente definiti i neighbors, ovvero i vicini che un nodo possiede e ai quali invia le nuove transazioni che carica sul Tangle. Quando un altro nodo riceve la transazione, esso la aggiungerà alla sua copia del registro e potrà convalidarla. I nodi mantengono anche un archivio contenente tutti gli indirizzi con un saldo maggiore di 0.

2.3.1 Validazione di una transazione

Per validare una transazione, un nodo deve controllare alcuni aspetti di essa. Innanzitutto, verifica che sia stata svolta la Proof-of-Work, in accordo con la Minimum Weight Magnitude (MWM) specificata.

Minimum Weight Magnitude

La MWM è una variabile, specificabile dall'utente durante la creazione di una transazione, che determina il lavoro minimo svolto durante lo svolgimento della PoW. Questo parametro ha un valore minimo richiesto e più è alto, più è complicato portare a termine la Proof-of-Work.

Oltre a ciò, quando un nodo verifica una transazione esegue anche altri controlli. Ispeziona la validità di tutte le firme, controlla che il *value* di ogni transazione

non ecceda il numero di *token* globale e controlla che tutti i token in uscita siano depositati in altri indirizzi.

2.3.2 Il Coordinator

Attualmente IOTA fa uso di una componente chiamata Coordinator. Scopo di esso è emettere periodicamente una transazione definita Milestone. Una Milestone, come tutte le altre transazioni è referenziata ad altre due nel Tangle. Quando la transazione Milestone approva due transazioni, direttamente o indirettamente, queste sono confermate. Il Coordinator è stato implementato nella fase iniziale di IOTA. Esiste un programma, denominato *Coordicide* [10], il cui scopo è quello di rimuovere il Coordinator, lasciando ai nodi la responsabilità di raggiungere il consenso definitivo.

2.3.3 Selezione delle *tip*

Per ogni transazione caricata nel Tangle è necessario approvare due *tip*. Per selezionare due *tip* da approvare, il nodo IOTA responsabile del caricamento della transazione seleziona un sub-grafo, ovvero una parte del Tangle, ed esegue due cammini casuali in esso. Il sub-grafo va da una Milestone a un gruppo di tip. I due cammini casuali svolti dovrebbero ritornare due transazioni, le quali rappresentano le due transazioni da approvare. I cammini avvengono secondo l'algoritmo Markov Chain Monte Carlo.

2.4 Client

I client sono coloro che si appoggiano ai nodi della rete IOTA per eseguire operazioni di lettura e scrittura all'interno del Tangle. Un client ha una chiave segreta denominata *seed*, una stringa composta da 81 trytes. Attraverso il proprio seed

un client riesce a derivare degli indirizzi denominati *address*, anch'essi stringhe da 81 trytes.

2.4.1 Address

Un indirizzo è come un account, esso appartiene a un seed e possiede un discreto numero di IOTA tokens. Questa stringa è la metà pubblica di una coppia di chiavi pubblica/privata. Come avviene in altre applicazioni, per inviare una transazione da un indirizzo a un altro, la transazione è firmata con la chiave privata, di cui solo il proprietario del seed è a conoscenza. La parte pubblica, cioè l'indirizzo, è comunque condivisibile perchè non permette di risalire né alla chiave privata, né al seed.

Un address è creato a partire da un *seed*, da un valore *index* e da un livello di sicurezza. A un *seed* possono appartenere circa 9^{57} address, uno per ogni valore di *index*, il quale varia proprio da 0 al valore sopracitato. Il livello di sicurezza invece varia da 1 a 3 e specifica appunto l'omonima caratteristica di un address.

Spent address

IOTA utilizza lo schema di firma Winternitz (W-OTS), detta One Time Signature. Questo schema è progettato per resistere ad attacchi svolti da computer quantistici, tuttavia, presenta lo svantaggio di essere sicuro solamente una volta. Per questo motivo un indirizzo, dopo essere stato utilizzato, viene classificato come *spent*. Se dei token dovessero essere depositati su un indirizzo esaurito, questi rischierebbero di essere sottratti mediante un attacco di forza bruta alla chiave privata. Questo è il motivo per cui in IOTA a un seed possono appartenere molti indirizzi diversi tra loro.

2.5 Transazioni

Una transazione [11] è un trasferimento di dati da un client a un nodo della rete, il quale è incaricato di attaccare la stessa al Tangle ed eseguire le operazioni di cui discusso in precedenza. Una transazione può rappresentare un trasferimento di tokens da un address a un altro, oppure potrebbe contenere solamente dati, un messaggio o una firma. In questi ultimi casi, quando non è previsto uno scambio di token, la transazione è denominata zero-value. Essa è composta da 2673 trytes, suddivisi per diversi campi. Inoltre una transazione può far parte di un pacchetto di transazioni, denominato bundle. Di seguito se ne riporta la struttura.

Struttura di una transazione

- **hash**: stringa hash derivata dal valore di ogni campo della transazione e dalla PoW (81 trytes)
- **signatureMessageFragment**: Una firma o un messaggio, entrambi possono essere frammentati in un bundle. Il campo contiene il trytes 9 quando nulla è definito (2187 trytes)
- **address**: Contiene l'address di chi invia o di chi riceve tokens. (81 trytes)
- **value**: Quantità di IOTA tokens scambiata. (27 trytes)
- **timestamp**: Unix epoch, a partire dall' 1 gennaio 1970 (9 trytes)
- **currentIndex**: Indice della transazione corrente in un bundle (9 trytes)
- **lastIndex**: Indice dell'ultima transazione in un bundle (9 trytes)
- **bundle**: Hash del bundle, calcolato da alcuni campi: address, value, obsoleteTag, currentIndex, lastIndex, and timestamp (81 trytes)

- **trunkTransaction**: Hash di una transazione esistente nel tangle che la transazione corrente referenzia, oppure la transazione successiva nel bundle (81 trytes)
- **branchTransaction**: Hash di una transazione esistente nel Tangle che la transazione corrente referenzia (81 trytes)
- **attachmentTag**: Tag definito dall'utente (27 trytes)
- **nonce**: Rappresentano la Proof-of-Work (27 trytes)

2.5.1 Bundle

Un bundle, letteralmente pacchetto, è un gruppo di transazioni le quali si referenziano le une alle altre. Un bundle è costituito da un head, un body e una tail. La tail ha *currentIndex* pari a 0, mentre head ha *currentIndex* più alto nel bundle. Tutte le transazioni nel bundle sono referenziate mediante il campo *trunkTransaction*. Anche per i bundle possono essere identificati due tipi, transfer bundle e zero-value. I primi prevedono il trasferimento di token da un indirizzo a un altro, mentre i secondi no.

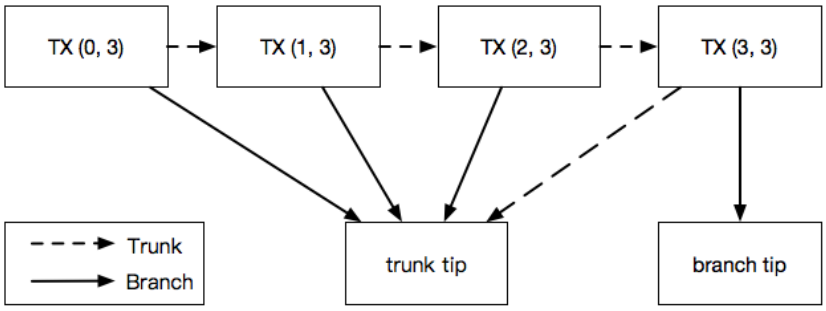

Figura 2.6: Transazioni in un bundle [5]

Si osserva in figura 2.6, a partire dalla prima transazione, come questa abbia index 0, ovvero è la parte *tail* del bundle. Il parametro *trunkTransaction* collega

la transazione 0 alla transazione 1 del Bundle, mentre *branchTransaction*, l'altro parametro di referenziazione, la collega a un'altra transazione nel Tangle. Si prosegue in questo modo fino alla transazione 3, la parte *head* del bundle.

2.6 Snapshot e Permanode

IOTA è un registro distribuito permissionless, questo significa che chiunque, in qualsiasi momento, può depositare sul Tangle una moltitudine di dati. Oltretutto essendo IOTA zero fee, non esistono limiti alle transazioni che un utente può creare, fatta eccezione per quel piccolo lavoro svolto per la Proof-of-Work. Ne consegue che la dimensione del Tangle sia sempre in crescita, come succede per altre criptovalute come Bitcoin. Per ovviare al problema e far in modo che i nodi rimangano leggeri e veloci, IOTA e i propri nodi svolgono periodicamente un'operazione denominata *snapshot*. Quest'operazione consiste nell'eseguire un'istantanea del Tangle, raccogliere tutti gli address con bilancio maggiore di zero ed eliminare tutto il resto. In questo modo i nodi possono sincronizzarsi molto velocemente in quanto il Tangle contiene molti meno dati.

Se da una parte è comodo per i nodi eliminare molti dati dalle proprie strutture, dall'altra un registro distribuito è concepito per essere immutabile e per mantenere i dati immessi. Per questo motivo nel registro sono presenti anche dei nodi denominati Permanode, i quali non sono soggetti a snapshot. Un permanode immagazzina tutta la storia e quindi tutte le transazioni del Tangle dalla *genesis*, rendendo possibile così la ricerca di dati di ogni periodo sul registro.

Capitolo 3

MAM: Masked Authenticated Message

3.1 Introduzione

Masked Autheticated Message (MAM) è un protocollo di comunicazione dati che consente di pubblicare flussi di dati criptati, denominati channel, mediante l'uso di transazioni sul Tangle. La caratteristica di IOTA di essere zero-fee si coniuga bene con il protocollo MAM, in quanto permette a client o dispositivi di varia natura di creare canali su cui poter inserire messaggi cifrati e a un ipotetico subscriber di usufruirne, proprio come accade per un canale YouTube. La cifratura e di conseguenza la proprietà di un MAM channel è data dal conoscere il *seed* del client.

La struttura di un MAM channel è vista come una catena di messaggi MAM. Ogni messaggio ha un identificativo preciso denominato *root* e conosce l'indirizzo *root* del successore, tramite il parametro *nextRoot*. Non conosce, invece, il messaggio precedente della catena. Non è possibile infatti accedere a quest'ultimo se non conoscendone la *root*.

3.2 Tipologie di canali

Vi sono 3 tipologie diverse di channel: Public, Private e Resticted. La differenza tra essi sta nel livello di visibilità e cifratura che questi hanno.

3.2.1 Public

Un canale Public può essere visualizzato da qualsiasi utente che possieda la *root* di un messaggio. Da quel messaggio preciso in poi, egli potrà rimanere in ascolto per altri messaggi caricati nel canale, perché decifrando il messaggio verrà a conoscenza della *nextRoot*, la prossima root. In un Public channel la stringa *root* rappresenta sia l'indirizzo di un messaggio, sia la chiave di cifratura e decifratura.

3.2.2 Private

Un canale Private è visualizzabile e decifrabile solamente dal possessore del *seed*, quindi solo dal proprietario del canale. L'indirizzo del messaggio è rappresentato dall'hash(*root*).

3.2.3 Restricted

Un canale Restricted ha come indirizzo hash(*root*), ma è decifrabile solamente da chi possiede una chiave denominata *sideKey*. In questo caso è il proprietario del canale che decide sia la sideKey, sia a chi fornirla, concordando così l'accesso ai dati.

3.3 Merkle tree

Un merkle tree è un struttura ad albero binario le cui foglie sono rappresentato dall'hash del loro contenuto, mentre ogni nodo non foglia è rappresentato dall'hash dei due contenuti dei figli (in questo caso anch'essi hash).

MAM utilizza il merkle tree. La root di un messaggio MAM è creata utilizzando un merkle tree a partire da alcuni indirizzi generati dal seed. Di conseguenza si capisce come il seed sia fondamentale.

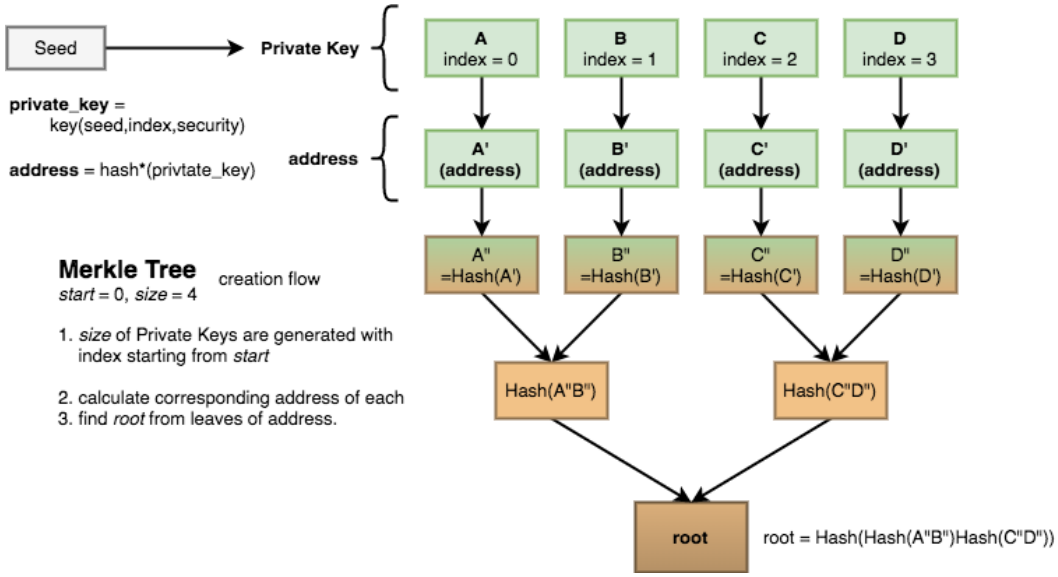

Figura 3.1: Rappresentazione di un Merkle tree [12]

Per generare l'albero Merkle, sono richiesti in input due parametri, il primo *start* rappresenta l'indice da cui partire a generare indirizzi dato un seed, mentre il secondo rappresenta il numero di indirizzi da generare e di conseguenza anche la dimensione dell'albero ottenuto.

Attraverso il seed genero gli indirizzi e le relative chiavi private che rappresentano le foglie dell'albero, come detto in precedenza un seed può avere più indirizzi modificando il parametro index.

Le chiavi private A, B, C, D rappresentano le foglie dell'albero, mentre A', B', C', D' rappresentano gli indirizzi corrispondenti. Gli indirizzi sono poi sottoposti ad hash, andando a creare i primi nodi hash dell'albero (A",..., D"). Creo, facendo hash su due nodi, ulteriori nodi intermedi dell'albero, fino ad arrivare via via verso la radice, definita *root*.

3.4 Struttura di un messaggio

Un messaggio MAM è composto da un bundle di almeno 3 messaggi. Due messaggi sono necessari per contenere la firma e almeno un altro per contenere il messaggio vero e proprio. La sezione che contiene la firma è denominata Signature section, mentre l'altra è denominata MAM section. Sono entrambe contenute nel signatureMessageFragment, uno dei parametri della transazione visti in precedenza.

3.4.1 MAM section

La parte MAM section è criptata e in base al tipo di canale richiederà una chiave di decriptazione diversa. In Public mode la chiave sarà *root*, mentre in Restricted sarà la *sideKey*. La sezione MAM contiene diverse informazioni: nextRoot, branchIndex, siblings.

nextRoot

Il parametro nextRoot è fondamentale all'interno di un messaggio MAM. Esso collega un messaggio con il suo successivo. Come visto precedentemente una root è generata a partire da n address generanti un Merkle tree. Si evince quindi che per creare un messaggio MAM, il quale deve contenere il parametro *nextRoot*, è necessario generare due Merkle tree, in modo tale da generare due root, rispettivamente la *root* del messaggio vero e proprio e la *nextRoot* del messaggio successivo.

branchIndex

Il parametro *branchIndex* è un valore scelto tra le foglie del Merkle tree. In riferimento alla figura 3.1, l'albero è generato a partire da address con indice da 0 a 4, *branchIndex* ha uno di questi possibili valori.

siblings

Il parametro siblings è in realtà un set di valori. La particolarità del Merkle tree è quella di poter risalire alla radice dell'albero avendo solo una piccola quantità di nodi, a causa della concatenazione di hash. A partire dall'address di *branchIndex*, siblings contiene i valori hash di alcuni nodi che combinati insieme all'address possono generare la *root*. In riferimento alla figura 3.1, preso un *branchIndex* 0, i siblings per ottenere la *root* sono B" e Hash(C"D").

3.5 Lettura di un messaggio

Per leggere un messaggio ed eventualmente quelli successivi a esso è necessario conoscere la *root* del canale. In caso il canale sia di tipo restricted sarà necessaria anche la *sideKey*.

Prima di tutto la *root* è convertita in *address*. Una volta trovato il messaggio, questo è decriptato a seconda del tipo di canale, se public la chiave di decriptazione è la *root* stessa, se restricted si usa la *sideKey*.

Una volta decriptati i dati questi conterranno il messaggio vero e proprio, *nextRoot*, *branchIndex* e *siblings*.

3.5.1 Verifica autenticità

Per verificare l'autenticità di un messaggio MAM è necessario validare la firma. Dopo questo processo di validazione, si ha come valore di ritorno un *address*, il quale rappresenta una delle foglie del merkle tree, esattamente quello dove index = *branchIndex*. Combinando questo valore con i siblings si ottiene una root, denominata *temp_root*. Successivamente avverrà il confronto tra *root* e *temp_root*. Se uguali, si avrà la certezza che il messaggio MAM è valido.

Capitolo 4

Query su infrastruttura MAM

4.1 Analisi del problema

Il protocollo *Masked Authenticated Message* permette la creazione e la sottoscrizione a canali intesi come stream di dati, che da un dispositivo confluiscono nel Tangle. Per ricevere dati provenienti da un determinato canale però, è necessario conoscere esattamente la sua Root, ovvero il suo indirizzo.

In un contesto in cui il dato è creato allo scopo di rimanere circoscritto al creatore o a una ristretta cerchia di utenti specifici, questo meccanismo non ha limiti di sorta, soprattutto se associato a channel di tipo *restricted* o *private*.

Al contrario, in un contesto in cui le informazioni sono caricate sul Tangle non solo a fini privati, ma per la distribuzione, indifferentemente se gratuita o a pagamento, dover conoscere l'indirizzo preciso di un canale risulta limitante. Un utente non può cercare informazioni sulla rete IOTA, se non inserendo indirizzi precisi di transazioni o canali, fatta eccezione per il parametro tag, discusso in seguito. Risulta perciò difficile interrogare il Tangle per reperire dati, non potendo utilizzare qualche specifico filtro di ricerca in grado di organizzare le informazioni, allo stesso modo in cui succede nelle DHT descritte nel capitolo 1.2.

Utilizzando le API open-source del registro IOTA è possibile creare transazioni e messaggi MAM arricchite da un parametro TAG. Questa voce è una tra le tante che compongono una transazione, ed è definita da una stringa di 27 trytes.

Attraverso le librerie è altresì possibile interrogare il Tangle affinché restituisca tutte le transazioni appartenenti a un TAG. Questo scenario è sicuramente il più semplice a livello implementativo ma presenta un problema: sarebbe molto semplice per un utente malintenzionato inserire transazioni spam con un determinato TAG. I risultati che si otterrebbero da una query verso il Tangle sarebbero perciò distorti.

Per questo motivo in questo testo è presentato un overlay, che riesca a indicizzare le informazioni all'interno del Tangle e le restituisca quando richieste. Questa struttura è rappresentata da una Distributed Hash Table ottimizzata per la ricerca keyword-based attraverso una topologia a ipercubo. L'idea è quella di mappare ogni transazione in un vettore a r-bit, in accordo con il suo set di keyword, e vedere questi vettori a r-bit come punti di un ipercubo a r-dimensioni [9]. La rete DHT quindi, sarà essenzialmente una struttura overlay contenente oggetti. Questi oggetti rappresenteranno effettivamente gli specifici indirizzi MAM, associati a un tag e inseriti appositamente all'interno della rete. All'interno del testo, nel contesto DHT, fare riferimento a oggetti o a transazioni è equivalente, dato che gli oggetti gestiti dalla tabella hash distribuita sono rappresentazioni di transazioni IOTA.

Sviluppando la struttura su un modello a ipercubo ogni singolo oggetto sarà indicizzato da un solo nodo e sarà facile determinarlo a partire dal set di keyword associato. Una volta definito un set di keyword, si potrà determinare univocamente il nodo che se ne occupa. Inoltre, la struttura a ipercubo garantisce che oggetti contenuti nella rete con set di keyword simili, siano gestiti da nodi vicini tra di loro. Quest'ultima è una proprietà molto importante che permetterà un routing tra nodi molto veloce, nel caso si decidesse di volere via via risultati più precisi.

4.2 Keyword search su DHT network

Come anticipato nel capitolo 1.3, la ricerca, l'inserimento e l'eliminazione di dati su rete DHT necessitano di un parametro specifico, definito come id dell'oggetto. Nell'architettura presentata le tre operazioni sopra riportate saranno gestite dalle sole keyword, perché sarà il nodo di riferimento a occuparsi dell'id specifico degli oggetti.

Per keyword search si intende ovviamente una ricerca per parole chiave. Si assume che ogni oggetto $\sigma \in \mathcal{O}$ nel sistema è associato a un keyword set K_σ. Per ogni oggetto σ, si dice che un keyword set K descrive σ se $K \subseteq K_\sigma$. Come da modello di riferimento[9], il punto di partenza per il servizio è sempre il *keyword* set. Sono stati sviluppati due tipi di ricerca: **Pin Search** e **Superset Search**.

Il primo consiste in una ricerca puntuale del set K specificato. Il servizio cioè ritornerà un set di oggetti $(\sigma_1...\sigma_n)$ descritti esattamente dal *keyword* set.

Il secondo, dato in input un set K e una soglia c, ritornerà un set di oggetti $(\sigma_1...\sigma_c)$ composto da almeno c risultati rappresentabili dal set K.

Prima di presentare il modo in cui questi due servizi sono stati progettati e poi implementati, è necessario presentare il modello su cui la Distributed Hash Table in questione è basata: l'ipercubo.

4.2.1 Ipercubo

Per ipercubo si intende quella forma geometrica regolare rappresentata in uno spazio di 4 o più dimensioni. Un particolare uso si trova all'interno della teoria dei grafi [13], infatti, un ipercubo di dimensioni r, può essere rappresentato da un grafo composto da 2^r nodi. Ognuno dei nodi u è costituito da un vettore binario di dimensione r. Con $u[i], 0 \leq i \leq r-1$ si intende l'i-esimo bit del nodo u a partire da destra.

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 collegano quest'ultimi. Un arco

edge(u,v) collega u con v se, e solo se u differisce da v di uno e un solo bit.

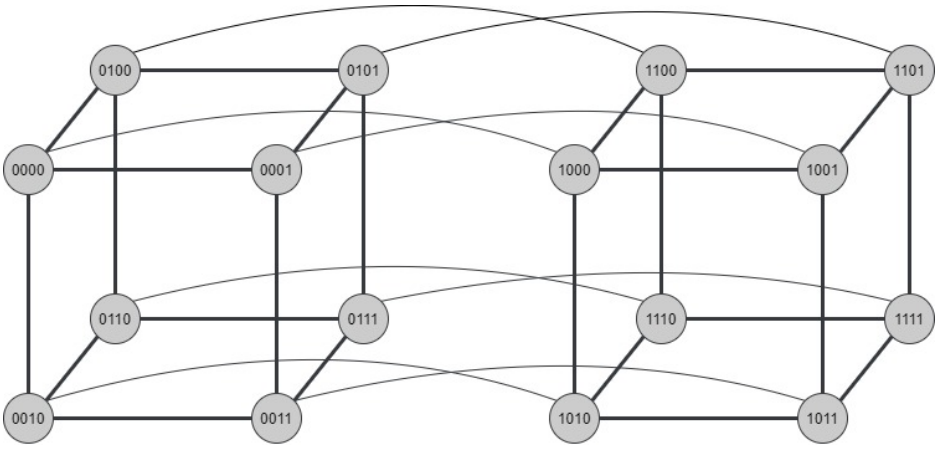

Figura 4.1: Rappresentazione di un ipercubo H_4

Ad esempio, si faccia riferimento a un ipercubo a 4 dimensioni. Ogni nodo u è rappresentato da una stringa di 4 bit ed è presente un collegamento fra due nodi solamente se questi differiscono di un solo bit, ad esempio, tra il nodo $v = 1011$ e $u = 1010$ esiste un collegamento (u, v).

Si definiscono inoltre i due insiemi *One(u)* e *Zero(u)* riguardanti il nodo u. $One(u) = \{i \mid u[i] = 1, 0 \leq i \leq r-1\}$ e $Zero(u) = \{i \mid u[i] = 0, 0 \leq i \leq r-1\}$. Rispettivamente *One(u)* rappresenta le posizioni dei bit con valore 1 del nodo u, mentre *Zero(u)* le posizioni con valore 0. Con riferimento all'esempio precedente, il nodo $u = 1010$ ha $One(u) = \{1, 3\}$ e $Zero(u) = \{0, 2\}$. Si dice che v contiene u se e solo se $One(u) \subseteq One(v)$.

Si definisce sub-ipercubo indotto da u, chiamato $\mathcal{H}_r(u)$, 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. In figura 4.2 è rappresentato un sub-ipercubo indotto dal nodo 0100, il quale può essere chiamato $\mathcal{H}_4(0100)$.

Spanning Binomial Tree

Lo Spanning Binomial Tree (SBT) è una struttura dati ad albero di estrema importanza per la ricerca di oggetti all'interno della rete DHT progettata. Si basa sul sub-ipercubo $\mathcal{H}_r(u)$ definito in precedenza, in quanto contiene esattamente gli stessi nodi e ha come radice dell'albero u.

Un *SBT* che ha come radice u, ha come nodi a livello 1, cioè sono figli di u, tutti quei nodi che contengono u e si discostano da esso solo di un bit, un nodo compare una e una sola volta all'interno dell'albero. La regola è ripartita per ogni nodo su qualsiasi livello dell'SBT. Nella figura si può osservare l'albero *SBT(u)* costruito a partire dal nodo $u = 0100$.

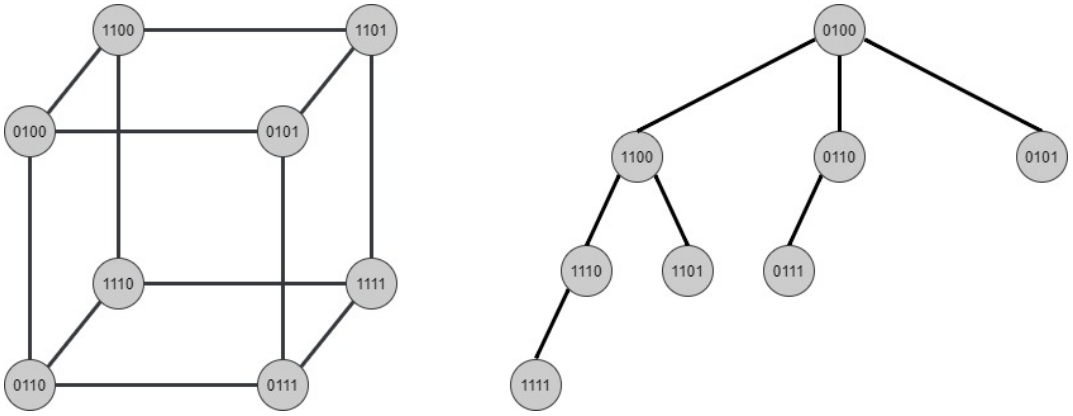

Figura 4.2: Il sub-ipercubo $\mathcal{H}_4(0100)$ e il relativo $SBT(0100)$

Si introduce inoltre la definizione di distanza di Hamming. La distanza di Hamming tra due stringhe $u = \{u_i, u_{i-1}, ..., u_0\}$ e $v = \{v_i, v_{i-1}, ..., v_0\}$ con $0 \leq i \leq r-1$, è il numero di posizioni i in cui $u_i \neq v_i$. Nel caso di due stringhe binarie, la distanza di Hamming è rappresentata dal numero di bit a 1 successivamente a un'operazione di XOR. $Hamming(u, v) = \sum_{i=0}^{r-1}(u[i] \bigoplus v[i])$
All'interno di un generico *SBT(u)*, il nodo v posizionato al j-esimo livello dell'albero avrà distanza di Hamming j dal nodo radice u.

4.2.2 La struttura ipercubo su DHT

Dopo aver introdotto l'ipercubo, il sub-ipercubo e l'albero SBT è necessario stabilire il modo in cui queste strutture si integrano con il network DHT. Una distributed hash table è costituita da nodi, che non per forza devono rappresentare esattamente tutti i nodi dell'ipercubo. Ad esempio, i nodi logici (quelli dell'ipercubo) potrebbero essere di più dei nodi fisici(quelli della rete DHT), o il contrario. Per semplicità, il modello è stato definito prevedendo per ogni nodo dell'ipercubo un nodo nella rete DHT.

A ogni modo, questa semplificazione potrebbe essere omessa e, nella situazione in cui vi siano più nodi logici che fisici, bisognerebbe mappare i nodi dell'ipercubo a quelli della DHT, in uno scenario in cui un nodo fisico si occuperebbe di più nodi logici. Nella situazione in cui, invece, vi fossero più nodi fisici rispetto a quelli logici, alcuni nodi fisici non parteciperebbero al processo di indicizzazione dei dati, ovvero, non si occuperebbero di nessun set di keyword.

Sia W il set di tutte le keyword considerate nel sistema. Ogni keyword è mappata in un numero intero da 0 a r-1 attraverso la funzione $h : W \rightarrow \{0, 1, ..., r-1\}$. Si introduce inoltre una mappatura che consente di determinare il nodo responsabile a partire da un set di parole chiave: $F_h(K) = u$ se e solo se $One(u) = \{h(w)|w \in K\}$. In altre parole, il nodo responsabile per il set è quello che ha i bit settati (a 1) nella stessa posizione ricavata con la funzione h. Il set $One(u)$ deve equivalere al set di interi ricavato con h.

Di seguito è illustrato un esempio sviluppato attorno a un ipercubo a 4 dimensioni, quindi $r = 4$. Si introduca il set di keyword $W = \{a,\ b,\ c,\ d\}$, questo deve essere mappato in interi da 0 a *r-1*. Per semplicità, ogni keyword verrà mappata con il numero della sua stessa posizione all'interno del set. Ad esempio, per il keyword set $\{a,\ c,\ d\}$ si ottengono $h(a) = 0$, $h(c) = 2$, $h(d) = 3$, considerando che la lettura dei bit di qualsiasi nodo dell'ipercubo parte da destra, si determina semplicemente il nodo responsabile per il set: 1101.

4.2.3 Le principali operazioni all'interno dell'ipercubo

Inserimento

L'inserimento di un oggetto σ nella rete richiede in input i dati e le keyword K_σ a esso associate. Il nodo che avvia la richiesta di inserimento sarà anche il nodo che immagazzinerà i dati dell'oggetto. In contemporanea, verrà individuato, mediante la funzione $\mathcal{F}_h(K)$, il nodo responsabile per il set di keyword K_σ associate all'oggetto σ, chiamato *target*. Al nodo *target* verrà richiesto di mantenere la referenza dell'oggetto σ, in modo da poterla restituire al momento di una query riguardante quel set K.

Ricerca

Il caso d'uso base dell'applicativo è quello di fornire, dato in input un set di parole chiave, una lista di root MAM riguardanti quelle stesse keyword. La ricerca, anche detta *pin search*, inizia con l'individuazione del nodo che gestisce il keyword set K in input, mediante la funzione $\mathcal{F}_h(K)$. Una volta individuato, la sua tabella *references* fornirà gli ID di tutti gli oggetti σ associati a K. Successivamente sarà chiamata la funzione $Read(\sigma)$ propria di ogni algoritmo di partizionamento su DHT, per ritornare l'oggetto vero e proprio, ovvero l'indirizzo del canale MAM.

Per quanto riguarda invece la ricerca *superset*, dovranno far parte dei risultati tutti quegli oggetti che possono essere descritti dal keyword set K, senza esserlo strettamente. Verranno quindi ricercati tutti i nodi che gestiscono il superset, i quali costituiscono il sub-ipercubo indotto da u, dove $u = \mathcal{F}_h(K)$. In particolar modo, la ricerca nel sub-ipercubo può essere svolta esplorando l'albero *SBT(u)* con radice in u. Questo è costruito in maniera tale che, navigando per livelli, la ricerca sarà sempre più specifica, man mano si va in profondità nell'albero. Rifacendosi alla distanza di Hamming, la quale aumenta di 1 ogni livello, si ha che il nodo v al j-esimo livello di *SBT(u)* avrà j keyword in più nel suo set rispetto al set K iniziale.

Capitolo 5

Implementazione

Una distributed hash table è una struttura dati decentralizzata che consente di mantenere informazioni distribuite tra i nodi, l'implementazione su una struttura a ipercubo consente l'uso di keyword per effettuare query e garantisce una buona distribuzione del carico. Come si potrà verificare nel capitolo 6, permette di implementare un sistema che non soffre di scalabilità. Anche in un'architettura composta da molti nodi, il routing per raggiungere uno di essi sarà indipendente dal numero degli oggetti contenuti nella rete e il numero di passi massimi effettuati per raggiungere un nodo sarà r, ovvero la dimensione dell'ipercubo. Si è deciso nel corso del progetto di memorizzare all'interno della rete DHT, non solo le root iniziali di canali MAM associati, bensì le root di ogni messaggio MAM di ogni canale. Questo per consentire a un canale di gestire, se lo richiedesse, informazioni riguardanti keyword diverse.

Ogni messaggio, oltre ovviamente a essere caricato nel Tangle, è indicizzato nella rete DHT. Quest'ultima non mantiene i dati del messaggio, bensì solamente l'indirizzo per individuarlo all'interno del Tangle. In questo modo non si perde la proprietà di integrità dei dati che caratterizza i sistemi DLT.

5.1 Simulazione DHT

Rappresentando uno scenario sperimentale, la rete DHT è stata simulata. A tal proposito è stata implementata, in linguaggio Java, una struttura a oggetti. Questa è stata sviluppata al fine di far dialogare i vari oggetti nodi, come fossero realmente all'interno di una rete DHT.

Questa scelta implementativa deriva dal fatto che il focus del progetto è incentrato sull'ipercubo e la sua struttura, la quale è si collegata a una Distributed Hash Table, ma le logiche di inserimento e ricerca di un oggetto seguono la topologia a ipercubo.

5.1.1 Nodi

Alla base della struttura vi sono ovviamente i nodi, rappresentazione dei nodi dell'ipercubo. Ogni nodo ha dei vicini, definiti anche neighbors e visibili in figura 4.1. Due nodi *u,v* si definiscono neighbors se vi è un arco *edge(u,v)* che li collega, ovvero se differiscono tra loro di un solo bit. Un oggetto nodo conosce l'indirizzo dei suoi neighbors e mantiene in memoria le referenze in una tabella indice.

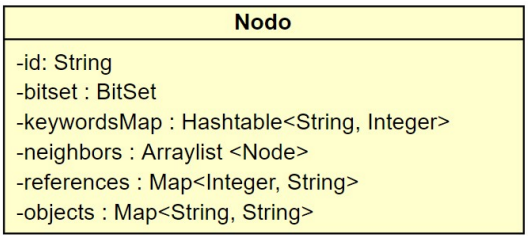

Figura 5.1: Diagramma della classe Nodo

Un nodo è rappresentato da un ID (in binario), un BitSet (equivalente all'ID), il quale permette di lavorare al meglio con gli id, avendo già implementate all'interno della sua classe operazioni logiche tra cui OR e XOR, importanti per

la costruzione dell'albero SBT. Inoltre un nodo possiede ovviamente la lista dei suoi neighbors, di cui conosce gli indirizzi. KeywordsMap rappresenta una mappatura *keyword : numero intero*, non si è fatto uso di funzione hash in quanto utilizzando un set di chiavi limitato non era fondamentale. References raccoglie tutti gli id degli oggetti di cui un nodo si occupa. Objects invece, contiene gli oggetti che il nodo mantiene in memoria, ma di cui non per forza detiene la responsabilità (perché potrebbero avere keyword diverse).

La figura 5.2 riporta l'esempio di uno schema di indicizzazione in un ipercubo a 4 dimensioni. In ogni nodo sono presenti due tabelle, la tabella Objects contiene gli oggetti veri e propri che il nodo mantiene, mentre la tabella References consente di indicizzare gli oggetti di cui il nodo è responsabile (per via del loro set di keyword).

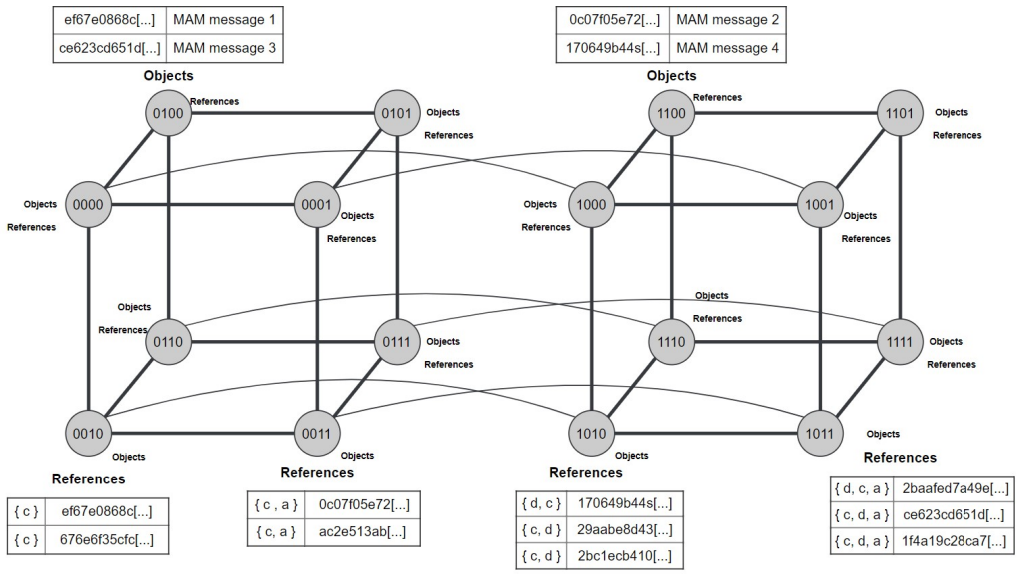

Figura 5.2: Schema di referenze all'interno dell'ipercubo

Nel resto del capitolo si analizzano nel dettaglio le principali funzioni dell'overlay, quali inserimento di un oggetto, ricerca per keyword, sia puntuale che superset. Prima di queste si analizza l'algoritmo di routing che permette di ottenere un percorso ottimale tra due nodi.

Routing tra nodi

Si immagini un nodo di partenza A con ID=0101 e un nodo *target* B con ID=0110. Si vuole raggiungere il nodo B attraverso il cammino minimo all'interno della rete. A meno che i nodi non abbiamo un solo bit di differenza, non sarà possibile ottenere un collegamento diretto, sarà necessario quindi passare attraverso i neighbors di A e, se necessario, avanzare in questo modo ricorsivamente. Il messaggio che viaggia di nodo in nodo deve avvicinarsi sempre di più al *target*.

Di seguito è rappresentato un semplice esempio in un sub-ipercubo a 4 dimensioni, il cammino è perciò semplice e immediato. In un ipercubo di dimensioni

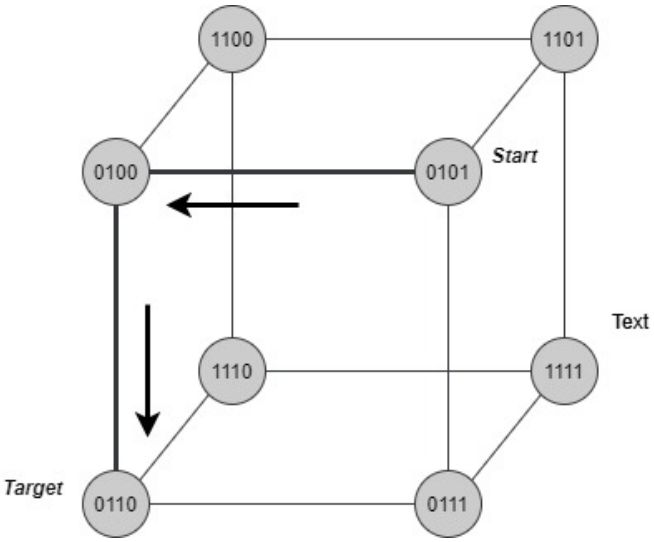

Figura 5.3: Routing ottimale fra due nodi

maggiori di 4 invece, il problema è di importanza considerevole, trovare il cammino minimo mediante un algoritmo efficiente permette di risparmiare passaggi inutili.

Il nodo A è il nodo di partenza, esso prende in input il messaggio da inviare e il nodo B da raggiungere. Se il nodo da raggiungere è A stesso, termina. Altrimenti ripete ricorsivamente l'algoritmo sul nodo che, tra i suoi neighbors, è il più vicino al *target*. Il nodo più vicino è rappresentato dal nodo che ha più bit

in comune con il *target*, ovvero quello che in confronto a tutti gli altri neighbors ha distanza di Hamming minore. E' possibile che vi siano più percorsi ottimali, questi sono equivalenti, nel senso che vengono portati a termine con gli stessi passi. L'algoritmo sceglie sempre il primo che incontra.

5.1.2 Inserimento di un oggetto

L'inserimento di un oggetto nell'overlay prevede alcuni passaggi. Innanzitutto prende in input l'oggetto σ, costituito dal set di chiavi K_σ associate a esso e il valore *data* dello stesso. Verranno salvate nella rete le coppie $< idObject, data >$ e $< K_\sigma, idObject >$. Queste due coppie verranno immagazzinate in nodi diversi del sistema, la prima dipenderà dall'algoritmo di partizionamento della DHT, mentre la seconda dipenderà da K_σ, ovvero verrà memorizzata dal nodo responsabile del determinato set di keyword.

Procedendo con ordine, un nodo u prende in carico la richiesta di *Insert* gestendo in input *data* (l'indirizzo MAM) e il *keywordSet* K associato. Essendo la variabile *data* rappresentante un indirizzo, quindi una stringa, si è deciso per semplicità di prendere questo valore come *filename*. La stringa è sottoposta a funzione *hash* SHA-1 al fine di determinare l'id univoco dell'oggetto σ. Una volta ottenuto l'id, la coppia $< idObject, data >$ è aggiunta all'hash table *objects* mantenuta dal nodo u che ha fatto la richiesta, mediante un messaggio $put(id, data)$. Questa operazione è l'operazione classica eseguita su una DHT, l'inserimento di una coppia chiave-valore. In una reale DHT però, il messaggio $put(id, data)$ sarebbe dovuto passare di nodo in nodo fino a raggiungere il nodo corrispondente all'id secondo l'algoritmo di partizionamento della rete. In questo caso l'algoritmo di partizionamento è rappresentato da una index table contenente entry del tipo $< idObject, idNodo >$, così da risalire, in base all'id dell'oggetto, al nodo che possiede la copia dell'oggetto.

Successivamente, è necessario creare la referenza che permetta il collegamento $K\sigma \rightarrow\ idObject$. Viene perciò inviato un messaggio $Insert((F_h(K_\sigma)), K_\sigma, \sigma Id)$

il quale passando da nodo a nodo mediante l'algoritmo di routing, arriva al responsabile $F_h(K_\sigma)$, il quale aggiungerà la referenza nella sua tabella *references*.

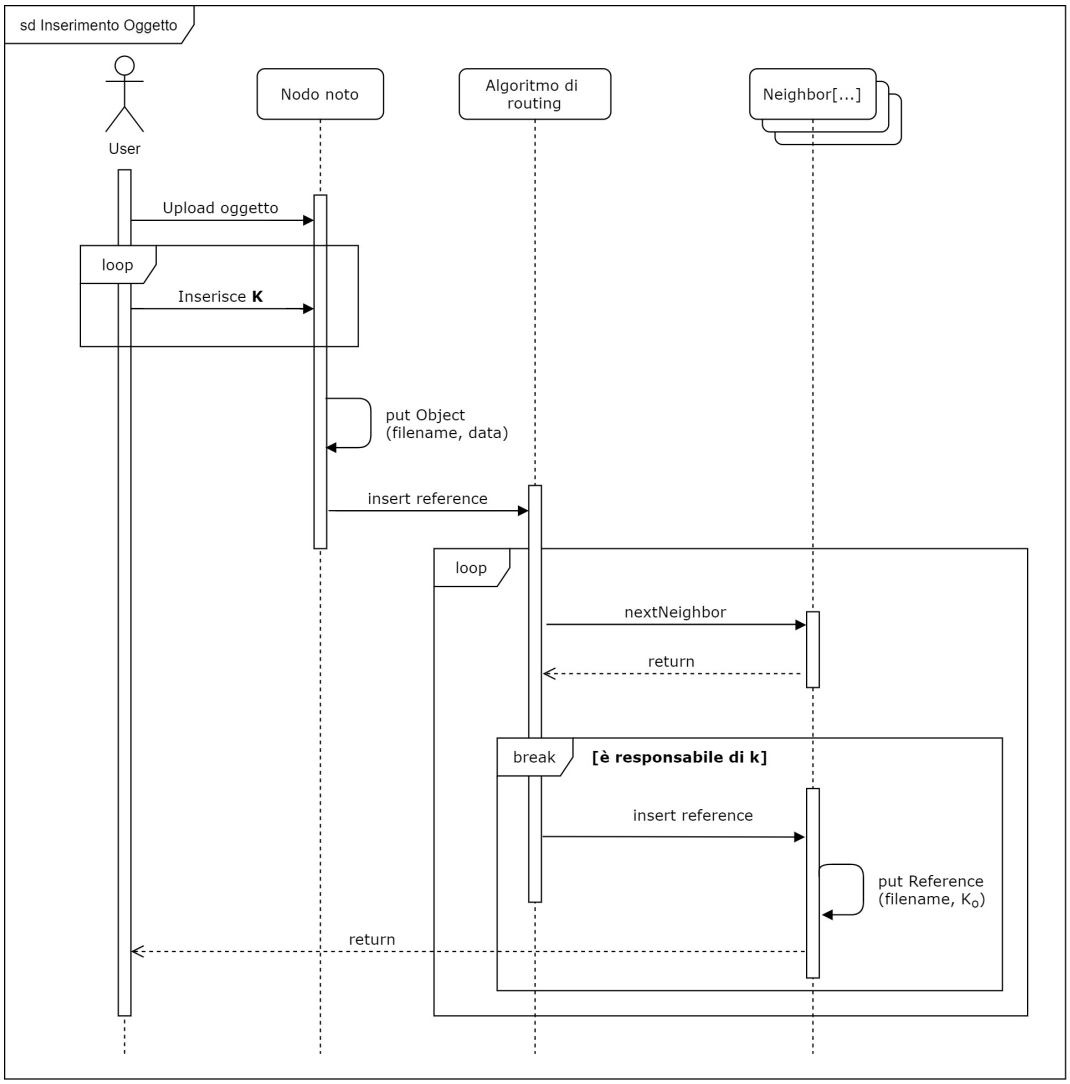

Figura 5.4: Diagramma di sequenza: scenario Insert

5.1.3 Pin Search

La ricerca parte dall'inserimento in input, da parte dell'utente, di un set di chiavi $K = (k_1, ..., k_n)$, al fine di recuperare oggetti descrivibili dal set stesso.

Il set K è memorizzato in un HashSet$< String >$, il quale è convertito nel BitSet corrispondente mediante la funzione definita nel paragrafo 4.2.2 e nominata $F_h(K)$. Dal nodo u, a cui un utente è collegato per eseguire la richiesta, sarà inviato un messaggio $getReferences(F_h(K), K)$, il quale attraverso l'algoritmo di routing arriverà al nodo responsabile del set di keyword. Una volta ricevuto il messaggio, il nodo in questione restituirà tutte le references degli oggetti da lui gestiti, sotto forma di ArrayList$< idObject >$. Ora la ricerca procede come una normale ricerca all'interno di una rete DHT. Sarà richiamata la funzione $Read(idObject)$. Questa ricava il nodo che se ne occupa secondo la mappatura e restituisce l'oggetto, ovvero l'indirizzo del messaggio MAM.

5.1.4 Superset Search

La ricerca *Superset* riprende *Pin Search*, ma la estende anche ad altri nodi, rappresentanti di altri set di keyword. In input della richiesta, oltre al set di keyword K, viene definita anche una variabile c, rappresentante un valore minimo di risultati in risposta. Il sistema quindi risponderà con almeno c valori che possono essere descritti dal set K.

Per fare ciò, la ricerca parte sempre dal nodo $F_h(K)$, ma da qui si estende a tutto il sub-ipercubo indotto da $F_h(K)$. Per ricercare nel sub-ipercubo, è istanziato l'albero SBT(u), dove u è proprio il nodo $F_h(K)$.

Creazione albero SBT

Viene istanziato un oggetto di tipo Tree, classe creata appositamente, il quale rappresenterà la root dell'albero. Quest'oggetto ha un id (equivalente all'id del

nodo $F_h(K)$), un padre (settato a *null* in quanto radice) e una lista di figli. La creazione dell'SBT segue i passi di un algoritmo ricorsivo.

```
function  generateSBT (int r) {
//this → nodo che richiama la funzione
//r responsabilita'

new root ← this //radice dell'albero
//esploro solo i neighbors che includono un nodo
For each node in neighborsIncluded {
  if (node is leaf) {
  root.addChild(node); //aggiungo foglia
  } else if (xorChild(this.id, node.id) <= r) {
            //richiamo la funzione sul nuovo figlio
            root.addChild(generateSBT(xorChild(idFthr,idChld)))
             }
return root
}}
```

La funzione xorChild() ritorna la posizione del primo bit settato a 1 dopo un'operazione di xor tra due BitSet.

```
function xorChild(BitSet father, BitSet child)
BitSet result ← xor(father,child) //xor di due stringhe di bit
return result.nextSetBit(0) //posizione del primo bit settato
```

Vengono ispezionati i neighbors del nodo in questione, se il valore *id* è quello di una foglia, viene aggiunto un figlio all'albero. Se il valore *id* non è foglia, ma è comunque contenuto nell'SBT, viene istanziato un nuovo sotto albero SBT e sullo stesso viene richiamata la funzione generateSBT(). L'algoritmo termina quando sono stati aggiunti tutti i nodi del sub-ipercubo. L'albero SBT è poi esplorato seguendo l'algoritmo Breadth-First search (BFS), ovvero è eseguita una ricerca in ampiezza tra i vari nodi. In questo modo, avanzando con la ricerca, gli oggetti ritornati dalla funzione saranno via via più specifici, perché descritti da più keyword.

5.2 Comunicazione con il Tangle

Come precedentemente visto, nel Tangle viene immagazzinata qualsiasi transazione avvenga all'interno della piattaforma IOTA. La IOTA Foundation mette a disposizione le proprie librerie, disponibili in diversi linguaggi di programmazione, per la connessione e comunicazione al registro. Per una questione di completezza si è deciso di utilizzare le librerie Javascript, in quanto più mature rispetto ad altri linguaggi.

5.2.1 Reti Mainnet e Devnet

IOTA prevede al suo interno due network di nodi differenti, i quali mantengono due Tangle distinti. Il primo è il network "ufficiale" denominato Mainnet, mentre il secondo è una rete di test denominata Devnet. Entrambe le reti si comportano in maniera uguale, ma sulla prima i token sono reali e lo svolgimento della PoW è più dispendiosa, mentre la seconda è una rete adibita al testing, dove i token sono solamente simulati e non hanno valenza reale. Il progetto di tesi si svolge su quest'ultima.

5.2.2 Creazione canali MAM

Al fine di testare l'applicativo sono stati creati diversi MAM channel. Ogni messaggio MAM è solitamente composto da dati rilevati da un sensore, a questo, è stato aggiunto un campo tag, scelto in maniera casuale da un set, in maniera tale da avere una prova se il sistema fosse in grado o no di indicizzare i messaggi in base a quest'ultimo. Il set di keyword con cui è stato popolato il tag è lo stesso set su cui si basa la DHT creata in precedenza.

Infatti, durante la creazione di ogni messaggio, la rete DHT è stata utilizzata per indicizzare le root corrispondenti in base al tag che ognuno di questi portava.

Al fine di creare un canale MAM su Devnet, la prima cosa da fare, oltre ovviamente a richiedere i pacchetti delle librerie IOTA, è quella di connettersi a un nodo sulla rete.

```
const iota = Iota.composeAPI({
provider: 'https://nodes.devnet.iota.org:443'
});
```

Viene poi inizializzato il canale MAM. Se non specificato, il seed è scelto casualmente.

```
let mamState = Mam.init(iota.provider);
```

Dopo le operazioni preliminari, viene poi definito un json object contenente i dati del messaggio MAM da caricare, compreso il tag. L'oggetto viene poi convertito in trytes, per consentire al sistema IOTA di gestirlo.

```
  const trytes = Converter.asciiToTrytes(JSON.stringify(json));
```

Una volta che il messaggio è stato convertito in trytes, una funzione apposita della libreria MAM si occuperà di gestire la root, l'indirizzo e lo stesso oggetto in trytes.

```
  const message = Mam.create(mamState, trytes);
  mamState = message.state;
```

La funzione tornerà 4 valori principali. Il payload, ovvero il contenuto vero e proprio del messaggio, contenente anche la firma e cifrato in tutte le sue parti. La root del messaggio MAM, l'address del messaggio e lo state. La variabile state contiene informazioni per un eventuale successivo messaggio, infatti mamState, che era stato inizializzato attraverso la funzione di init, acquista un nuovo valore.

Dopo aver ottenuto questi valori, il payload è caricato sul Tangle sotto forma di transazione.

```
   iota
   .prepareTransfers(mamState.seed, transfers)
   .then(trytes => {
   return iota.sendTrytes(trytes, depth, minimumWeightMagnitude);
      })
      .then(bundle => {
        console.log("root del messaggio MAM: "
                    + mamMessage.info.root);
      })
      .catch(err => {
        console.error(err)
      });
```

5.2.3 Ricezione e lettura messaggi MAM

Attraverso le librerie fornite da IOTA è possibile ricevere per intero la catena di messaggi MAM, dalla prima root posseduta fino all'ultimo messaggio del canale. Durante il progetto però, il singolo messaggio MAM è stato visto come un singolo oggetto, perciò è stato implementato un sistema in grado di leggere dal Tangle il singolo messaggio appartenente a una root.

Dopo aver richiamato le librerie, essersi collegati a un nodo del network e aver inizializzato il mamState, sarà necessario richiamare la funzione seguente. A partire da una root, il messaggio MAM verrà decriptato come trattato nel capitolo 3 e ne verrà verificata la validità. Successivamente il messaggio sarà convertito in ascii per renderlo leggibile in modo chiaro.

```
    async function fetch() {
      const data=await Mam.fetchSingle(rootMam, 'public', null);
      console.log(trytesToAscii(data.payload));
}
```

Capitolo 6

Discussione

Nel capitolo in questione viene ripreso il modello di ricerca implementato e vengono analizzati i vari passi necessari al completamento della richiesta. Successivamente, si illustrano gli sviluppi futuri dell'applicativo.

6.1 Analisi

Riassumendo, l'iter di ricerca nell'overlay (*Superset search*) è il seguente:
Sia dato in input il keyword set K, il sistema dovrà ritornare tutte le root dei messaggi MAM che possono essere descritte da K. Inizialmente viene individuato il nodo che si occupa del set K mediante la funzione $F_h(K)$. Successivamente viene creato un albero denominato $SBT(u)$ con radice in u, dove u è proprio il nodo responsabile per K. A partire dalla radice dell'albero e scendendo poi livello per livello, verranno ritornati gli id degli oggetti contenuti nella DHT conformi al set K. Una volta ottenuti gli id (la chiave in una rete DHT), si eseguirà una normale ricerca nella DHT a partire da questi. La ricerca pin search è analoga con la sola differenza che non viene creato l'albero $SBT(u)$, quindi la ricerca non è estesa ad altri nodi al di fuori di $F_h(K)$.

6.1.1 Ricerca pin search

Durante la ricerca pin search è importante capire qual è il costo per raggiungere un determinato nodo v a partire da un nodo di partenza u. Come discusso nel capitolo 5.1.1, il routing tra un nodo e l'altro è svolto mediante l'utilizzo di un algoritmo *greedy*. Questo tipo di algoritmo sceglie, passo per passo, una soluzione che appare globalmente ottimale al fine di raggiungere un determinato nodo *target*.

Siano dati due nodi (u,v) aventi id composti da r-bit, all'interno di un ipercubo a r-dimensioni. Per semplicità, si assumono u e v come gli identificativi degli id dei due nodi. Si definiscono inoltre i due insiemi $One(u)$ e $One(v)$, contenenti le posizioni dei bit settati a 1 in u e in v. Il numero di passi minimi necessari per raggiungere il nodo u dal nodo v, o viceversa, è dato da $|One(u \oplus v)|$. L'operatore logico $\oplus$ (XOR) permette di evidenziare le differenze tra due set di bit, mentre viene calcolata la cardinalità per quantificare questa differenza. Ad esempio, siano a=010100 e b=101101. Sia $c = a \oplus b = 111001$. I bit differenti tra a e b sono $|One(c)| = 4$.

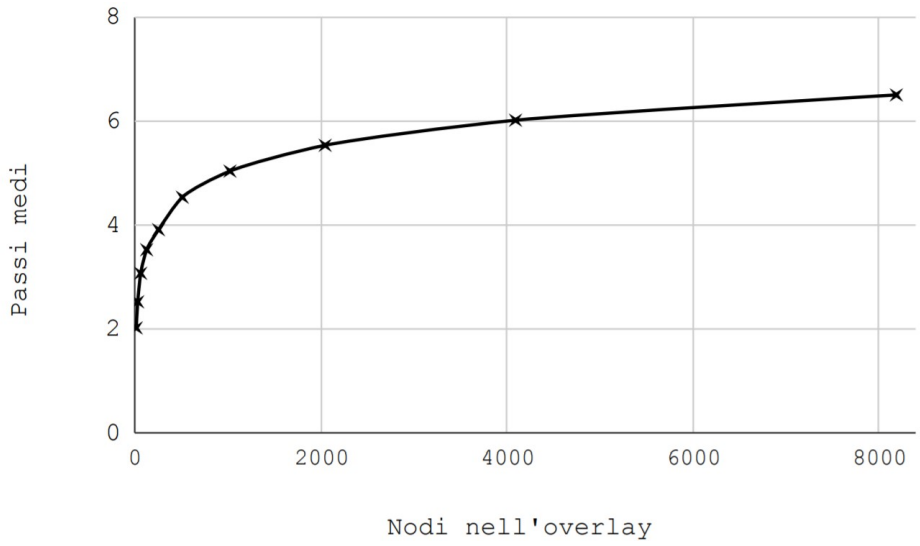

Figura 6.1: Passi medi routing tra due nodi

Durante il routing si procede navigando per i neighbors, quindi un nodo u si avvicina al massimo di una posizione verso il nodo *target* v per passo. Per questo motivo, nel caso peggiore, il numero di passi massimi che un messaggio dovrebbe percorrere è pari a r.

E' stato tracciato il grafico 6.1, per il quale sono stati creati diversi modelli di ipercubo, ognuno con una dimensione r di partenza diversa. Il numero dei nodi nel sistema è strettamente collegato a r, il parametro varia da 16 (r=4) a 8192 nodi (r=13). Si sono svolte circa 1500 route di test, tra due nodi scelti in maniera casuale nel network, per ogni dimensione r.

6.1.2 Ricerca superset

Anche la ricerca superset inizia con il routing di cui discusso al paragrafo precedente. Successivamente viene costruito l'SBT(u) e vengono contattati, livello per livello, i nodi all'interno dell'albero. A partire da un nodo u, preso come responsabile di un set di keyword K e radice dell'SBT(u), i nodi che comporranno l'albero saranno $2^{r-|One(u)|}$, nodo u radice compreso. Questo sarà il numero massimo di nodi da contattare durante la ricerca, ma tutto dipenderà dal numero di oggetti trovati rispetto al parametro c in input, discusso nel paragrafo 5.1.4. Inoltre, i nodi che vanno a comporre un SBT conoscono i propri vicini, quindi il routing sarà diretto.

6.1.3 Ricerca in DHT

Successivamente alle due ricerche, sia puntuale che superset, si avrà in ritorno una lista di id. Questi id rappresentano gli identificativi per gli oggetti contenuti nella rete DHT, ovvero le root dei messaggi MAM. Il costo della ricerca su rete DHT dipende dall'implementazione della stessa. In una rete basata su protocollo Chord [6] o Pastry [7], il costo per raggiungere un oggetto attraverso il suo id è stimato in O(log N), dove N è il numero di nodi nel sistema.

6.2 Sviluppi futuri

6.2.1 Smart contracts

L'architettura alla base dell'overlay potrebbe essere estesa mediante l'utilizzo di smart contracts, per la gestione delle *sideKey* nei messaggi MAM di tipo *restricted*, i quali necessitano di una particolare chiave per decifrare il loro contenuto. Avere molti messaggi provenienti da channel diversi, necessiterebbe la conoscenza della *sideKey* per ognuno di essi. Il processo potrebbe quindi essere automatizzato mediante l'uso di smart contracts. Attualmente IOTA non include al suo interno un sistema di computazione distribuita come l'Ethereum Virtual Machine, ma la situazione potrebbe cambiare nel corso dei prossimi mesi con il rilascio di Qubic. Qubic è la soluzione proposta da IOTA per la computazione quorum-based e l'implementazione di smart contracts. Mediante l'utilizzo di questo protocollo potranno essere sviluppate vere e proprie applicazioni distribuite, come accade per le dApp sviluppate in Ethereum al momento. Riguardo a quanto implementato, potrebbe venir implementato un contratto che, a seguito di determinate condizioni, rilasci la *sideKey* per uno specifico messaggio.

6.2.2 Una struttura a ipercubo per IOTA

Nel paragrafo 2.6 si è discusso di una caratteristica che differenzia IOTA da altri registri DLT. Questo registro infatti permette ai propri nodi di svolgere periodicamente Snapshot, ovvero eliminare tutte le vecchie transazioni e ripartire da un *genesis* contenente solo i bilanci positivi e gli account relativi. I nodi quindi, dopo un'istantanea, ripartono con un database vuoto, con conseguenti vantaggi di spazio di archiviazione e velocità. L'intera storia del Tangle è comunque mantenuta dai Permanode, particolari nodi non soggetti a Snapshot.

Si potrebbe immaginare di strutturare quest'ultimi come veri e propri nodi di un ipercubo, sviluppato però con una struttura ternaria [14]. Un ipercubo

ternario non modificherebbe in alcun modo la logica discussa nel corso di questa tesi. Si rifletterebbe piuttosto sulla rappresentazione dei nodi, in quanto questi verrebbero rappresentati in stringhe di trits.

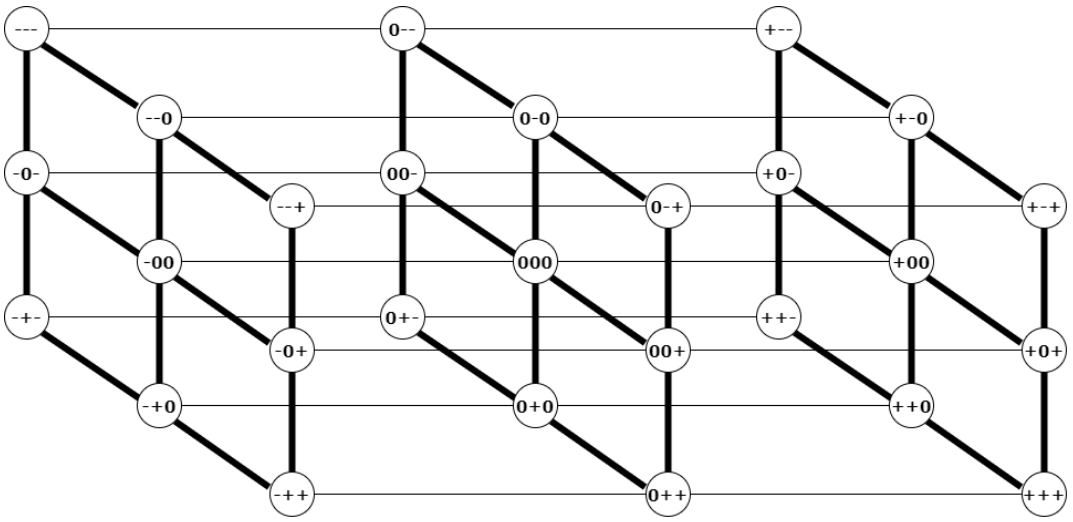

Figura 6.2: Rappresentazione di un ipercubo ternario

In figura si riportano la struttura e i nodi di un ipercubo ternario. A differenza di un convenzionale ipercubo, data la rappresentazione in trits, un ipercubo ternario $H_r(V, E)$ di dimensione r è composto da 3^r nodi. Un arco $edge(u, v)$ collega il nodo u con il nodo v se, e solo se, u differisce da v di un solo trit e la differenza è di un solo valore. Ad esempio, due nodi $u = 0-1-1$ e $v = 0+1-1$ non sono neighbor anche se differiscono di un solo trit, perché la differenza nel trit che li separa è maggiore di 1.

In questo scenario l'ipercubo non sarebbe più uno strato sopra la rete, bensì la struttura con cui i Permanode sono organizzati, portando due principali benefici:

- I Permanode non dovrebbero più mantenere tutto il Tangle e i dati in esso non sarebbero replicati. Ogni nodo prenderebbe in carico solamente le transazioni che lo riguardano rispetto a determinati parametri, quali ad esempio le keyword.

- In secondo luogo potrebbe essere implementato un sistema che consenta di effettuare query già all'interno del registro, senza la creazione a posteriori di un overlay che indicizzi i dati. In questo specifico scenario, si pensa a un sistema che, a partire da un set di keyword K in input e un Permanode a cui essere collegati, ricavi mediante una funzione simile a $F_h(K)$, il Permanode che si occupa di quel relativo set e restituisca i dati richiesti.

Capitolo 7

Conclusioni

E' stato discusso e affrontato il problema di come poter svolgere query keyword-based all'interno di uno scenario distribuito. L'abitudine su questo tipo di sistemi distribuiti è quella di ricercare esattamente un dato, mediante codice hash, o nel caso specifico di messaggi MAM, mediante la root. Questa ricerca è limitante, sia per il fatto di non poter aver un set di dati in risposta, sia perché spesso non si conosce esattamente il nome del dato da ricercare.

La prima soluzione, di facile comprensione ma di elevata complessità computazionale, sarebbe quella di "filtrare" i risultati a posteriori della ricerca, ovvero richiedere prima tutti i dati e solo successivamente ispezionarli.

Una soluzione alternativa a quest'ultima è stata descritta lungo il corso di queste pagine e implementata. L'idea alla base è quella di indicizzare i contenuti del registro IOTA, mappando ogni oggetto in un vettore a r-bit, in accordo con il suo set di keyword e immagazzinando la sua referenza all'interno di una Distributed Hash Table. La topologia con cui è stata progettata la DHT segue un modello a ipercubo, grazie al quale è stato possibile ottenere percorsi di routing efficienti, una distribuzione degli oggetti all'interno della rete bilanciata e ricerche indipendenti dall'id di un oggetto. Tutta l'architettura ruota intorno a un keyword set e alla mappatura di questo in un vettore composto esclusivamente

da bit. La logica di mappatura è implementata sia durante l'inserimento di un oggetto per collocarlo nel nodo responsabile del set, sia durante la ricerca per raggiungere il nodo responsabile di quella ricerca. L'uso dell'albero SBT ha permesso poi di effettuare query dinamiche, ovvero espandibili durante il percorso di ricerca, permettendo di raggiungere oggetti non strettamente descritti dal keyword set. Molto importante inoltre, l'idea di un ipercubo ternario alla base della logica di organizzazione dei Permanode. Questa struttura permetterebbe non solo un risparmio di spazio d'archiviazione, ma una ricerca keyword-based integrata al registro IOTA, senza ulteriori layer o meccanismi di indicizzazione.

Bibliografia

[1] Satoshi Nakamoto. «Bitcoin: A Peer-to-Peer Electronic Cash System». In: (2008).

[2] Osama Hosam. «Hiding Bitcoins in Steganographic Fractals». In: (2018), p. 512.

[3] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized application platform*. 2014. URL: `https://github.com/ethereum/wiki/wiki/White-Paper`.

[4] Serguei Popov. «The tangle». In: (2015).

[5] IOTA Foundation. *IOTA Documentation*. URL: `https://www.iota.org/get-started/what-is-iota`.

[6] I. Stoica et al. «Chord: a scalable peer-to-peer lookup protocol for Internet applications». In: (2003), pp. 17–32.

[7] Antony Rowstron e Peter Druschel. «Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems». In: (2001), pp. 329–350.

[8] B. Y. Zhao et al. «Tapestry: a resilient global-scale overlay for service deployment». In: (2004), pp. 41–53.

[9] Yuh-Jzer Joung, Li-Wey Yang e Chien-Tse Fang. «Keyword Search in DHT-based Peer-to-Peer Networks». In: (2007), pp. 46–61.

[10] IOTA foundation. *Coordicide*. URL: https://files.iota.org/papers/Coordicide_WP.pdf.

[11] IOTA Foundation. *Structure of a transaction*. URL: https://docs.iota.org/docs/getting-started/0.1/transactions/transactions.

[12] ABmushi. *IOTA: MAM Eloquently Explained*. URL: https://medium.com/coinmonks/iota-mam-eloquently-explained-d7505863b413.

[13] Frank Harary, John P. Hayes e Horng-Jyh Wu. «A Survey of the Theroy of Hypercube Graph». In: (1988).

[14] I.-J Wang et al. «Supporting content-based music retrieval in structured peer-to-peer overlays». In: (2016), pp. 401–407.