Small-scale DHT System

Gianluca Ronzello
Università degli Studi di Roma Tor Vergata
gianlucaronzello@gmail.com

***Abstract*—In questo report viene descritta la realizzazione di un'infrastruttura distribuita che implementa le funzionalità di una hash table.**
L'infrastruttura è ispirata al modello CAN (Content-Addressable Network).

I. Introduzione

Una hash table è una struttura dati che consente di mappare efficientemente chiavi a valori. I sistemi distribuiti possono trarre grandi benefici da tali funzionalità, specialmente in scenari che richiedono accesso rapido e bilanciamento del carico. L'infrastruttura proposta è progettata per essere scalabile, tollerante ai guasti e completamente auto-organizzante.

II. Design

La progettazione si basa su uno spazio di coordinate cartesiane bidimensionale. Questo spazio è completamente logico e non ha alcuna relazione con il sistema fisico sottostante. In ogni istante, l'intero spazio è dinamicamente partizionato tra tutti i nodi del sistema, in modo che ciascun nodo possieda una propria zona individuale e distinta.

Lo spazio virtuale viene utilizzato per memorizzare coppie chiave-valore: la chiave è mappata in modo deterministico in un punto dello spazio tramite due funzioni di hash, una per la coordinata X e una per la Y. La coppia viene quindi memorizzata nel nodo che possiede la zona contenente quel punto.

Per recuperare l'elemento corrispondente a una chiave, ogni nodo applica le stesse funzioni di hash alla chiave per determinare il punto P corrispondente, e quindi recuperare il valore associato. Se il punto non appartiene al nodo richiedente, la richiesta viene instradata nella rete fino a raggiungere il nodo che lo contiene.

I nodi si auto-organizzano in un overlay network che rappresenta lo spazio virtuale. Ogni nodo apprende e mantiene informazioni solo sui propri vicini, inclusi il loro indirizzo e le coordinate. Queste informazioni costituiscono la tabella di routing, necessaria per effettuare l'instradamento tra due punti qualsiasi nello spazio.

A. Routing

Il routing avviene seguendo un percorso lineare nello spazio, dalle coordinate di partenza fino a quelle di destinazione. Ogni nodo mantiene una tabella di routing composta dall'indirizzo e dalle coordinate dei propri vicini, dove un vicino è definito come un nodo con cui esiste un'area di sovrapposizione lungo l'asse X o Y.

Per effettuare il routing, ogni messaggio contiene le coordinate di destinazione. Il messaggio viene quindi instradato in modo greedy verso il nodo più vicino alla destinazione, secondo la distanza cartesiana.

Poiché possono esistere diversi percorsi verso il nodo di destinazione, in caso di guasto di un nodo intermedio il sistema tenterà automaticamente un percorso alternativo. Questo viene realizzato tramite un flooding controllato all'interno della rete, in cui il nodo cerca tra i propri vicini un altro nodo in grado di raggiungere la destinazione.

B. Costruzione

Lo spazio è diviso tra i nodi del sistema. Per permettere alla rete di crescere in modo incrementale, un nuovo nodo che si unisce al sistema deve essere allocato in una porzione di spazio. Questo avviene dividendo la zona di un nodo esistente a metà: il nodo originario mantiene una metà, mentre la restante viene assegnata al nuovo nodo.

Come primo passo, il nuovo nodo deve entrare in contatto con un nodo già presente nella rete. Per farlo, si rivolge al nodo di bootstrap, che mantiene una lista di nodi considerati attivi e ne restituisce uno scelto casualmente.

Il nuovo nodo seleziona quindi un punto casuale nello spazio. Il nodo restituito dal bootstrap utilizza il meccanismo di routing per individuare il nodo responsabile della zona contenente quel punto. Per garantire una distribuzione più uniforme, il nodo che riceve la richiesta verifica se ha un vicino con una zona più grande della propria. Se lo trova, sarà quel vicino a dividere la propria zona; altrimenti, divide la propria. La divisione avviene prima lungo l'asse X, e successivamente lungo l'asse Y, facilitando un'eventuale futura riunificazione delle zone nel caso in cui un nodo lasci la rete. Le risorse contenute nella nuova zona vengono trasferite al nodo appena aggiunto. Esempio figura 1
Una volta assegnata la zona, il nuovo nodo apprende l'indirizzo e le coordinate dei vicini dall'ex occupante. Entrambi i nodi aggiornano la propria lista dei vicini, rimuovendo quelli non più adiacenti a seguito della modifica delle zone. Successivamente, informano i rispettivi vicini del cambiamento avvenuto.

Poiché due nodi vicini potrebbero, simultaneamente, aggiungere nuovi nodi, può succedere che due nuovi nodi non vengano a conoscenza della reciproca esistenza, pur essendo vicini. Per gestire questa eventualità, viene eseguita una fase

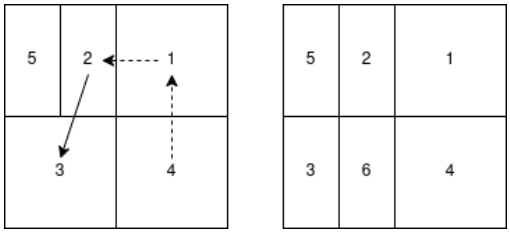

Fig. 1. Il nodo che vuole aggiungersi contatta il nodo 4 e ha come punto di inserimento (0.4,0.8).
La linea tratteggiata rappresenta il routing fino al punto, la linea continua indica che il nodo che se ne occupa trova un vicino con zona più grande

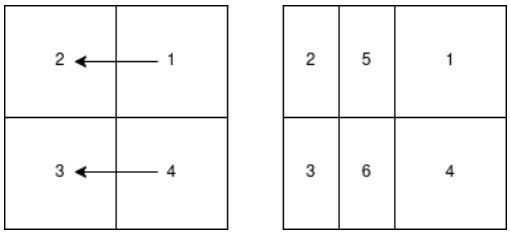

Fig. 2. Esempio aggiunta simultanea nodi

ulteriore di *handshaking* tra il nuovo nodo e i vicini dei suoi vicini. Figura 2

Periodicamente, ogni nodo invia ai propri vicini un messaggio contenente le proprie coordinate e la lista aggiornata dei vicini. Questo meccanismo di aggiornamento assicura che tutti i nodi mantengano una visione coerente della topologia locale. L'aggiunta di un nuovo nodo ha impatto su un numero molto limitato di altri nodi, localizzati nelle sue vicinanze spaziali.

C. Partenza di un nodo, recupero e manutenzione della rete

Quando un nodo lascia la rete, è necessario garantire che la zona di cui era responsabile venga assegnata ad altri nodi. Se l'abbandono è volontario, il nodo verifica se possiede un "fratello", ovvero un nodo con una zona dello stesso volume situato nella direzione dell'ultima divisione. Per accertarsi che si tratti effettivamente di un fratello (e non semplicemente di un nodo con zona equivalente lungo lo stesso asse), viene controllato che l'unione delle due zone produca un intervallo valido, ottenibile da suddivisioni binarie dello spazio iniziale $[0, 1]$. Se tale nodo esiste, la zona e tutte le risorse vengono a lui affidate. Figura 3

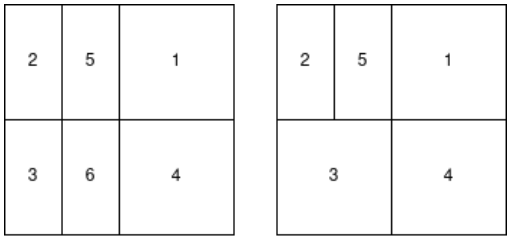

Fig. 3. Esempio 1 rimozione nodo

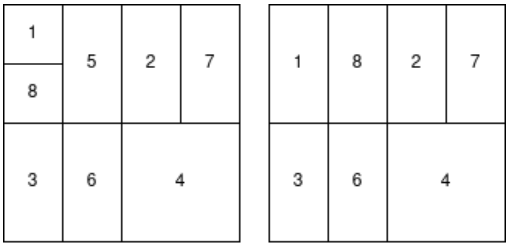

Fig. 4. Esempio 2 rimozione nodo 5

In caso contrario, il nodo trasferisce temporaneamente le risorse al vicino con la zona di dimensioni minori. Questo attiverà una procedura ricorsiva: ogni nodo verifica se ha un fratello oppure passa il controllo al vicino con zona più piccola. La procedura continua finché non si trova un nodo con un fratello. Quest'ultimo unirà le due zone originarie, mentre l'ultimo nodo coinvolto si occuperà della zona liberata dal nodo uscente. Figura 4

Per gestire i guasti imprevisti, ogni nodo mantiene una replica delle proprie risorse presso un insieme di nodi di backup. Il numero di nodi di backup è configurabile. Essi vengono selezionati casualmente tra i nodi attivi disponibili al momento dell'ingresso nella rete. Se non è possibile trovarne un numero sufficiente, il nodo tenterà di completare la lista in seguito, ad esempio quando riceverà un nuovo messaggio di *heartbeat*. Ogni nodo riceve periodicamente segnali di *heartbeat* dai propri nodi di backup, per verificarne lo stato. Se uno dei backup non risponde, il nodo si rivolge al nodo di bootstrap per ottenere un sostituto.

Nel messaggio di *heartbeat* che ogni nodo invia ai propri vicini, sono inclusi anche gli indirizzi dei nodi di backup. In caso di fallimento, queste informazioni permettono ai vicini di sapere a chi rivolgersi per il recupero delle risorse.

Se un nodo smette di ricevere segnali di *heartbeat* da un vicino, quest'ultimo viene considerato fallito. A quel punto, i vicini attivano una procedura di *takeover*: il primo nodo che rileva il guasto interroga gli altri vicini del nodo fallito per conoscere le dimensioni delle loro zone. Se scopre di avere una zona più grande rispetto a un altro vicino, rinuncia al *takeover*, lasciando il compito al nodo con zona più piccola. Questo nodo recupera quindi le risorse dal primo nodo di backup disponibile e attiva la procedura ricorsiva per cercare un nodo con un fratello. Il comportamento seguente replica quello del caso di un'abbandono volontario.

III. Performance

Il sistema proposto mostra buone proprietà di scalabilità e robustezza, grazie alla natura decentralizzata e auto-organizzante della rete. La complessità del routing tra due nodi qualsiasi nello spazio virtuale è in media $O(\sqrt{n})$, dove n è il numero totale di nodi nella rete. Questa complessità deriva dalla topologia logica bidimensionale e dal meccanismo di routing

greedy, che limita il numero di salti necessari per raggiungere il nodo responsabile di una certa zona.

Il carico viene distribuito uniformemente tra i nodi grazie al meccanismo di inserimento basato sulla divisione spaziale. La selezione del punto casuale nel piano e la successiva verifica della dimensione delle zone garantiscono una ripartizione bilanciata anche in presenza di inserimenti multipli. Ciò riduce la probabilità di congestione o sovraccarico di specifici nodi, mantenendo stabile la latenza media delle operazioni di inserimento, ricerca e aggiornamento delle chiavi.

Il sistema è inoltre progettato per essere fault-tolerant. La presenza di nodi di backup consente un recupero rapido in caso di guasto, limitando l'impatto del fallimento di un singolo nodo.
L'aggiornamento periodico delle informazioni di vicinato tramite messaggi di *heartbeat* assicura una rapida propagazione delle modifiche topologiche, mantenendo consistenti le tabelle di routing locali.

Inoltre, l'efficienza della procedura di *takeover* garantisce che la rete possa mantenere l'integrità dello spazio di allocazione anche in scenari di fallimenti multipli. Il numero di nodi coinvolti in queste operazioni è limitato all'intorno locale del nodo interessato, il che riduce il traffico complessivo sulla rete e mantiene bassa la latenza di recupero.

Nel complesso, il sistema offre un buon compromesso tra efficienza del routing, bilanciamento del carico, tolleranza ai guasti e semplicità dei meccanismi di gestione, rendendolo adatto a contesti altamente dinamici e distribuiti.

IV. Deployment

Il deployment del sistema è stato realizzato utilizzando le istanze EC2 (Elastic Compute Cloud) di Amazon Web Services (AWS), una soluzione altamente scalabile e flessibile per l'hosting di applicazioni in cloud. Le istanze EC2 consentono di eseguire e gestire server virtuali, noti come "istanze", in un ambiente cloud, con la possibilità di scegliere tra diverse configurazioni hardware, software e di rete. Nel contesto del progetto, è stata implementata un'infrastruttura basata su una Virtual Private Cloud (VPC) per garantire un isolamento sicuro tra le risorse. La configurazione prevede un'istanza bootstrap iniziale e altri nodi worker, i quali operano come partecipanti al sistema distribuito.
La creazione e gestione di queste istanze è automatizzata tramite l'uso di Terraform, uno strumento che permette di definire l'infrastruttura come codice. In questo modo, è possibile gestire e versionare l'infrastruttura in maniera efficiente. Durante il processo di deployment, ogni nodo viene configurato con script di inizializzazione personalizzati che installano le dipendenze necessarie e avviano il sistema distribuito. Inoltre, le istanze sono collegate tra loro tramite una rete privata, con Security Group configurati per consentire la comunicazione tra le macchine sulle porte necessarie (HTTP, gRPC) e per garantire l'accesso sicuro tramite SSH. Questo approccio consente di avere un'infrastruttura flessibile, scalabile e facilmente gestibile, con la possibilità di aggiungere nuovi nodi senza compromettere il funzionamento del sistema esistente.
Per quanto riguarda l'interfacciamento esterno, è stato realizzato un client HTTP che si connette a uno dei nodi tramite la sua interfaccia REST esposta sulla porta. Il client può inviare richieste HTTP per eseguire operazioni sul sistema distribuito, come l'aggiunta, la rimozione o il recupero di risorse. Questa modalità di interazione semplifica l'accesso al sistema da parte di utenti esterni o di altri servizi, consentendo un'integrazione semplice e modulare. Il client agisce come punto di ingresso per l'utilizzatore finale e comunica con i nodi che, a loro volta, cooperano tramite il protocollo gRPC per garantire la coerenza e la distribuzione dei dati all'interno della rete.

V. Conclusioni

In questo lavoro è stata presentata la progettazione e la realizzazione di un'infrastruttura distribuita che implementa una hash table scalabile e fault-tolerant, ispirata ai principi di un Content-Addressable Network (CAN). Il sistema proposto offre una suddivisione logica dello spazio delle chiavi, garantendo un'allocazione dinamica e un meccanismo di recupero delle risorse in caso di guasti, grazie alla gestione dei nodi di backup e delle procedure di *takeover*.

E' stato dimostrato che la rete mantiene buone proprietà di scalabilità, con una complessità di routing che cresce in modo sub-lineare rispetto al numero di nodi, e garantisce un bilanciamento efficace del carico tra i nodi. Il sistema è in grado di adattarsi rapidamente a modifiche topologiche, con un impatto minimo sulle prestazioni globali, grazie alla strategia di aggiornamento continuo delle informazioni di vicinato e all'uso di tecniche di *handshaking*.

Le performance in scenari di guasto sono risultate soddisfacenti, con un recupero rapido delle risorse e una gestione dei fallimenti che coinvolge solo un numero ristretto di nodi vicini, riducendo la latenza e il traffico complessivo nella rete. La resilienza del sistema è inoltre migliorata dalla capacità di ridistribuire automaticamente le zone di spazio non appena un nodo lascia la rete, senza compromettere la coerenza globale.

In sintesi, l'infrastruttura proposta si dimostra promettente per applicazioni distribuite che richiedono alta disponibilità, scalabilità e tolleranza ai guasti, fornendo una solida base per la realizzazione di sistemi più complessi in contesti di rete ampi e dinamici.

Riferimenti

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *ACM SIGCOMM*, 2001.

[2] https://people.eecs.berkeley.edu/~sylvia/papers/cans.pdf