LPN: Log Partition Network

Daniele Ferrarelli*, Marco Ferri†, Lorenzo Valeriani‡
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli studi di Roma "Tor Vergata"
Roma, Italia
*daniele.ferrarelli@alumni.uniroma2.eu
†marco.ferri.98@alumni.uniroma2.eu
‡lorenzo.valeriani.459326@alumni.uniroma2.eu

***Abstract*—In questo documento si illustra una soluzione implementativa di uno storage chiave-valore distribuito per edge computing scritta in Go.**

***Index Terms*—sistemi distribuiti, peer-to-peer, edge computing, cloud computing, distributed system, key-value storage**

I. Introduzione

Le reti peer-to-peer rappresentano una soluzione comune per il problema dello storage distribuito. Tra queste, quelle strutturate si prestano particolarmente alle tipologie di storage chiave-valore. Attraverso tecniche come il consistent hashing, si supportano le operazioni di ricerca delle informazioni in modo efficiente L'edge computing è un paradigma di computazione che estende il cloud tradizionale con le funzionalità di computazione e storage offerte da nodi localizzati ai bordi della rete e quindi vicino agli utenti. La soluzione proposta sfrutta una rete peer-to-peer strutturata per indicizzare le risorse distribuite sui vari nodi edge, cercando di trarre beneficio da entrambe le realtà. Il protocollo utilizzato dalla rete peer-to-peer è Kademlia,nell'implementazione in Go fornita da IPFS. I dati sono replicati su più nodi affinché il sistema sia tollerante ai guasti. La consistenza tra le repliche dei dati è affidata all'algoritmo di consenso distribuito Raft[1][2]. Il progetto fa largo uso dei servizi cloud forniti da Amazon Web Services, quali RDS, Lambda e DynamoDB. La comunicano tra i nodi e con i client è attuata tramite chiamate gRPC.

II. Topologia

I nodi sono collegati per formare una rete peer-to-peer strutturata, su questa è presente un'ulteriore serie di collegamenti che costituisce la rete di replicazione dei dati. Quando si inserisce un nuovo dato nel sistema, questo sarà memorizzato almeno nel nodo che ha ricevuto l'operazione di inserimento, per far sì che la latenza di rete tra l'utente ed il dato sia minima. Quest'ultimo nodo sarà il responsabile di quel dato, identificato da una chiave. Oltre a questo, il dato sarà replicato su un insieme di nodi che possono essere contattati in caso di fallimento del responsabile

A. Rete di indexing

Il sistema sfrutta la DHT di Kademlia per associare ad ogni chiave memorizzata l'indirizzo IP del nodo responsabile. Così facendo si mantiene l'efficienza del lookup di una rete strutturata (O(log N)), permettendo la possibilità di posizionare il dato in un posizione vicina al client che vi accede.

B. Rete di replicazione

Raft, Dettagli implementativi da qui in giü

III. Storage locale

Interfaccia DB, Badger, Redis e transazioni

IV. Consistenza e tolleranza ai guasti

Gli errori supportati sono solamente errori temporanei: se un nodo subisce crash, si assume che questo tornerà ad essere disponibile dopo un intervallo finito di tempo.

V. Migrazione

Sliding window, single thread che valuta e ricerca binaria

VI. Integrazione con il cloud

RDS, Lambda, Gateway, DynamoDB con offloading e registrazione

VII. Limitazioni

Max size dynamo, leave non ancora supportata, join e leave costose lineare per il numero di chiavi e per il numero di nodi(rete strutturata), delete sulla dht non implementata

VIII. Sviluppi futuri

SSS, possibile implementazione della leave, fantascientificamente network proximity

IX. Testing

Come abbiamo fatto i test ed i risultati

X. Manuale d'uso

Dettagli su docker, porta utilizzata 50051,42424

A. Installazione

Utilizzo dei docker

B. Configurazione

File di configurazione

C. Esecuzione

Eseguire come docker che come non docker, all'inizio aspetta di avere una numero di nodi sufficiente

References

[1] ThetaRangers. *The Raft Consensus Algorithm*. URL: https://raft.github.io.

[2] Hashicorp. *Raft*. URL: https://github.com/hashicorp/raft.