Distributed Key-Value Store with Fault Tolerance

Architetture dei Sistemi Distribuiti

Carlo Di Cicco
Valerio Montanaro

Luglio 2024

Contents

1 Introduzione

In questo documento, viene descritta la progettazione e l'implementazione di un archivio distribuito di valori-chiave con tolleranza ai guasti. L'obiettivo principale è garantire la disponibilità e la consistenza dei dati attraverso la replica e la gestione dei guasti.

2 Descrizione del Sistema

2.1 Architettura del Sistema

L'architettura del sistema è composta da vari componenti che interagiscono tra loro per gestire le richieste di lettura e scrittura, mantenere la consistenza dei dati e garantire la tolleranza ai guasti.

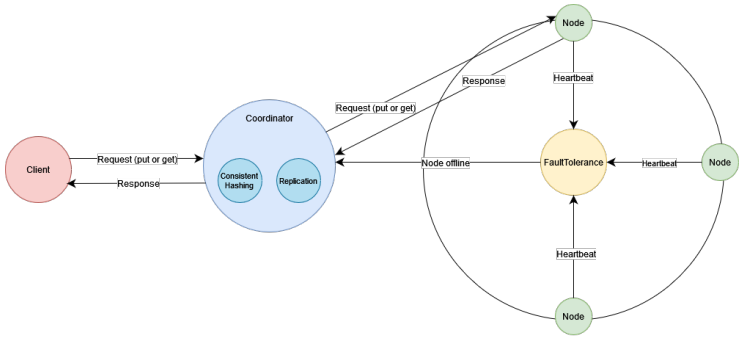

Figure 1:
Diagramma dell'architettura del sistema nel caso in cui sono presenti tre nodi.

2.1.1 Client

Il *Client* invia richieste di lettura (GET) e scrittura (PUT) al sistema. Le richieste vengono indirizzate al *Coordinator*, che risponde al *Client* dopo aver processato la richiesta.

2.1.2 Coordinator

Il *Coordinator* è il componente centrale che gestisce le richieste di lettura e scrittura dei *Client*. Si avvale di due moduli interni: il *Consistent Hashing* e la *Replication*.

- **Consistent Hashing**: Questo modulo determina l'insieme di *Node*, in base al *replication_factor*, responsabili per ogni chiave , permette di distribuire i dati in modo uniforme tra i *Node* e facilita la rimozione di *Node* senza una significativa riorganizzazione dei dati.

- **Replication**: Questo modulo si occupa di eseguire le operazioni sui *Node* precedentemente individuati. In particolare in base alle richieste:
  - PUT: Invia i dati ai *Node* e attende conferme.
  - GET: Raccoglie i dati dai *Node* e determina il valore da restituire.

2.1.3 Node

I *Node* rappresentano i singoli server nel sistema distribuito. Oguno di essi infatti:

- Gestisce localmente le operazioni di lettura e scrittura.
- Comunica periodicamente con il *Fault Tolerance Node* inviando heartbeat per indicare che è attivo.

I *Node* rispondono alle richieste del *Coordinator* e memorizzano le repliche dei dati per garantire la disponibilità e la tolleranza ai guasti.

2.1.4 Fault Tolerance Node

Il *Fault Tolerance Node* monitora lo stato dei *Node* del sistema:

- Riceve heartbeat dai *Node* per verificare la loro operatività.
- Rileva i guasti dei *Node* se non riceve heartbeat entro un intervallo di tempo predefinito.
- Notifica il *Coordinator* quando rileva che un *Node* è offline.

Questo può essere considerato un nodo speciale, esso è cruciale per mantenere l'affidabilità del sistema, garantendo che i guasti dei *Node* vengano gestiti tempestivamente.

2.1.5 Flusso delle Operazioni

- Il *Client* invia una richiesta di PUT o GET al *Coordinator*.
- Il *Coordinator* utilizza il modulo di *Consistent Hashing* per determinare i *Node* responsabili e invia le richieste di PUT o GET ai nodi appropriati mediante *Replication*.
- I *Node* elaborano le richieste e rispondono al *Coordinator*.
- Il *Coordinator* aggrega le risposte e risponde al *Client*.
- I *Node* inviano periodicamente heartbeat al *Fault Tolerance Node* per indicare la loro operatività.
- Se un *Node* non invia heartbeat entro il tempo predefinito, il *Fault Tolerance Node* notifica il *Coordinator* che il *Node* è offline.

- Il *Coordinator* gestisce il failover e la replica di nuovi dati o, vecchi dati per cui si hanno richieste di lettura, su altri nodi attivi.

Questa architettura assicura che il sistema rimanga operativo e coerente anche in presenza di guasti.

3 Modello di Consistenza

È stato scelto un modello di consistenza basato sul quorum per il bilanciamento tra disponibilità e consistenza. Questo modello richiede che un'operazione di lettura o scrittura raggiunga un numero minimo di repliche (quorum) per essere considerata riuscita.

3.1 Quorum di Scrittura

Quando un dato viene scritto, deve essere replicato con successo su almeno *quorum_write* repliche. Questo assicura che il dato sia persistente anche in presenza di guasti di alcuni *Node*.

3.2 Quorum di Lettura

Quando un dato viene letto, deve essere letto con successo da almeno *quorum_read* repliche. Questo garantisce che il dato letto sia aggiornato rispetto alle ultime scritture.

4 Esperimenti e Risultati

Il sistema è stato testato simulando cadute dei *Node* per valutare la tolleranza ai guasti. I risultati mostrano che il sistema è in grado di tollerare guasti senza perdere dati e che la latenza delle operazioni è accettabile.

5 Configurazione del sistema

Di seguito vengono elencate alcune scelte importanti effettuate nella configurazione del sistema:

- Affinché il sistema funzioni è necessario che il numero di *Node* sia strettamente maggiore del *replication_factor*. Questo garantisce che il sistema sia distribuito oltre che funzionante in caso di caduta di un nodo.
- Se un *Node* va offline, i parametri di *quorum_read* e *quorum_write* del sistema decrementano di uno a prescindere da quale dato bisogna leggere o scrivere.

- Nel Consistent Hashing, all'interno dell'anello si fa uso di repliche virtuali dei *Node* per ottenere una distribuzione dei dati all'interno del sistema più uniforme possibile.
- Quando si risponde ad una richiesta di lettura, il sistema restituisce sempre il primo valore disponibile nel caso in cui viene soddisfatto il *quorum_read*. Questa scelta favorisce la disponibilità a discapito della consistenza.

6 Attuali limiti e possibili sviluppi futuri

Il sistema di storage distribuito è in grado di sostenere il guasto dei *Node*, ma non il possibile recupero né l'aggiunta di nuovi *Node* mentre il sistema sta offrendo il servizio. Tuttavia, il sistema risulta sicuramente scalabile e queste caratteristiche potrebbero essere implementate aggiungendo nuovi metodi e endpoint di comunicazione al codice già esistente senza modificarlo.

7 Conclusioni

In questo progetto, è stato implementato un sistema distribuito di valori-chiave con tolleranza ai guasti. Si sono esplorate le sfide su bilanciamento consistenza e disponibilità e valutato le prestazioni del sistema attraverso esperimenti. I risultati indicano che il sistema è robusto e capace di gestire guasti senza compromessi significativi sulla disponibilità dei dati.