Tabele hash distribuite. Sistemul Chord

Cum găsim rapid datele într-un ocean de noduri?

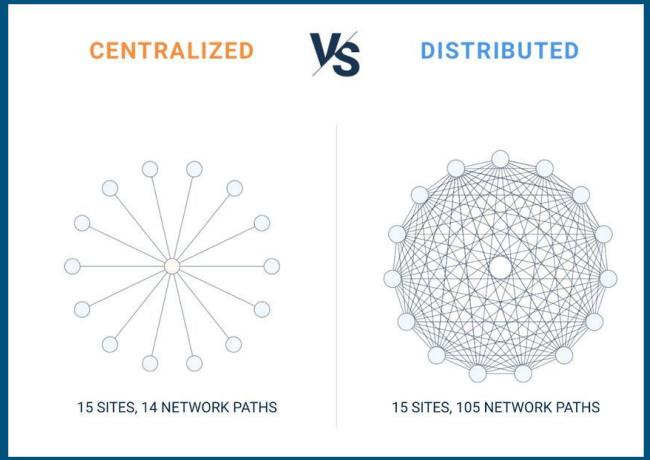
Sisteme distribuite

Dincă Andrei

Grupa 343

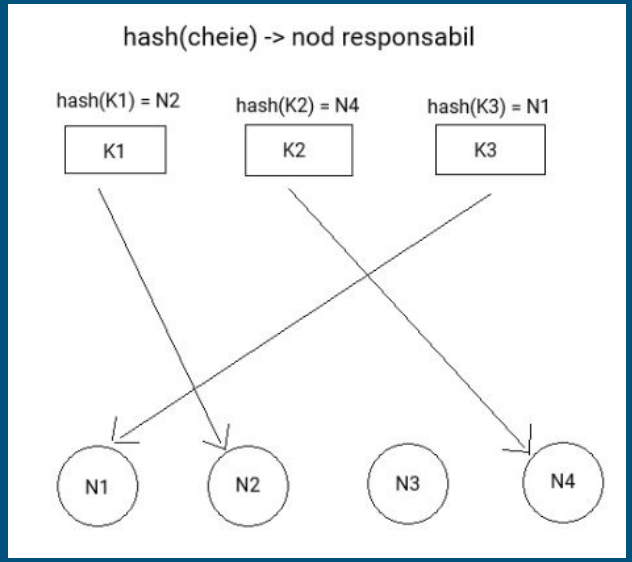
Problema: unde se află datele?

- Sistem distribuit: multe noduri, date răspândite
- Întrebarea de bază: „Pe ce nod găsesc cheia X?”
- Variante naive:
  - Server central -> punct unic de eșec, aglomerare
  - Broadcast către toate nodurile -> trafic mare, nu e scalabil
- Avem nevoie de o metodă eficientă și distribuită de localizare

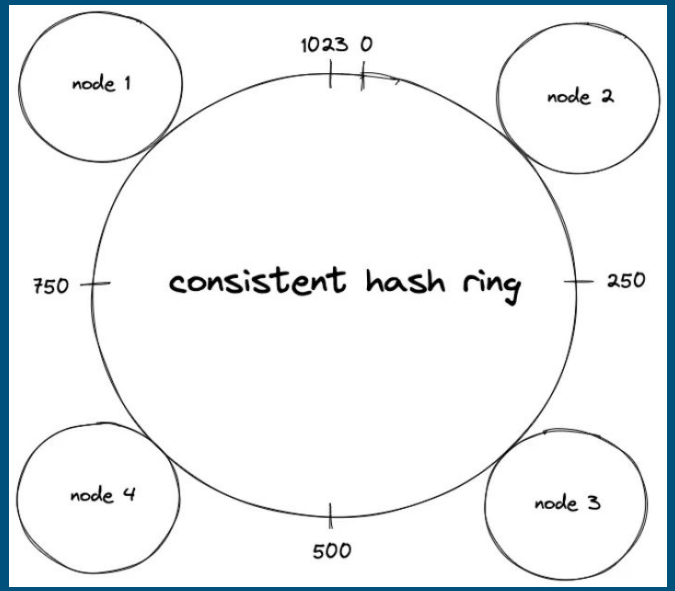


Ce este o Tabelă Hash Distribuită (DHT)?

- Structură logică pentru stocarea perechilor (cheie, valoare) pe mai multe noduri
- Fiecare cheie este mapată (prin hash) la un anumit nod responsabil
- Operații de bază:
  - put(key, value)
  - get(key)
- Proprietăți dorite:
  - Distribuire uniformă a cheilor
  - Scalabilitate (funcționează și cu un număr mare de noduri)
  - Fără nod central



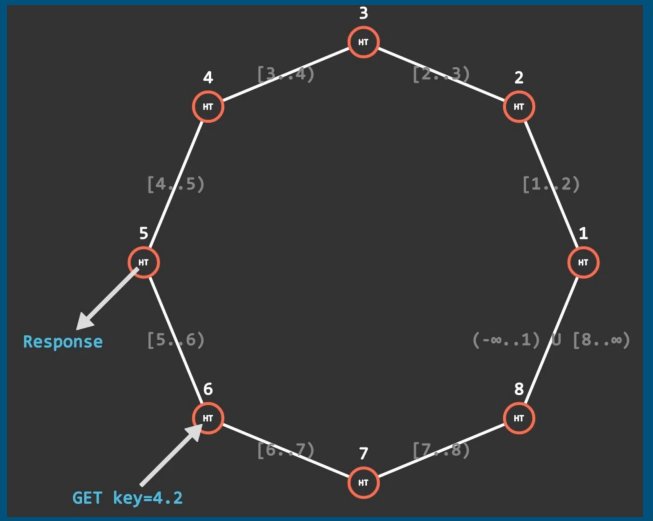
Hashing consistent și spațiul de identificatori



- Folosim o funcție hash pentru a transforma:
  - Nodurile în identificatori (ID_nod)
  - Cheile în identificatori (ID_cheie)
- Toți identificatorii sunt în același spațiu 0...2^m - 1
- Spațiul de identificatori este organizat circular (inel)
- Fiecare cheie este atribuită unui nod din inel

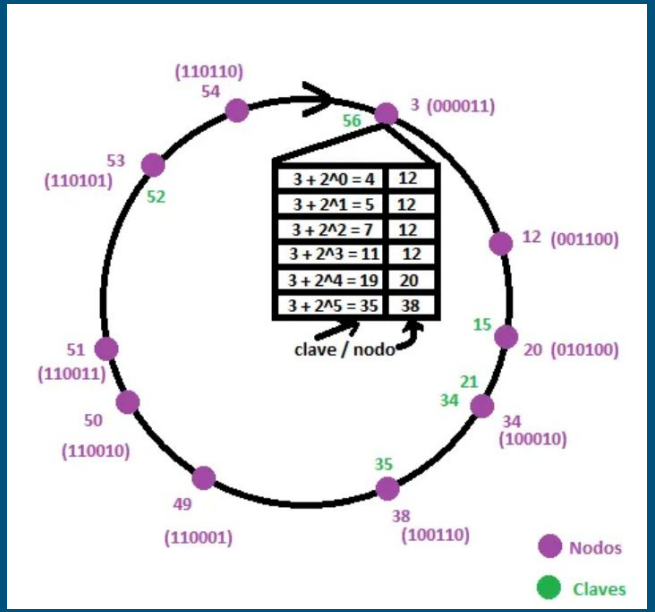
Organizarea în Chord: inelul de identificatori

- Nodurile formează un inel logic (Chord ring)
- Fiecare nod știe:
  - succesorul (următorul nod pe inel)
  - predecesorul (nodul dinainte)
- Interval de chei pentru un nod n:
  - nodul n stochează cheile din (ID_predecesor(n), ID_n]
- Finger table:
  - Link-uri către nodurile mai îndepărtate de pe inel



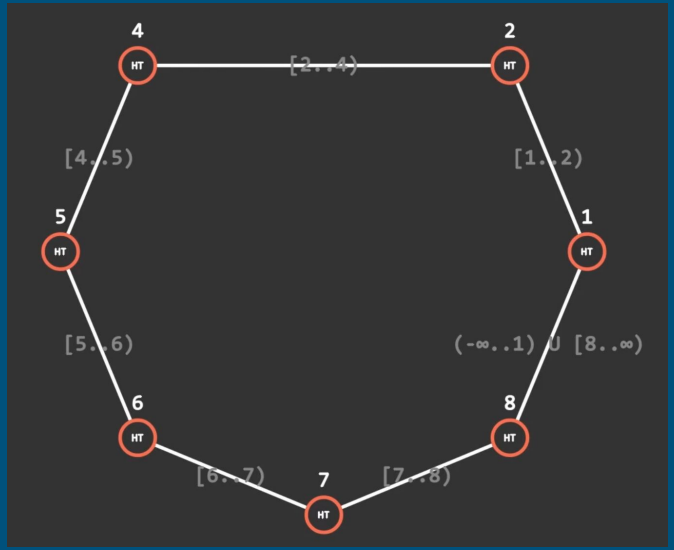
Căutarea unei chei în Chord (lookup)

- Vrem să găsim nodul responsabil pentru ID_cheie
- Algoritmul la nodul n:
  - Dacă ID_cheie este în intervalul (ID_n, ID_succesor(n)]:
    - succesor(n) este nodul responsabil
  - Altfel:
    - Alege din finger table nodul cu ID < ID_cheie, cel mai mare posibil
    - Trimite cererea acolo
  - Nodul următor repetă acești pași
- Lookup-ul are complexitate medie O(log N) hop-uri



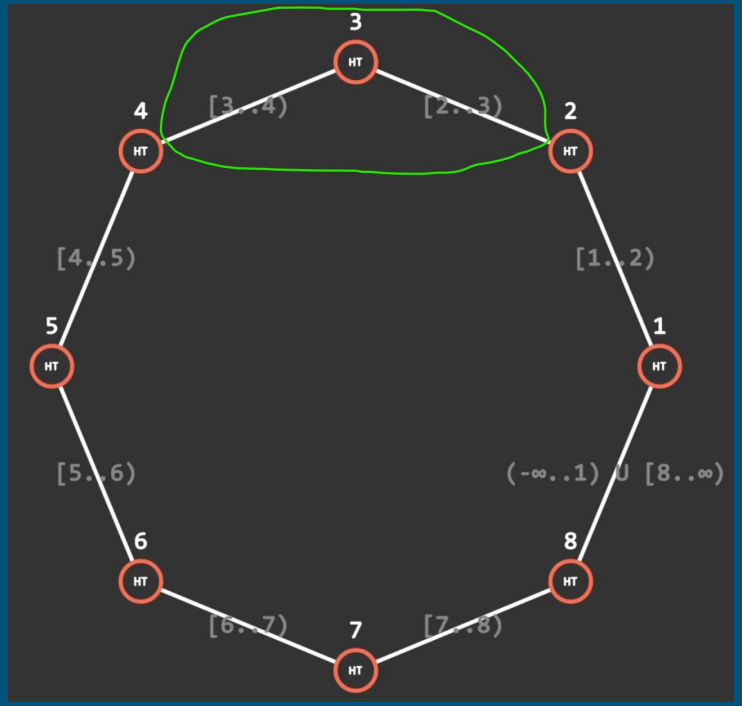
Ieșirea nodurilor și stabilizarea inelului

- Ieșirea controlată:
  - Nodul transferă cheile succesorului său
- Cădere bruscă (crash):
  - E detectată prin timeouts / conexiuni eșuate
- Copii de rezervă:
  - Se pot păstra mai mulți succesori (succesor list)
- Stabilizare periodică:
  - Nodurile verifică dacă succesorul / predecesorul sunt corecți
  - Repară legăturile și actualizează finger table-urile



Noduri care intră în sistem (join în Chord)

- Noul nod își calculează ID_n = hash(adresa lui)
- Contactează un nod deja existent în inel
- Folosește find_succesor(ID_n) ca să-și afle succesorul
- Setează:
  - Succesor și predecesorul
  - Preia cheile pentru care devine responsabil
- Ulterior:
  - Se actualizează finger table-urile prin proceduri de „stabilizare”



Chord – ce oferă și unde se folosește

- Proprietăți:
  - Căutare (lookup) în O(log N) hop-uri
  - La join/leave se redistribuie doar cheile „locale”
- Avantaje:
  - Scalabil și decentralizat
  - Se adaptează la noduri care intră și ies
- Limitări:
  - Suportă doar căutări simple pe cheie
  - Overhead pentru menținerea inelului și a finger table-urilor
- Aplicații:
  - Sisteme peer-to-peer (P2P)
  - Stocare / cache distribuit

Bibliografie

- Distributed Hash Tables: In a nutshell (Reupload)
- Deep dive into Chord for Distributed Hash Table
- Chord - A Distributed Hash Table
- Chord Wikipedia (poze)
- Chord Wikipedia
- Chord - A Distributed Hash Table (YouTube)