Protocolul Chord

Vrabie Victor, B6

Introducere

Chord este un protocol și un algoritm pentru retelele peer-to-peer care utilizeaza tabele hash distribuite. Un tabel hash distribuit păstrează perechi (cheie, valoare), mapand chei în fiecare nod care este responsabil sa depoziteze obiectul valoare. Chord specifica cum anume asignam unui nod o cheie și cum căutam valoarea corespunzatoare cheii după cheie.

Descriere detaliata a protocolului

Inelul Chord si functia hash

Fie un m – numar întreg astfel încât numărul maxim de noduri în rețea sa nu depaseasca 2^m,atunci nodurilor și respectiv cheilor li se vor asigna valoarea unei hash-functii peste cheie și respectiv peste nod (peste IP-ul nodului) pastrand ultimii m biti din valoarea furnizata de hash-functie.Avem nevoie de o funcție hash persistenta (probabilitatea unei coleziuni este neglijabila și care formeaza o distribuite uniforma a valorilor în spațiu de căutare) care sa garanteze unele proprietăți pentru a putea utiliza cu siguranța protocolul considerandul ca are un grad înalt de eficiența.

Ca functie hash va fi utilizata functia SHA1 din libraria *openssl/sha.h*
Nodurile ,respectiv cheile vor fi etichetate cu un identificator de la 0 la 2^m-1 (cu m suficient de mare pentru a evita coleziunile).

Fiecarui nod ii va corespunde un nod succesor (fie n nodul ,atunci n.succesor este succesorul lui n în ordine anti-trigonometrica pe inelul format de spațiul de distributie a cheilor (0 2^m-1),respectiv un predecesor (n.predecesor) primul nod care apare în ordine trigonometrica.

Succesorul cheii *k* este nodul care are ID (eticheta corespunzatoare lui) *k* sau următorul după *k* în sensul antritrigonometric.Deci pentru depistarea nodului care mapeaza valoarea cheii *k* este necesar un sir de apeluri recursive care ofera o complexitate de O(N) (pentru a depista o cheie va fi necesar sa trimitem mesaje tuturor nodurilor din rețea).Pentru a câștiga în eficiența se introduce așa numitul finger_table despre care se va expune în continuare.

Finger table si calcularea lui

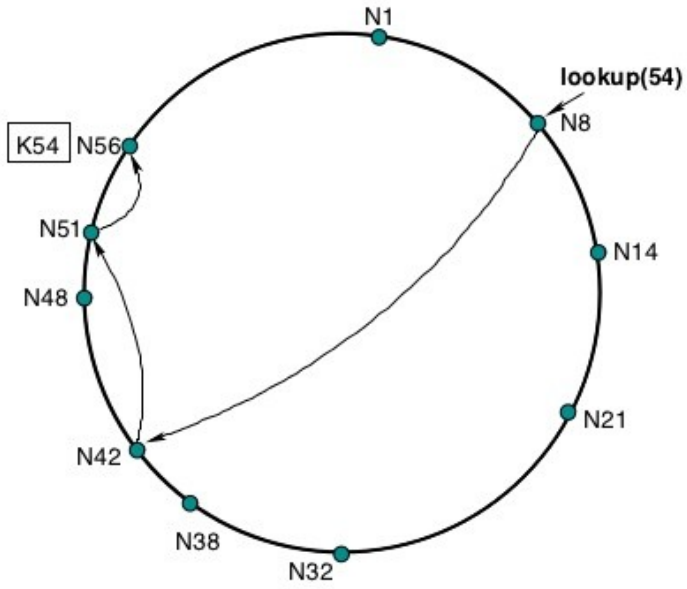
Utilizarea trucului cu finger_table permite atingerea unei limite maxime de log N mesaje transmise nodurilor din rețea pentru a depista o cheie.

Pentru căutarea eficiența , specific în continuare semnificatia acestui tabel: `figer_table[i]= succesorul cheii n+2^i (mod 2^m)` **(0<=i<m)**. Astfel, succesorul lui n este dat de `finger_table[0]`.

Chord protocol

```
// ask node n to find the successor
// of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// search the local table for the
// highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;
```



În imaginea de mai sus este prezentat pseudo-codul aplicabil pentru determinarea eficienta a succesorul unei chei.

Studiem acum finger_table-ul corespunzător nodului N8:

Aici m=6,deci avem pe inel valori din intervalul [0;63];
finger_table[0]=succesorul lui (8+2^0) mod 64= succesorul lui 9 = 14
finger_table[1]=succesorul lui (8+2^1) mod 64=succesorul lui 10= 14
finger_table[2]=succesorul lui (8+2^2) mod 64=succesorul lui 12= 14
finger_table[3]=succesorul lui (8+2^3) mod 64=succesorul lui 16= 21
finger_table[4]=succesorul lui (8+2^4) mod 64=succesorul lui 24= 32
finger_table[5]=succesorul lui (8+2^5) mod 64=succesorul lui 40= 42

Simulam acum căutarea cheii K54 în rețea dacă apelul de căutare se face din nodul N8:

1. Mai întâi privim dacă cheia nu se afla intre nodul curen și succesorul lui în acest caz succesorul cheii cautate ar fi fost succesorul nodului curent, deoarece 54 nu se incadreaza în intervalul (8;14].
2. Căutam în finger_table cea mai mare valoare (în sensul deplasarii dinspre nodul curent în sens antitrigonometric) care este în intervalul [nod curent;cheie căutată]; în cazul nostru căutam cea mai mare valoare din finger care se incadreaza în intervalul [8;54] care este 42 și deci continuam căutarea succesorului din nodul 40 care devine nod curent; revenind iar la pasul 1.

3. Deoarece 54 nu se incadreaza în intervalul [42;succesorul lui 42=48], trecem la pasul 2 și obținem un nou nod 51 extras din tabela finger_table corespunzatoare nodului 42. Deja 54 se incadreaza în intervalul [51;succesorul lui 51=56], și deci succesorul cheii 54 este 56 deci valoarea corespunzatoare cheii 54 va fi stocata la nodul 56 (nodul poate stoca fisiere, si cheia corespunzatoare fisierului poate fi valoare hash-functiei peste numele fisierului sau peste continutul fisierului în caz ca exista mai multe fisiere cu acest nume).

Dinamismul aparitiei nodurilor

Deoarece o rețea peer-to-peer este una foarte dinamica,care permite intrarea în rețea a unor noduri și deconectarea altora, trebuie ca conectarea unui nod și stabilirea informatiei necesare lui pentru a putea exista în rețea să se efectueze eficient și respectiv la plecarea unui nod din rețea restabilirea legaturilor dintre celelalte noduri existente.

Pseudo-codul pentru operație de join și de stabilizare este dat mai jos:

```
n.create()
  predecessor = nil;
  successor = n;

// join a Chord ring containing node n'.
n.join(n')
  predecessor = nil;
  successor = n'.find_successor(n);

// called periodically. n asks the successor
// about its predecessor, verifies if n's immediate
// successor is consistent, and tells the successor about n
n.stabilize()
  x = successor.predecessor;
  if (x∈(n, successor))
    successor = x;
  successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
  if (predecessor is nil or n'∈(predecessor, n))
    predecessor = n';

// called periodically. refreshes finger table entries.
// next stores the index of the finger to fix
n.fix_fingers()
  next = next + 1;
  if (next > m)
    next = 1;
  finger[next] = find_successor(n+);

// called periodically. checks whether predecessor has failed.
n.check_predecessor()
  if (predecessor has failed)
    predecessor = nil;
```

Pentru mentinerea retelei la perfomante constante este necesar periodic de a apela din fiecare nod procedura de stabilizare care restabileste legăturile și tabelul finger_table pentru fiecare nod, obținând astfel o rețea adaptabila.

Un alt aspect important al acestui protocol este măsurile luate în cazul în care cade un nod din rețea fără ca sa anunțe ieșirea.Netratarea acestui caz ar duce la pierderea unor obiecte importante care sunt stocate la nodul în cauza , și deci Chord prevede sa pastreze multiplicat fisierele fiecarui nod astfel putandu-se de restabilit fisierele care au căzut să fie stocate de nodului în cauza. (mai multe detalii vor fi adaugate în faza de implementare).

Intrarea în rețea a unui nod nou se face prin generarea unui numar random și aplicarea asupra lui a unei hash-functii astfel obtinandu-se eticheta noului nod, după care se executa transmiterea unui mesaj către unui nod deja inclus în rețea pentru a cauta succesorul corespunzător nodului care urmează a fi introdus și completarea în continuare a datelor necesarea existenței nodului în rețea (finger_table,succesorul,predecesorul,valorile stocate care intra în responsabilitatea noului nod).

Amintesc faptul ca atunci când un nod intra în rețea ,succesorul lui este obligat sa cedeze valorile corespunzatoare cheilor care sunt prevazute de algoritm ca să fie gestionate de nodul nou introdus, si atunci când un nod părăsește reteaua și anunța acest fapt atunci succesorul sau este obligat sa preia datele gestionate de nod și sa le gestioneze el în continuare(cazul când un nod părăsește reteaua din cauza unei erori, nu se pot lua măsuri asupra acestui fapt decât dacă replicam valorile gestionate de el, ceea ce va complica mult implementarea acestui detalui).

Detalii de implementare

Functia hash utilizata este SHA1 implementata in libraria openssl/sha.h
si se utilizeaza in felul urmator: pentru fiecare nod concatenez intr-un sir de caractere IP si PORTUL nodului respectiv, aplicand ulterior functia SHA1 si returnand 8 octeti din rezultatul returnat de functie :

```
 int eticheta(char * s1,char * s2)
{
static char s[30];
static char hash[30];
s[0]=0;
strcat(s,s1);
strcat(s,":");
strcat(s,s2);
SHA1(s,strlen(s),hash);
        //utilizez primii 8 biti din hash
return (hash[0]+256)&(255);
}
```

Aceeasi functie o utilizez si pentru a afla cheia respectiva pentru o pereche
(trebuie sa mentionez ca o pereche inseamna 2 siruri de caracter nume prenume pentru a demonstra functionalitatea protocolului).

Fiecare nod are partea sa de server TCP concurent:

```
pthread_mutex_lock(&lock);
if ((nod_partener=accept(sock,(struct sockaddr *)&adr,&len))<0)
{
perror("eroare la apelul accept");
exit(-1);
}
     thread[(int) arg].client=nod_partener;
    pthread_mutex_unlock(&lock);
```

Utilizez o variabila partajata de toate thread-urile care imi asigura accesul mutual la zona de accept si pentru a evita ciocnirile thread-urile pe memoria partajate utilizez un vector de structui un aloc memorie suficienta ,astfel incat fiecare thread stiindu-si id-ul are acces exclusiv la memoria alocata fiind al id-lea element din vectorul de structuri:

```
typedef struct thread_data
{
pthread_t id;
uint32_t client;
node nod;
char msg[30];
char op;
uint32_t k;
} _th;


_th thread[100];
```

Implementez multe operatii, si uniformizez trimiterea de mesaje. Pentru un string char msg[30]:
<cod operatie pe 1 byte><4 bytes eticheta><IP[0] pe 4 bytes><IP[1] pe 4 bytes><IP[2] pe 4 bytes><IP[3] pe 4 bytes><PORT pe 2 bytes>

Mesajele se trimit cu ajutorul functiei *`void trimite_mesaj(struct sockaddr_in * adresa_nod,char * msg);`* *care* primeste adresa destinatarului si mesajul si in spate se face un connect cu tratarea eroarilor respective si scrie rezultatul obtinut in aceeasi zona de memorie care cincide cu zona mesajului de trimis.

Alte functii implementate:

`void impacheteaza(char * msg, char cod_operatie, uint32_t eticheta, uint32_t * IP, uint16_t PORT);`

`void despacheteaza(char * msg,char * code_operatie,node * t_node);`
(are loc serealizarea mesajelor)

Functii care sunt apelate la crearea thread-urilor:

`void creat_thread_i(int i);`

`void * thread_nou_pentru_citire_comenzi(void *);`

Concluzie:

Protocolul Cord este o solutie eleganta si usor implementabila pentru o retea distribuita descentralizata.Tin sa amintesc ca acest protocol sustine operatii a caror complexitate este usor demonstrabila si ca cu o foarte mare probabilitate, rămâne robust si scalabil.

Bibliografie:

https://en.wikipedia.org/wiki/Chord_(peer-to-peer)
https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf