ReadMe – Tema 2 APD – CHORD MPI

Bîrsan Alexandru Vlad
333CC

1. Introducere

Timp de lucru: 4h

Efectuare rutare eficienta O(logN)

2. Implementare

void build_finger_table()

Am calculat start[i] folosind formula: $start_i = (n.id + 2^i) \mod 2^m$

Iar variabila de node din finger[i] a devenit succesorul acestui start[i] folosind functia find_successor_simple(self.finger[i].start) .

int closest_preceding_finger(int *key*)

Parcurg vectorul de finger invers (int i = M - 1; i >= 0; i--) ca sa fac match pe cel mai mare pas posibil (cel care il preceda pe cel care detine key ul).

In loop am 2 verificari prima si cea mai importanta este cea care verifica daca key ul este egal cu variabila node a finger[i] current si cea de a doua cea in care self.id este egal cu self.finger[i].node pentru ca aceste doua nu fac progres si pot duce la bucle.

Dupa aceste verificari pot folosi direct functia data in scheletul temei

```
if (in_interval(self.finger[i].node, self.id, key)) {
        return self.finger[i].node;
}
```

aici verific daca nodul din finger se afla strict in intervalul (self.id, key) si daca acest lucru este adevarat returnez valoarea node pe care am gasit-o (cea mai mare posibila)

Altfel daca se termina loop-ul se ajunge la fallback ul dat deja in schelet care returneaza succesorul direct al nodului curent pentru a continua rutarea.

void handle_lookup_request(LookupMsg *msg)

Adaug la index-ul path_len in path self.id pentru a tine evidenta traseului si incrementez path_len pentru a o putea folosi in continuare.

Ulterior daca succesorul nostru este responsabil de cheie(cheia se afla intre self.id si self.succesor) il adaug si pe el in path si incrementez mai departe path_len. Aici adaug si TAG_LOOKUP_REP in mesaj cand il trimit direct spre initiator pentru ca a fost gasit nodul resp de cheie.

Daca cheia nu se afla in intervalul acesta adica nu succesorul este responsabil de aceasta trimit mesajul cu TAG_LOOKUP_REQ catre closest_preceding_finger(*msg*→key) care gaseste cel mai bun nod urmator.

int main(int *argc*, char **argv)

inițierea lookup-urilor locale

Dupa ce se construiesc Inelul si finger table-ul pentru fiecare cheie este nevoie de instantierea unei structuri LookupMsg msg.
In care am setat:

```
msg.initiator_id = self.id;
msg.current_id = self.id;
msg.key = lookups[i];
msg.path_len = 0;
```

Dupa care trebuie trimis mesajul cu MPI_Send catre propriul rank(rank_from_id(self.id)) folosind tag ul TAG_LOOKUP_REQ.

service-loop distribuit

Am folosit un loop while care ruleaza cat timp variabila completed (care numara cate noduri au terminat executia totala) este mai mica decat world_size.

In interiorul buclei primesc mesaje cu MPI_Recv de la MPI_ANY_SOURCE si verific tag-ul:

Status-ul poate avea 3 cazuri pe care le tratez:

- Daca Tag-ul e REQ:
  - handle_lookup_request(&msg);
- Daca Tag-ul e REP:
  - Afisez path ul mesajului si decrementez left_lookups(variabila care contorizeaza cate raspunsuri mai astept la cererile initiate de mine)
- Daca Tag-ul e DONE:
  - Incrementez variabila completed(contor de terminare – aceasta opreste loop-ul)

In final trimit TAG_DONE la fiecare nod din retea (printr-un for de la 0 la world_size), dar doar daca left_lookups == 0 (am terminat treaba mea locala) si variabila sent este 0. Dupa trimitere, setez sent = 1 pentru a ma asigura ca trimit semnalul de terminare o singura data, evitand inundarea retelei.