DOCUMENTAZIONE PROGETTO RETI INFORMATICHE A.A. 2020/2021

Architettura della rete

Ogni peer ha come vicini i due peer con numero di porta più vicino, perciò la rete peer-to-peer nel suo complesso ha una struttura ad anello, in cui ogni nodo ha un vicino che lo precede ed un vicino che lo segue.
Per rispettare tale struttura, ho dovuto considerare come vicini il peer con numero di porta minore ed il peer con numero di porta maggiore.
La rete include inoltre un discovery server (DS) centrale che gestisce l'ingresso e l'uscita dei peers dal network.

I peer vicini comunicano tra loro mediante connessioni TCP permanenti, stabilite al momento della registrazione di un nuovo peer e chiuse al seguito di una disconnessione di un peer dalla rete.
Le comunicazioni con il DS avvengono invece attraverso messaggi UDP.

Sottodirectory

Poichè l'applicazione è lanciata dallo stesso host, nella directory vengono generate le sottodirectory relative ad ogni peer rinominate con il loro numero di porta.
All'interno di esse saranno generate all'occorrenza altre sottodirectory:

- *registers:* in cui vengono salvati i register giornalieri. Le entry rispettano tutte il formato "*yyyy:mm:dd*,*type*,*value*", dove type vale *t* (tamponi) o *n* (nuovi positivi);
- *quantities:* in cui si trovano i files con le quantità totali di ogni giornata dopo aver ricevuto le entry da tutti i peer della rete al seguito di un flooding;
- *aggr:* in cui vengono salvati i risultati delle richieste di elaborazione.

Eventualmente possono essere generati due files:

- *PeersList*: in ogni riga ho il numero di porta di un peer da cui ho ricevuto un registers. Mi evita di ciclare tutti i numeri di porta possibili quando ricerco un file nella cartella registers;
- *StopLog*: salva la data dell'ultimo register inviato dal peer durante una disconnessione. Evita di inviare registers duplicati ai neighbors ogni volta che entriamo in fase di disconnessione.

Richieste di elaborazione

Al seguito di una richiesta di elaborazione, il peer verifica se non ha già calcolato il dato aggregato richiesto andando a guardare tra i files della sottocartella *aggr.*
Successivamente, poiché i peer non hanno memoria di quali altri peer facevano parte della rete in un certo periodo, l'unico modo per essere sicuri di avere tutte le quantità necessarie per calcolarsi il dato è quello di verificare che non manchi alcun file relativo al periodo richiesto nella sottocartella *quantities*.
Nel caso in cui manchi qualche quantità, il peer crea una lista di N sottoperiodi mancanti allocando in memoria N strutture con due campi: l'inizio del sottoperiodo (espresso in secondi da epoch) e la durata in giorni dello stesso.
A questo punto manda una richiesta REQ_DATA ai vicini seguita dalla richiesta di elaborazione nel formato "*aggr type yyyy:mm:dd-yyyy:mm:dd*" avente dimensione costante di 27bytes (*aggr* è su 3bytes, mentre *type* su 1 bytes).
Prima di inviare il file con il risultato, i vicini inviano la dimensione del file: se questa è zero vuol dire che non possiedono il risultato ed il peer è costretto ad effettuare il flooding per recuperare tutte le entry del periodo richiesto.

Flooding
Prima di tutto, il peer contatta il DS per conoscere il numero di peers nella rete al momento del flooding, in modo da settare i due counter (uno passato al neighbor precedente ed uno al neighbor successivo) che ad ogni hop del flooding saranno decrementati dai peer riceventi richiesta di flooding: quando un counter raggiunge 0, il flooding non viene più inoltrato e viene contattato il requester. Se entrambi i counter arrivano a 0, il requester è sicuro di aver ricevuto tutti i register possibili e procede con il calcolo del dato aggregato. In questo modo si limita la propagazione della query all'essenziale, limitando l'elevato overhead dovuto alla natura broadcast del meccanismo di ricerca.
Oltre al counter, un peer a cui viene inoltrata la richiesta di flooding riceve anche la lista di N sottoperiodi calcolata precedentemente dal requester, più precisamente riceve N volte due interi, rispettivamente su 32 e su 16 bit. Questo permette, insieme alle informazioni salvate nel file *PeersList*, di ridurre molto i tempi di ricerca dei register e di conseguenza dei tempi di risposta di ogni peer toccato dal flooding.
Se viene trovato un register appartenente ad un sottoperiodo, viene comunicato all'indietro al neighbor richiedente insieme al numero di porta del requester su 16 bit, dopodichè il replier attende che il requester si riconosca in tale numero di porta ed accetti la richiesta di connessione TCP. Stabilita la connessione TCP, il replier invierà prima il nome dei files posseduti, e solo nel caso il requester non li abbia già ricevuti da altri peers sarà inviato anche il contenuto dell'intero register. Ho scelto di non fare distinzioni fra i due tipi di entry ma di inviare l'intero register perché in questo modo evito di accedere ai file riga per riga ma leggo e scrivo a blocchi, notando inoltre che in un contesto reale molto probabilmente le richieste di elaborazione sul numero di tamponi e sul numero di nuovi positivi avverrebbero comunque conseguentemente l'una all'altra essendo dati da correlare fra loro ed in questo modo avremmo già tutti i registers necessari alla prima richiesta, risparmiandoci una seconda richiesta di flooding.

Richiesta di connessione alla rete
Sotto è riportato uno schema in pseudocodice del colloquio tra DS e peer al seguito di una richiesta di connessione alla rete. Ciò che viene chiamato *lista* è una stringa formattata "*IP1:port1;IP2:port2*" e il DS prima di inviarla informa il peer della sua lunghezza in byte.