GoTorrent

Gossip-Based BitTorrent

JORDI BORRELL VIVES-MIHAI SERBAN COPIL

SISTEMES DISTRIBUÏTS

2016-2017

Introducció

L'objectiu d'aquesta tasca és implementar un prototip d'un sistema de BitTorrent per mitjà de la biblioteca Python PyActor. Tot i que moltes parts del protocol estàndard BitTorrent es usaran com a tal, la difusió dels arxius serà a través de protocols de comunicació *gossip* en lloc del habitual mecanisme que usa el BitTorrent.
Aquest estil de comunicació, el *gossip protocol*, ja es pot trobar en molts sistemes distribuïts moderns, com ara Amazon DB i Cassandra (Facebook), per nomenar-ne uns pocs, i es considera una bloc de construcció bàsic per a molts sistemes de gran escala, aquí radica la seva importància.

Decisions de disseny

Tracker:

En el tracker per guardar la informació dels torrents i els peers hi guardem un diccionari on les claus són el nom dels torrents i la informació que hi guardem serà un altre diccionari on les claus d'aquests són referències als peers d'aquest tracker, i la seva informació serà la data i el moment de l'últim cop que s'han anunciat al tracker, per eliminar-los si fa més de 10s que no s'anuncien.

Per retornar 3 peers aleatoris en el getPeers utilitzem la funció sample, així a part de que siguin peers aleatoris del torrent sabem que no seran peers repetits. Per altra banda com que realment amb aquest mètode el que estem enviant són referències a aquests peers, podrem cridar als mètodes d'aquests peers des d'on sigui que ens hagi donat les referències d'aquests el getPeers.

Peers:

Cada peer té la mida en nombre de chunks del torrent que volen, un diccionari amb la informació del torrent, on les claus són el número del chunk i la informació la lletra, que és el que volem del torrent, a més a més també tenen una llista amb els números dels chunks que els falten per a què quan facin pull d'un chunk aleatori, només demanin chunks que encara no tinguin i així hi hagi un rendiment major.

Per a realitzar el push tenim dos mètodes: El pushing, que és el mètode que criden cada segon els peers per fer el push i consisteix en elegir un chunk aleatori dels que té aquest peer amb la funció choice i enviar-lo al peer que ens hagi donat el getPeers cridant a la seva funció push per referència. Aquesta altre mètode push serà en la qual si el peer no té ja aquest chunk que ens han passat el guardà a la seva informació.

Per realitzar el pull, de la mateixa forma que en el push, tenim dos mètodes: El pulling, que és el mètode que criden cada segon els peers per fer el pull i consisteix en demanar a les referències dels peers que ens dóna el mètode getPeers un dels chunks que li falten al peer amb el mètode pull, sempre que aquest peer no tingui ja tota la informació del torrent, en aquest cas aquest peer no faria res. L'altre mètode pull intenta retornar el chunk que es demana, en cas que no el tingui ja es tracta en l'altre mètode una excepció per a que continuï la seva execució.

Per realitzar la combinació híbrida de push i pull, enlloc de cridar directament als mètodes pushing i pulling d'aquests peers hem copiat el codi per a què tan sols es cridi una vegada al getPeers i els peers facin pull i push a cadascun dels peers que se'ls hi hagi passat, ja que el mètode getPeers no es massa ràpid i així ens surten menys errors per timeout i tenim un millor rendiment.

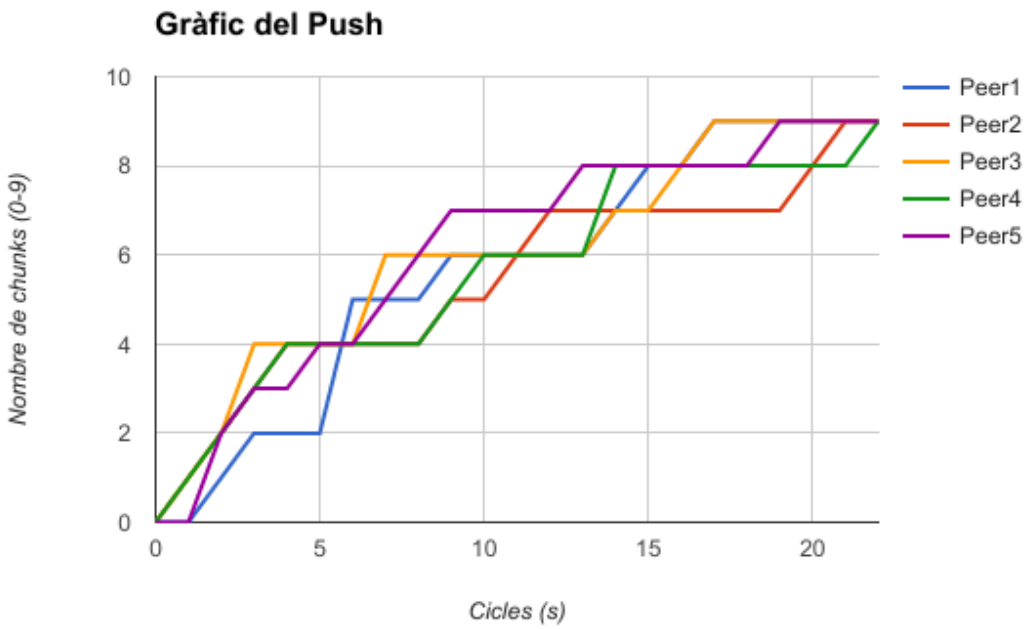
Joc de Proves

Per a comprovar que les diferents funcions de la nostra pràctica funcionen correctament hem creat una funció addicional anomenada infopeer la qual es la encarregada d'ensenyar per consola el numero de chunks que conte cada peer per veure si la transmissió i/o recepció de chunks entre els peers i seeds opera correctament.
Pel que fa el main hem decidit generar un main força bàsic amb la generació de 5 peers diferents i 1 seed que conte tota la transmesa; i amb aquests elements hem anat provant les 3 diferents implementacions tot canviant la part en negreta del main cridant la implementació que volíem provar en cada moment.

```
def infopeer(self):
        print "//////////////////////////////////"
        for x in self.tDictionary:
                print(x)
                for y in self.tDictionary[x]:
                        #refPeer= self.host.lookup(y)
                        print " -"+y.getId()+": "
                        print y.getTorrent()
                print "//////////////////////////////////"


if __name__ == "__main__":
        set_context()
        h = create_host()
        p1 = h.spawn('peer1', Peer)
        p2 = h.spawn('peer2', Peer)
        p3 = h.spawn('peer3', Peer)
        p4 = h.spawn('peer4', Peer)
        p5 = h.spawn('peer5', Peer)
        seed = h.spawn('seed', Peer)
        t = h.spawn('tracker', Tracker)
        t.init_start()

        p1.afegir_host(h)
        p2.afegir_host(h)
        p3.afegir_host(h)
        p4.afegir_host(h)
        p5.afegir_host(h)
        seed.afegir_host(h)
        t.afegir_host(h)

        p1.afegir_tracker(t)
        p2.afegir_tracker(t)
        p3.afegir_tracker(t)
        p4.afegir_tracker(t)
        p5.afegir_tracker(t)
        seed.afegir_tracker(t)


        seed.make_seed()
        sleep(2)
```

```
p1.init_combinado("GOT")
p2.init_combinado("GOT")
p3.init_combinado("GOT")
p4.init_combinado("GOT")
p5.init_combinado("GOT")

seed.init_combinado('GOT')

for x in range(100):
        sleep(1)
        t.infopeer()
```

Preguntes



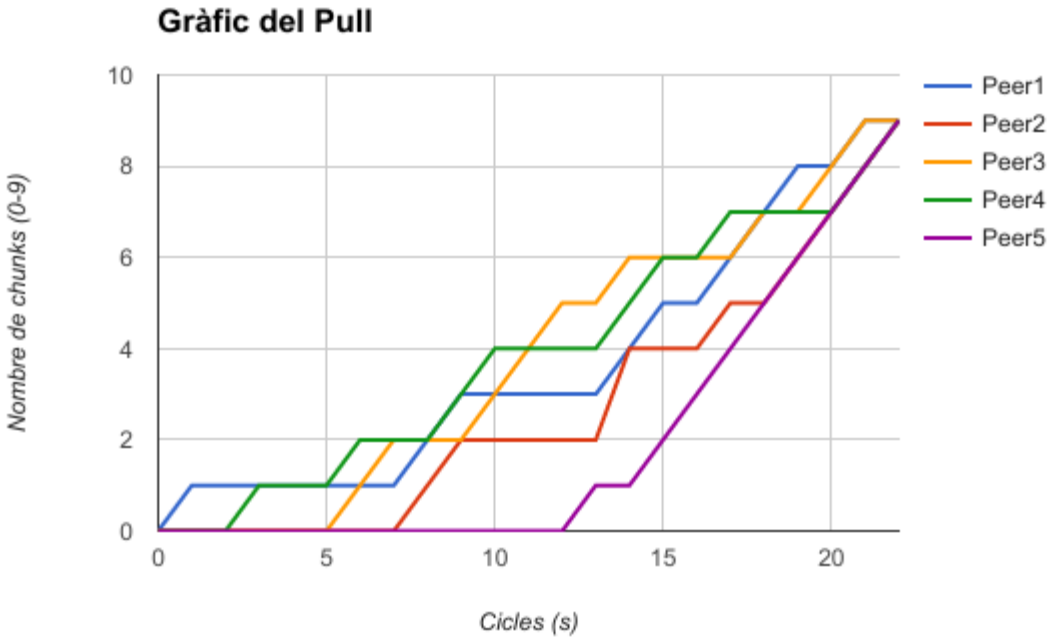
Explica el que es veu a la figura. Creus que hi ha molts missatges redundants?

A la figura es veu com progressa la informació que obtenen els peers durant el temps fins que tots completen la informació que tenen amb un sistema push gossip.

Hi ha d'haver bastants missatges redundants, perquè els Peers fan push aleatòriament a 3 peers, que si dóna la casualitat que algun es repetit o és ell mateix estarien fent realment menys pushs, per tant, hi han d'haver missatges redundants per força, ja que si no fos així en cosa d'uns 6 segons ja haurien de tenir tots els peers la informació, ja que el seed en 3 segons ja podria haver fet push dels seus 9 chunks, i en 2-4 segons més aquests peers haurien pogut donar la informació d'aquests chunks a la resta.

En el nostre cas donat el fet de què nosaltres agafem els peers del mateix torrent de forma aleatòria i fem push dels seus chunks també de forma aleatòria, es generen força missatges redundants i quant menys chunks el hi queden als peers per completar la informació dels torrents més els hi costa que els hi enviïn la informació d'aquests chunks, ja que el fet de que això es doni és aleatori.

Per aquest motiu els gràfics comencen pujant força ràpidament i acaben estancant-se una mica al final i els hi costa més completar del tot la informació del torrent quant els va faltant menys informació per finalitzar.

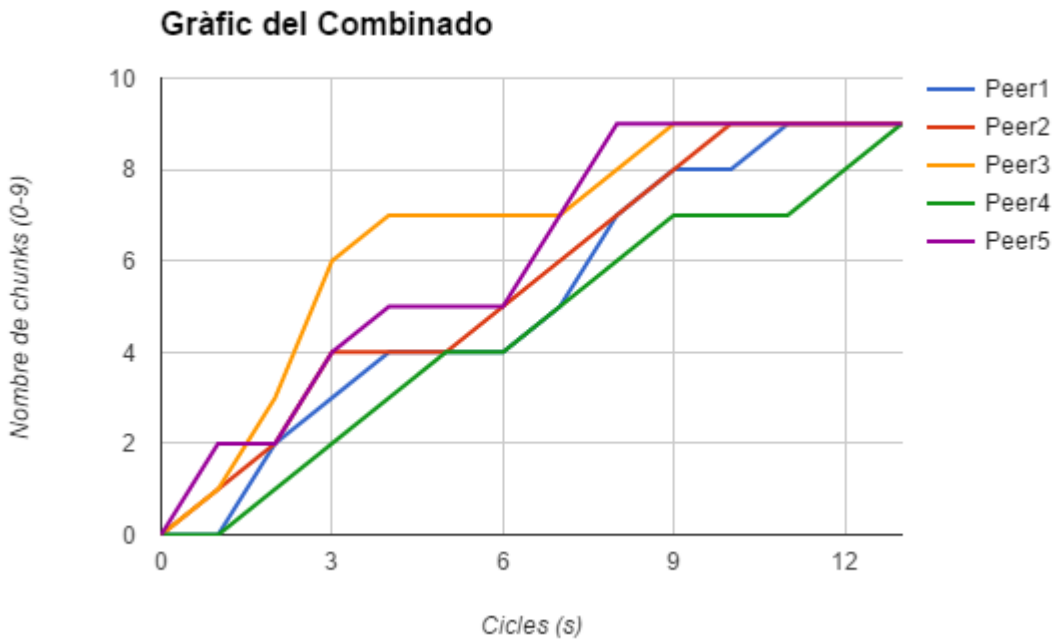


Explica el que es veu a la figura i el compara'l amb el push gossip. Quina de les dues tècniques és més eficient? Per què?

A la figura es veu com progressa la informació que obtenen els peers durant el temps fins que tots completen la informació que tenen amb un sistema pull gossip.

En comparació amb el push, amb la resta de proves que hem fet, amb la nostra implementació i amb aquesta quantitat de peers i seeds, més o menys les dues tècniques són iguals d'eficients, la diferència radica en el fet que en el pull, al principi als peers els hi costa començar a agafar informació (chunks), i un cop els peers van tenint cada vegada més informació els hi costa menys de completar-se, just a la inversa que ens passa en el push.

Això és degut al fet de que a diferència que en el push, en el pull nosaltres hem implementat que cada peer li demani als altres peers que obté amb el getPeers si té un dels chunks que li falten, per aquesta raó al principi tan sols aconseguiran algun chunk d'informació els peers que per casualitat li demanin al seed, per aquesta raó al principi tarda més en aconseguir informació, però un cop tots els peers van obtenint més informació, al anar demanant algun chunk a algun dels peers hi ha més probabilitat de que el pugui obtenir i per això cada vegada obtenen informació més ràpidament.



Explica el que es veu a la figura. Hi ha molta diferència respecte el pull?

A la figura es veu com progressa la informació que obtenen els peers durant el temps fins que tots completen la informació que tenen amb un sistema push-pull gossip(combinat).

Al combinar les dues implementacions en una sola, ens donem compte que obtenim un gràfic força més lineal que analitzant les 2 implementacions per separat, això és degut a que el push és més eficient al principi, al aconseguir els primers chunks per als peers, en canvi, el pull és més eficient cap al final, alhora de completar els peers amb els últims chunks restants.
Aquest fet es degut a que en el pull nosaltres hem implementat que cada peer li demani als altres peers que obté amb el getPeers si té un dels chunks que li falten, per aquesta raó al principi tan sols aconseguiran algun chunk d'informació els peers que per casualitat li demanin al seed, però al combinar el push i el pull, aconseguim que els peers obtinguin chunks més ràpid al principi i per tant, un cop els peers ja tinguin uns quants chunks, els altres peers podran demanar els chunks que els fan falta d'aquests peers.
Com a conclusió, s'aconsegueix un millor rendiment ja que combinem els avantatges de cada implementació, una sent millor al principi i l'altra és millor durant l'esprint final.

Codi

Tracker:

```
from datetime import datetime
from sets import Set
from pyactor.context import interval
from pyactor.context import set_context, create_host, sleep, shutdown
from peer import *
from random import sample

class Tracker(object):
    _tell = ['announce', 'comprovarPeers', 'init_start', 'infopeer', 'afegir_host']
    _ask = ['getPeers', 'getMidaTorrent']
    _ref = ['announce', 'getPeers', 'getTorrent', 'afegir_host']

    def __init__(self):
            self.tDictionary = {}

    def init_start(self):
            self.interval_check = interval(self.host, 1, self.proxy, 'comprovarPeers')

    def getMidaTorrent(self, idTorrent):
            mida = 0
            fitxer = open("fitxer.txt", "r")    #Aqui s'obriria el fitxer del torrent
            line=fitxer.readline()
            fitxer.close()
            mida = (len(line)-1)
            return mida

    def announce(self, idTorrent, refPeer):
            try:
                    # printself.tracker.getPeers(idTorrent) "announce "+refPeer.getId()
                    self.tDictionary[idTorrent][refPeer] = datetime.now()
            except:
                    self.tDictionary[idTorrent] = {refPeer:datetime.now()}

    def getPeers(self, idTorrent):
            try:
                    keys = self.tDictionary[idTorrent].keys()
                    return sample(keys, 3)
            except Exception as e:
                            return []

    def comprovarPeers(self):
            tactual = datetime.now()
            for key, peerList in self.tDictionary.items():
                    aux = {}
                    for peer, t_peer in peerList.items():
                            result=tactual-t_peer
```

```
                                if result.total_seconds() <= 10:
                                        aux[peer] = t_peer
                                else:
                                        print "eliminar "+peer

                        self.tDictionary[key]=aux

    def infopeer(self):
            print "//////////////////////////////////"
            for x in self.tDictionary:
                    print(x)
                    for y in self.tDictionary[x]:
                            #refPeer= self.host.lookup(y)
                            print " -"+y.getId()+": "
                            print y.getTorrent()
                    print "//////////////////////////////////"

    def afegir_host(self, host):
            self.host = host

if __name__ == "__main__":
    set_context()
    h = create_host()
    p1 = h.spawn('peer1', Peer)
    p2 = h.spawn('peer2', Peer)
    p3 = h.spawn('peer3', Peer)
    p4 = h.spawn('peer4', Peer)
    p5 = h.spawn('peer5', Peer)
    seed = h.spawn('seed', Peer)
    t = h.spawn('tracker', Tracker)
    t.init_start()

    p1.afegir_host(h)
    p2.afegir_host(h)
    p3.afegir_host(h)
    p4.afegir_host(h)
    p5.afegir_host(h)
    seed.afegir_host(h)
    t.afegir_host(h)

    p1.afegir_tracker(t)
    p2.afegir_tracker(t)
    p3.afegir_tracker(t)
    p4.afegir_tracker(t)
    p5.afegir_tracker(t)
    seed.afegir_tracker(t)

    seed.make_seed()
    sleep(2)

    p1.init_combinado("GOT")
    p2.init_combinado("GOT")
    p3.init_combinado("GOT")
    p4.init_combinado("GOT")
```

```
    p5.init_combinado("GOT")

    seed.init_combinado('GOT')

    for x in range(100):
            sleep(1)
            t.infopeer()

sleep(2)
shutdown()
```

Peer:

```
from pyactor.context import interval, later
from random import choice
from random import randint

class Peer(object):
    _tell = ['notify', 'afegir_tracker', 'afegir_host', 'init_start', 'stop_interval', 'pushing', 'init_push', 'push', 'make_seed',
'init_pull', 'pulling', 'init_combinado', 'combinado']
    _ask = ['getId', 'getTorrent', 'pull']
    _ref = ['afegir_tracker', 'push', 'afegir_host', 'pull' ]

    def __init__(self):
            self.mida_torrent = 0
            self.data = {}
            self.falten = []

    def init_start(self, idTorrent):
            self.interval = interval(self.host, 3, self.proxy, 'notify', idTorrent)

    def stop_interval(self):
            self.interval.set()

    def make_seed(self):
            pos = 0
            fitxer = open("fitxer.txt", "r")
            line=fitxer.readline()
            for pos in (range(len(line)-1)):
                    cont=line[pos]
                    self.data[pos] = cont
            fitxer.close()


    def init_push(self, idTorrent):
            self.interval = interval(self.host, 3, self.proxy, 'notify', idTorrent)
            self.interval_push = interval(self.host, 1, self.proxy, 'pushing', idTorrent)


    def init_pull(self, idTorrent):
            self.mida_torrent = self.tracker.getMidaTorrent(idTorrent)
            for i in range (0, self.mida_torrent):
```

```
                self.falten.append(i)

        self.interval = interval(self.host, 3, self.proxy, 'notify', idTorrent)
        self.interval_pull = interval(self.host, 1, self.proxy, 'pulling', idTorrent)

def init_combinado(self, idTorrent):
        self.mida_torrent = self.tracker.getMidaTorrent(idTorrent)
        for i in range (0, self.mida_torrent):
                self.falten.append(i)

        self.interval = interval(self.host, 3, self.proxy, 'notify', idTorrent)
        self.interval_combinado = interval(self.host, 1, self.proxy, 'combinado', idTorrent)


def getId(self):
        return self.id

def notify(self, nomFitxer):
        self.tracker.announce(nomFitxer, self)

def afegir_tracker(self, tracker):
        self.tracker = tracker

def afegir_host(self, host):
        self.host = host

def getTorrent(self):
        return len(self.data.keys())

def push(self, chunk_id, chunk_data):
        #print "received", chunk_id, chunk_data
        #print self.data[chunk_id]
        if chunk_id not in self.data.keys():
                #print chunk_data
                self.data[chunk_id] = chunk_data

def pushing(self, idTorrent):
        # print self.tracker.getPeers(idTorrent)
        for peer in self.tracker.getPeers(idTorrent):
                # print "push ", peer
                try:
                        chunk=choice(self.data.items())
                except:
                        #print "no tinc res per tant no puc enviar"
                        continue
                peer.push(chunk[0], chunk[1])

def pull(self, chunk_id):
        return self.data[chunk_id]


def pulling(self, idTorrent):
        if (len(self.data.keys()) < self.mida_torrent):
                for peer in self.tracker.getPeers(idTorrent):
                        chunk_id = choice(self.falten)
```

```
                    try:
                        self.data[chunk_id] = peer.pull(chunk_id)
                        self.falten.pop(chunk_id)
                        break
                    except:
                        break


def combinado(self, idTorrent):

    for peer in self.tracker.getPeers(idTorrent):
        try:
            chunk=choice(self.data.items())
            peer.push(chunk[0], chunk[1])
        except:
            pass
        if (len(self.data.keys()) < self.mida_torrent):
            chunk_id = choice(self.falten)
            try:
                self.data[chunk_id] = peer.pull(chunk_id)
                self.falten.pop(chunk_id)
                break
            except:
                break
```