Факултет техничких наука

Универзитет у Новом Саду

Паралелни и дистрибуирани алгоритми и структуре података

Bittorent

Аутор:
Александар Стојановић

Индекс:
Е2 119/2023

1. децембар 2023.

Садржај

1 Увод

Стандардан приступ преузимању података са интернета заснива се на клијент сервер архитектури, где произвољан број клијената преузима податке који се налазе на удаљеном серверу. Са повећањем величине података који се преузимају, као и броја клијената који покушавају да му приступе, овакав начин преноса података постаје неефикасан. Прво ограничење је ограничен пропусни опсег сервера, а друго је чињеница да ће брзина преузимања клијента, упркос његовој максималној брзини преузимања, бити највише онолика колика је брзина слања сервера. Такође, сервер код оваквог приступа представља "критичну тачку" (*single point of failure*). Сви ови проблеми донекле се могу решити додавањем нових сервера, али тиме се стварно решавање проблема само одлаже и повећавју финансијски захтеви система. *Bittorent* протокол својим *peer-to-peer* мрежним уређењем решава све горе наведене проблеме, тако што сваки учесник у мрежи може имати улогу потраживача података, а ако поседује барем и најмањи део траженог податка може имати и улогу понуђача делова података које поседује. Будући да сада постоји више извора за преузимање података (уклоњен је проблем критичне тачке), а чворови мреже који преузимају податке су способни да то раде са више извора истовремено, у могућности су да искористе своју пуну моћ преузимања, а оптерећење пропусног опсега је распоређено на више чворова понуђача. У даљим поглављима биће детаљније објашњена архитекрура *Bittorent* мреже и начин на који се постиже овако ефикасна комуникација и размена података између чворова.

2 Основне компоненте и учесници *Bittorent* мреже

У овом поглављу биће објашњени основни термини, концепти и учесници *Bittorent* мреже како би се у даљим поглављима могла објаснити њихова међусобна комуникација и сам начин функционисања читаве мреже. Из овог разлога читалац не би требао да се обесхрабри ако у потпуности не схвати појмове представљене у овом поглављу, већ само да пређе на наредно.

2.1 Делови (*pieces*) и блокови (*blocks*)

Да би се олакшао и омогућио дистрибуирани пренос података, сваки податак дели се на одређен број делова (***pieces***) фиксне дужине (од 32kB до 16MB). Даље, сваки део додатно се дели у блокове (***blocks***) који представљају основну јединицу трансфера података у мрежи. Може се рећи да један члан мреже поседује неки део када поседује све блокове који га сачињавају и само тада он може делити тај део података са другим члановима мреже.

2.2 Учесници мреже и њихова класификација

У зависности од тога да ли чланови мреже само шаљу или преузимају податке можемо их поделити на ***seeder***-е и ***leecher***-е. *Seeder*-и су чланови мреже који поседују комплетан скуп података и њихова улога је да само шаљу податке осталим члановима мреже. *Leecher*-и са друге стране покушавају да добаве недостајуће делове података не би ли комплетирали свој скуп података. Битно је нагласити да иако leecher-и не поседују комплетан скуп података они су такође способни да осталим члановима шаљу делове података које они поседују, не би ли се убрзао пренос података, као и скинуло оптерећење са самих *seeder*-а. У тренутку када *leecher* сакупи све делове скупа података он аутоматски постаје *seeder*. Алгоритам којим одређени чворови бирају од кога ће преузимати и коме ће слати податке, као и животни циклус једног члана мреже биће детаљније објашњени у даљим поглављима.

Сви чланови мреже (***peers***) морају бити свесни постојања осталих чланова, те због тога сваки члан садржи свој *peer set* тј. листу чланова од којих може да преузима и шаље делове података. Додатно, сваки члан има свој *active per set* тј. подскуп листе чланова мреже од којих тренутно може да преузима и шаље податке.

2.3 *Tracker*

Раније је поменуто како сваки члан мреже носи у себи информацију који су остали чланови са којима може да комуницира и које делове податка они садрже, али поставља се питање како чланови долазе до тих информација? Да би члан мреже дошао

до тих информација он се мора обратити *tracker*-у. *Tracker* је компонента (најчешће веб сервер) која у себи садржи информације о активним члановима мреже(*seeder, leecher*). Када нови члан приступи мрежи он се обраћа *tracker*-у да би био регистрован у њој. Такође, чланови мреже се у току свог рада обраћају *tracker*-у да би га обавестили да су још увек активни како би он ту информацију освежио у својим регистрима и учинио је доступним осталим члановима мреже.

Иако *tracker* делује као критична тачка и компонента која нарушава сам концепт дистрибуираног система, битно је приметити да он у себи не садржи конкретне податке које треба дистрибуирати мрежом већ само информације које омогућавају иницијално међусобно проналажење чланова мреже након чега чланови своју комуникацију настављају у *peer-to-peer* режиму и ни на какав начин не зависе од *tracker*-а. Из овог разлога, допустиво је да *tracker* не буде доступан неко време, јер би то само значило да у том периоду не би било могуће додавање нових чланова, али не би дошло до било каквог губитка података и тренутни трансфери, будући да се обављају директно између чланова, а не посредством *tracker*-а, не би били прекинути.

2.4 *Torrent* фајл

Ако желимо да поделимо неки податак или скуп података путем *Bittorrent* мреже, неопходно је да направимо њему одговарајућ *torrent* фајл који садржи метаподатке који су неопходни за његову дистрибуцију. Метаподаци које садржи су:

- ***Announce*** - *URL tracker*-а који се брине о тренутним члановима мреже
- *Created by* - назив и верзија програма којим је креиран *torrent* фајл
- *Creation date* - датум и време креаирања *torrent* фајла
- *Encoding* - *encoding* који је коришћен унутар *torrent* фајла
- *Comment* - произвољан коментар креатора, може садржати опис података за које је креиран *torrent* фајл
- ***Info*** - конкретне информације о податаку који се преноси

Info секција у себи садржи:

- назив податка
- дужину податка у бајтовима
- *md5 checksum* податка

- дужину дела податка (*piece length*)
- листу хеш вредности свих делова података у правилном редоследу. Хеш се добија рачунањем хеш функције чији је улаз читав садржај једног дела податка

У случају када се преноси скуп података, структура *torrent* фајла већински остаје непромењена, једина разлика је у томе што се наводи назив коренског (*root*) директоријума и уместо информација за један податак имамо листу информација свих података којима су придружене њихове локације у односу на коренски директоријум.

Ова структура *torrent* фајла која садржи само метаподатке о подацима који треба да се пренесу дају му улогу лако преносивог "упутства" одакле да члан мреже преузме податке, од којих тачно делова се фајл састоји и како те делове да искомбинује да би добио конкретан податак. Хеширање делова података уклања потребу за смештањем комплетних делова података у *torrent* фајл, а опет даје довољно семантике потребне за састављање читавог податка од више делова које је члан преузео од осталих чланова мреже. Такође, хеширањем преузетог дела и упоређивањем са хешом тог дела из *torrent* фајла можемо проверити интегритет преузетог дела.

Torrent фајл се ради лакше и брже преносивости путем интернета серијализује у специјализованом *bencoding* бинарном формату.

3 Поглед на фунцкионисање *Bittorent* мреже кроз конкретан случај коришћења

Проблем је следећи, асистент за време рачунарских вежби, да би омогућио интерактиван рад са студентима, мора да подели податке величине неколико гигабајта на све остале рачунаре (отприлике 30 рачунара) у рачунарској учионици. Асистент, будући да није упућен у *Bittorent* технологију, овом проблему приступа тако што податке поставља у дељени директоријум који могу да виде сви остали рачунари у учионици и на тај начин омогућава студентима да их преузму на своје рачунаре. Лако се може уочити да овај приступ наличи класичној клијент сервер архитектури и да након што 30 студената у исто време покуша да преузме податке, мрежа, иако је локално повезани етернет, убрзо постаје загушена, а асистентов рачунар (сервер) преоптерећен. У даљем тексту биће описано корак по корак како би се овај проблем решио коришћењем *Bittorent*-а.

Тренутни чвор који садржи податке (асистентов рачунар) помоћу алата попут *qBitTorrent*-а [2] или *uTorrent*-а [4] креира *torrent* фајл. За време овог корака податак се дели на делове једнаке величине (сем последњег који може бити мањи), делови се хешују и резултати хеш функција сваког од делова у правилном редоследу уписују се у листу хешева унутар *torrent* фајла. Такође, уписује се и дужина појединачног дела, назив податка и све остале информације које су наведене у поглављу 2.4. Поред хешева делова података, најбитнија ствар која се уписује је адреса *tracker*-а коме се чланови могу обратити да би сазнали одакле могу скинути дељени податак. На ономе ко успоставља мрежу је одлука да ли ће имплементирати свој *tracker* сервер или ће навести линк неког од јавно доступних *tracker*-а [3]. Након креирања *info* секције 2.4 она се комплетна хешира и резултат (*infohash*) се користи да јединствено идентификује *torrent* фајл. Креирани *torrent* фајл асистент поставља на место доступно свима попут дељеног директоријума или неког јавног веб сервера, како би га остали студенти преузели на своје рачунаре. Будући да је он веома мале величине, то не представља проблем по пропусни опсег мреже.

Сада сви студенти на својим рачунарима, укључујући и асистента на свом , помоћу неког од *Bittorent* клијената попут *qBitTorrent*-а [2] или *uTorrent*-а [4] отварају *torrent* фајл и на тај начин се региструју код *tracker*-а који се налази на линку наведеном у фајлу. Студенти, будући да на својим рачунарима немају дељени податак, бивају регистровани као *leecher*-и од стране *tracker*-а, док асистентов рачунар, пошто садржи читав дељени податак, аутоматски бива регистрован као *seeder*.

Након регистровања код *tracker*-а, сваки члан мреже захтева од њега одређени *peer set*, тј. листу *seeder*-а и *leecher*-а са којима надаље могу да размењују податке.

Максималан број конекција које може да успостави један члан у мрежи је 80, док је стандардан број 50. Увек се гледа да се направи баланс између броја *seeder*-а и *leecher*-а у *peer set*-у који се шаље члану мреже, тако да дистрибуција података буде оптимална. На овај начин иако на пример можемо имати и преко хиљаду чланова неке *Bittorent* мреже, сваки члан ће бити свестан само њеног подскупа, како не би дошло до преоптерећења клијента. Члан мреже се уобичајено на сваких 30 минута обраћа *trackeru*-у како би освежио свој *peer set* или уколико број чланова његовог *peer set*-а падне испод 20.

Тренутно, чланови мреже су свесни постојања других чланова, али не знају код којих чланова се налазе подаци који су им потребни да би комплетирали њихов скуп података. Такве информације чланови стичу путем *gossip* алгоритма [1], који дефинише како чланови директно међусобно комуницирају да би информисали једни друге о поседовању одређених делова података.

Након испуњених свих претходник корака чланови мреже могу започети са разменом података. Конкретни алгоритми одабира делова података и чланова од којих ће ти делови бити преузимани биће описани у наредна два поглавља. Како који члан преузме неки део податка у целости, он постаје способан да дели тај део са остатком чланова мреже. На тај начин у нашем примеру оптерећење на асистентновом рачунару постојаће само на самом почетку док сваки од студентских рачунара не преузме један део податка са асистентовог рачунара, а затим рачунари могу почети међусобно да размењују податке који им недостају и на тај начин растерете асистентов рачунар. Тиме што сваки рачунар сада садржи неке од делова податка смањује се ризик губљења података у случају отказа асистентовог рачунара, јер би се читав скуп података могао реконструисати комбинујући делове података свих чланова мреже.

4 Алгоритми за одабир делова података

У зависности од стања мреже, односно дистрибуираности делова дељених података по члановима и количине делова података коју је конкретан члан преузео чланови бирају различите алгоритме одабира делова података чије ће преузимање захтевати од других чланова.

4.1 *Rarest first* селекциони алгоритам

Као што се може закључити по самом називу, овај алгоритам приоретизује преузимање делова података који су најређи у читавој мрежи. Овај приступ има неколико бенефита:

- Редукује се опасност од губитка делова података у случају отказа неког од чланова који их једини садржи.
- Смањује се оптерећење на *seeder*-у, због тога што ће му се чланови мреже обраћати само у случају преузимања дела који се нигде другде не налази у мрежи.
- Повећава се брзина преузимања будући да се претходно ретки делови, сада налазе код више чланова па је могуће паралелно преузимање блокова тих делова.
- Омогућава преузимање података. Више о овоме биће речено у наредном поглављу, али суштина је у томе да ако члан садржи веома редак део, сви остали чланови ће хтети да комуницирају са тим чланом да би га преузели од њега и тиме му дозвољавају да и он од њих преузима делове који њему недостају (*unchoking*).

Да се закључити да се овај алгоритам најчешће користи у случају када је мрежа недавно успостављена и делови података још увек нису стигли да се равномерно дистрибуирају по мрежи или се нови извор података тек појавио у мрежи.

4.1.1 Како чланови знају који је део најређи?

Постоје два приступа. Код првог сваки пут када неки члан преузме читав део он свим осталим члановима свог *peer set*-а одашиље *have* поруку којом их обавештава да он сада поседује тај део. Други начин је да сваки пут када два члана остваре конекцију (а након првог пута у одређеним временским интервалима) они размене *bitfield* поруке које су ништа друго него низови нула и јединица који имају елемената онолико колико укупно постоји делова података и ако члан садржи тај податак, на месту тог дела у низу стоји јединица, а ако га не садржи нула. Помоћу ова два приступа сваки чвор је свестан које делове који чланови поседују и из тога лако може да израчуна који делови су најређи у мрежи.

4.2 *Random first* селекциони алгоритам

Овим алгоритмом се најчешће служе чланови који су тек приступили мрежи (чланови који поседују мање од 4 дела) и којима је под хитно потребан било какав део података да би могли почети да их деле са другима, како би остварили право на преузимње делова података од других чланова (због чега је ово битно биће објашњено у наредном поглављу). Из тог разлога овај алгоритам функционише тако што члан захтева од осталих чланова насумичне делове података, не би ли што пре добио било какав део.

4.3 *Strict priority* полиса

Поменуто је да је основна јединица трансфера делова података блок и да се један део дели на мноштво блокова. Због овога могуће је да један члан истовремено преузима више блокова, где сваки припада различитом делу података. Ово, иако делује као убрзање, успорава трансфер података у мрежи због тога што члан не може да почне да дели део података све док не преузме читав део, те се због тога приоретизује истовремено преузимање више блокова који сви припадају истом делу.

4.4 *Endgame mode*

Када члан мреже дође у фазу где му фали само неколико делова како би комплетирао читав скуп података, преузимање тих делова може бити јако споро или из разлога што су делови јако ретки у мрежи или зато што је већина чланова заузета. Како би се ова фаза убрзала члан уместо да шаље захтев за неки део само једном члану он тај захтев *broadcastuje* свим члановима мреже у нади да ће му бар један од њих одгворити. Чим добије одговор од једног члана мреже, осталим члановима *broadcastuje* поруку да му тај део више није потребан како би смањио оптерећење на мрежи. Иако овакав начин прибављања података знатно оптерећује мрежу, он се толерише у завршној фази преузимања скупа података због тога што се рачуна да ће ова фаза трајати веома кратко.

5 *Choke* алгоритам

Због природе *Bittorent* мреже, не постоји никакав централни ентитет који би координисао комуникацију чланова мреже, оптимизовао је и кажњавао непожељно понашање. Сваком члану, ако би се понашао максимално себично, у интересу је да што пре преузме све делове скупа података и напусти мрежу и нема никакву мотивацију да те податке подели са другим члановима. Када би овакво понашање било дозвољено, читав концепт *Bittorent-a* био би обесмишљен и мрежа једноставно не би била у могућности да дистрибуира податке. *Choke* алгоритам покушава да унутар дистрибуираног система наметне правила таква да се постигне баланс између дељења и преузимања података, тако да сви чланови имају могућност да преузму све податке, а да уједно за време тог процеса помогну и осталим члановима да такође дођу до тих података.

5.1 Основни појмови

Слепим путником (***free rider***) назива се члан мреже који само преузима делове података, али не дели никакве податке заузврат. Алгоритам покушава да пенализује овакво понашање.

Дављење (***choking***) је привремено одбијање дељења података неком чвору. Кажемо да члан А дави члана Б ако у том тренутку А брани Б да преузме податке од њега. Дављење превентује преоптерећење мреже услед превеликог броја упућених захтева и злоупотребу мреже о чему ће више речи бити касније.

Када се каже да је чвор А **заинтересован** за чвор Б уколико А жели да преузме податке од чвора Б.

На самом почетку, свако свакога дави. Питање је како члан бира кога да престане да дави? Читав алгоритам заснива се на реципроцитету тј. узвраћању услуге. Размотримо пример где члан Б дави члана А што би значило да члан А не може да преузме податке од Б. Ако је Б заинтересован за А и А дозволи Б да преузме његове податке заузврат ће Б престати да дави А и тиме дозволити А да преузме његове податке. Ово би значило да је дељење података пожељно понашање у мрежи и да ће чланови који деле податке бити награђени могућношћу да преузимају податке.

Члан бира које ће чланове престати да дави тако што тражи чланове који имају велику брзину дељења тј. чланови ће престати да дају чланове са великом брзином дељења да би они што пре преузели податке. Тиме ће такође дозволити преузимање њихових података. На овај начин чланови се мотивишу да деле своје податке, јер заузврат добијају велику брзину преузимања од других чланова.

5.2 *Choke* алгоритам *leecher*-а

Члан извршава овај алгоритам:

- На сваких 10 секунди.
- Сваки пут када неки члан напусти или уђе у *peer set*.
- Сваки пут када други члан постане заинтересован за њега.

Како алгоритам функционише? Сваки пут када се покрене, сортира све заинтересоване чланове по брзини преузимања коју нуде конкретном члану (јер може да се деси да неки члан генерално нуди велику брзину преузимања, али се географски налази далеко од другог члана и због тога брзина пада) и престаје да дави (*unchoke*) 3 члана са највећом брзином. Такође, у обзир долазе само они чланови који су поделили бар један блок у последњих 30 секунди. На овај начин се узимају у обзир само активни чланови. Овај део алгоритма назива се *regular unchoke*. Поред ове методе, на сваких 30 секунди члан престаје да дави насумичног члана (*optimistic unchoke*). На овај начин дајемо шансу новим члановима који су тек приступили мрежи који можда имају велику брзину дељења, али немају делове података које би могли да поделе и овако они могу да добаве свој први део.Уколико је *unchoke*-овани члан незаинтересован он и даље остаје *unchoke*-ован, али члан *unchoke*-ује новог насумичног члана све док не наиђе на заинтересованог члана. На крају члан ће завршити са произвољним бројем *unchoke*-ованих чланова, али са максимално 4 заинтересована *unchoke*-ована члана. Овакав алгоритам максимално пенализује слепе путнике, због тога што они никада не деле податке и тиме ће се увек налазити на крају листе сортиране по брзини дељења. Једина шанса да слепи путник буде *unchoke*-ован је путем *optimistic unchoke*-а.

Још један од механизама који се користе је ***anti-snubbing*** механизам. Наиме, када је неки члан дављен од стране свих чланова, он тада такође почне да дави све остале чланове. Ако овај механизам примени још неколико чланова драстично ће се смањити број чланова који би могли да упадну у избор *regular unchoke*-а и тада се повећава број *optimistic unchoke*-ова, а тиме и шанса да члан који је био дављен од стране свих буде *unchoke*-ован.

5.3 *Choke* алгоритам *seeder*-а

Треба приметити да *seeder*, будући да поседује све податке, нема никакву мотивацију да остане у мрежи и дели податке са другима. Разлози због чега чланови остају у мрежи након што преузму све податке и тиме постану *seeder* су чисто алтруистичке природе. Упркос њиховим добрим намерама, они не могу делити податке

баш свим члановима мреже јер би тада постали уско грло система, те и они морају примењивати неки вид *choke* алгоритма.
Члан извршава овај алгоритам:

- На сваких 10 секунди.
- Сваки пут када неки члан напусти или уђе у *peer set*.
- Сваки пут када други члан постане заинтересован за њега.

Како алгоритам функционише? Сви чланови се сортирају према томе када су последњи пут *unchoke*-овани, с тиме да се даје приоритет члановима са већом брзином дељења. Иако *seeder* нема никакве користи од велике брзине дељења, она се бира као критеријум уместо брзине преузимања, јер би у супротном слепи путници са великом брзином преузимања увек били одабирани и саботирали мрежу. Такође, слепи путници будући да ништа не деле неће имати користи ни од сортирања чланова према времену последњег *unchoke*-а, јер они бивају јако ретко *unchoke*-овани само у случају *optimistic unchoke*-а. Чланови који никада нису били *unchoke*-овани сортирају се само према брзини дељења. Затим *seeder* 20 секунди *unchoke*-ује прва 3 члана у сортираној листи и једног насумичног члана, а наредних 10 секунди *unchoke*-ује прва 4 члана с тиме да тада *choke*-ује претходно насумично изабраног члана. Дељењем података са насумичним чланом на 20 секунди, са паузама од 10 секунди помаже члановима на лошијим позицијама у листи да преузму неке делове података и себи повећају шансу да буду *unchoke*-овани од стране других чланова и тиме узнапредују на листи.

6 Закључак

У овом семинарском раду представљена је мотивација за креирање дистрибуиране *peer-to-peer* мреже попут *Bittorent*-а и које су њене предности у односу на стандардне клијент сервер архитектуре за преузимање података преко интернета. Читалац је упознат са актерима система и животним циклусима мреже као и детаљима алгоритама који омогућавају ефикасну и отпорну на отказ дистрибуцију података која подстиче удружени рад чланова ка заједничком циљу преузимања података и њиховој даљој дистрибуцији кроз мрежу. Даље истраживање може бити усмерено ка конкретним применама оваквих мрежа или мерама заштите од слепих путника који се користе много лукавијим начинима за експлоатисање мреже.

Библиографија

[1] Gossip algorithms. https://web.stanford.edu/~boyd/papers/gossip_infocom.html, 2023. Accessed: November 27, 2023.

[2] qbittorrent. https://www.qbittorrent.org/, 2023. Accessed: November 27, 2023.

[3] Tracker list. https://www.torrenttrackerlist.com/torrent-tracker-list/, 2023. Accessed: November 27, 2023.

[4] utorrent. https://www.utorrent.com/, 2023. Accessed: November 27, 2023.