Sinhronizacija map z rsync

Seminarska naloga

Miha Kovač

Januar 2022

1 Opredelitev problema

Datoteke najprej ustvarimo na enem računalniku, v veliki večini primerov je dovolj da ostanejo na datotečnem sistemu tega računalnika, saj jih potrebujemo le lokalno. Nekatere datoteke pa le napišemo lokalno ter jih potrebujemo na nekem drugem računalniku, pogosto strežniku, zato potrebujemo način, da jih prekopiramo iz lokalnega računalnika na oddaljeno lokacijo ali v obratni smeri.

Za dani problem odlično deluje program scp (secure copy). Program scp prekopira lokalno datoteko na oddaljen sistem **v celoti.**

Denimo zdaj, da na lokaciji na katero kopiramo že obstaja starejša verzija datoteke. V tem primeru bi bilo smotrno prenesti le tiste dele datoteke, ki so se spremenili. Pri tem je potrebno program zasnovati tako, da stroški omrežne komunikacije, o tem, katerih podatkov ni potrebno prenesti, ne presežejo velikosti datoteke same. Namesto izraza kopiranje lahko v tem kontekstu uporabljamo termin sinhronizacija datoteke.

V ta namen je Andrew Tridgell v devetdesetih letih razvil program rsync, ki ga bomo podrobneje spoznali v nadaljevanju. Pogledali si bomo tudi implementacijo tega algoritma v pythonu.

Dodati je potrebno, da se programa scp in rsync ukvarajta samo s kopiranjem oz. sinhronizacijo ne pa tudi z varnim prenosom. V ta namen uporabljata tunel SSH.

2 Algoritem rsync

Da lahko algoritem izkoristi obstoječo verzijo datoteke, operira na blokih. Blok je del datoteke, torej zaporedje bytov vnaprej določene dolžine. Blokov, ki so pri prejemniku že prisotni ne pošiljamo ponovno. Da nam ni potrebno prenašati celotnega bloka, potrebujemo za razpoznavo enakih blokov njihov podpis (angl. checksum, signature). Izkaže se da je dobra ideja uporabljati dva podpisa. Prvi je enostaven in lahko izračunljiv, z njim iščemo potencialne kandidate. Drugi pa je počasnejši, ampak zanesljivo potrdi ujemanje.

Naj bosta A in B računalnika, ki imata vsak svojo verzijo datoteke D, D_a in D_b. Cilj algoritma je popraviti vsebino datoteko D_b, da bo enaka D_a (D_b želimo sinhronizirati z D_a).
Dalje naj bo:
- a_i: i-ti byte datoteke D_a
- b_i: j-ti byte datoteke D_b
- R: preslikava, ki bloku priredi hitri podpis
- H: preslikava, ki bloku priredi temeljiti podpis
- L: velikost bloka (v bytih)

ALGORITEM (povzeto po [1] str. 52):

1) B razdeli datoteko D_b na bloke B_j dolžine L.
   Za vsak blok B_j izračuna podpisa $R_j = R(B_j)$ in $H_j = H(B_j)$.
   Pare (R_j, H_j) pošlje računalniku A.
2) Naj bo A_i blok datoteke D_a dolžine L pri odmiku i (torej $A_i = [a_i, a_{i+1}, ..., a_{i+L-1}]$).
   A za vsak odmik i izračuna podpis $R_i = R(B_i)$.
3) A primerja R_i z vsemi R_j prejetimi od B.
4) Za vsak indeks j za katerega se R_i ujema z R_j,
   A izračuna še $H_i = H(A_i)$ in ga primerja s H_j
5) Če se ujemata tudi H_i in H_j,
   A pošlje sporočio B o ujemanju blokov A_i in B_j ter odmik i.
   Sicer A pošlje B byte a_i.
6) B prejme sporočila in bloke ter iz njih konstruira vsebino nove datoteke D_b

3 Tehnične podrobnosti

3.1 Temeljiti podpis

Temeljiti podpis dveh enakih blokov mora biti enak, dveh različnih pa različen. Če smo pripravljeni sprejeti metodo, ki z zelo majhno verjetnostjo dva različna bloka preslika v enaka podpisa, so zgoščevalne funkcije (angl. hash function) to kar iščemo. Rsync je najprej za zgoščevalno funkcijo uporabljal MD4, ki je bil kasneje zamenjan za MD5. Podpis, ki ga generirata MD5 in MD4 je 128 biten (16 B).

Ker bomo temeljiti podpis računali le bloke, ki bodo imeli enak hitri podpis, nas daljši čas računanja ne moti.

3.2 Hitri podpis

Hitri podpis moramo računati na vsakem odmiku, torej približno tolikokrat kot je dolžina datoteke Da v bytih, zato si časovno lahko privoščimo le manj zahtevne postopke.

Izbran je bil algoritem, ki ga lahko računamo iterativno (angl. rolling checksum). Če imamo podan podpis pri odmiku i ter prvi byte v bloku in prvi byte po bloku, lahko s preprostimi operacijami izračunamo podpis pri odmiku i+1.

Algoritem potrebuje še parameter M (za kongruenco po modulu M), ki vpliva na velikost podpisa. Če je M potenca števila 2, npr. $M = 2^m$, je množenje z M shift za m bitov v levo in je kongruenca po modulu M ekvivalentna bitni operaciji and z masko ki ima prižganih zadnjih m bitov. Bitne operacije se izvedejo zelo hitro, posledično je tudi izračun podpisa hiter. Rsync uporablja $M = 65536 = 2^{16}$, posledično je podpis 32 biten (4 B).

Podpis celotnega bloka izračunamo v dveh delih, najprej desnih 16 bitov (r_1), potem levih 16 bitov (r_2), nato pa oba dela še združimo v en 32 bitni podpis (R).

Uporabimo naslednje formule, označimo: L dolžina bloka; a_i i-ti byte datoteke:

(1a) $r_1(k,L)=(\sum_{i=0}^{L-1} a_{i+k}) \bmod M$

(2a) $r_2(k,L)=(\sum_{i=0}^{L-1} (L-i)\cdot a_{i+k}) \bmod M$

(3a) $R(k,L)=r_1(k,L)+M\cdot r_2(k,L)$

Ko podpis računamo postopoma, uporabimo naslednje rekurenčne zveze:

(1b) $r_1(k+1,L)=(r_1(k,L)-a_k+a_{k+L}) \bmod M$

(2b) $r_2(k+1,L)=(r_2(k,L)-L\cdot a_k+r_1(k+1,L)) \bmod M$

(3b) $R(k+1,L)=r_1(k+1,L)+M\cdot r_2(k+1,L)$

3.3 Učinkovito hranjenje prejetih podpisov

Pri 3. koraku algoritma mora A primerjati trenutni hitri podpis Ri z vsemi podpisi Rj, ki jih je prejel od B. Potrebujemo torej učinkovit način hranjenja podpisov blokov, bolj natančno, hraniti moramo prejete pare (hitri podpis, MD5). Vedno bomo preko hitrega podpisa dostopali do podpisa MD5. Tak dostop učinkovito implementiramo z uporabo zgoščene table.

Uporabimo odprto zgoščeno tabelo, ki za ključe uporablja 16 bitna cela števila. Če bi uporabljali celoten 32 bitni podpis, bi že prazna tabela zavzela ogromno pomnilnika. Ključ iz podpisa izračunamo kot vsoto obeh 16 bitnih polovic (po modulu 2^{16}).

4 Program iz algoritma

4.1 Mape

Do sedaj smo konstruirali učinkovit algoritem za sinhronizacijo ene datoteke. Če bi želeli sinhronizirati mapo, se sprehodimo po celotnem drevesu podmap in sinhroniziramo vsako datoteko posebej.

4.2 Varen prenos

Od programa pričakujemo tudi varen prenos datotek. Program SSH (angl. secure shell) nam v osnovi omogoča dostop do lupine oddaljenega računalnika, poleg tega pa lahko SSH uporabimo kot tunel, ki določen omrežni promet preusmeri na oddaljeni računalnik ali ga usmeri iz oddaljenega računalnika na lokalni. Tako lahko iz lokalnega računalnika dostopamo do vrat (angl. port) na oddaljenem računalniku in procesov, ki poslušajo na njih.

Ker imamo preko SSH-ja dostop do lupine in možnost zagona procesov, lahko na oddaljenem strežniku zaženemo proces, ki posluša na nekh vratih. Omogočen nam je tudi promet do teh vrat, zato lahko oba procesa komunicirata in pošiljata potrebne podatke.

4.3 Uporaba programa rsync

Splošna oblika ukaza za kopiranje datotek je:

rsync <source> <destination>

kjer destination predstavlja datoteko, ki bi jo radi sinhronizirali s source datoteko. Za sinhronizacijo map dodamo stikalo -r (recursive). Ena izmed lokacij je lahko na oddaljenem računalniku, pri čemer uporabimo notacijo kot pri SSH.
Primer:

rsync -r spo/ miha@192.168.1.32:~/Dokumenti/spo

5 Implementacija algorima rsync v pythonu

V prejšnjih poglavjih opisani algoritem sem poskusil v čim bolj enostavni obliki implementirati v programskem jeziku python. V grobem je projekt razdeljen na strežnik, odjemalec ter podporne razrede.

Strežnik razpolaga z aktualno verzijo datoteke, odjemalec pa ima nepopolno verzijo, ki bi jo rad sinhroniziral s tisto, ki jo ima strežnik. Programa zaganjamo ločeno, komunikacija med njima poteka preko vtičev (angl. sockets). Sporočila, ki imajo prenašajo vsebino so v formatu JSON, vsako je potrebno potrditi z nizom »success« zakodiranem v naboru ascii (ni v formatu JSON). Dokler potrdilo ni prejeto, program ne nadaljuje z izvajanjem. To negativno vpliva na hitrost sinhronizacije, omogoča pa enostavno implementacijo.

Odjemalec določa datoteko, ki se bo sinhronizirala, velikost bloka ter velikost podpisa za hitri podpis. Te parametre nastavi uporabnik ob zagonu z ustreznimi stikali, odjemalec pa jih v prvem sporočilu pošlje strežniku.

Preostanek programske kode predstavlja implementacijo algoritma, po začetnem sporočilu odjemalec pošlje pare podpisov za bloke, ki jih ima. Nato server pošlje sporočila o ujemanjih blokov ali pa posamezne byte, s katerimi odjemalec rekonstruira datoteko.

Zagon:

Premaknemo se v mapo server ter zaženemo strežnik z ukazom

python server.py [-p=<port>]

V ločeni lupini se premaknemo v mapo receiver in zaženemo še odjemalca z

python file_receiver.py <ime-datoteke> [-p=<port>] [-b=<block-size>] [-r=<rolling-checksum-size>]

Opomba: stikala v oglatih oklepajih predstavljajo opcijske parametre.

6 Viri

[1] Efficient Algorithms for Sorting and Synchronization (https://www.samba.org/~tridge/phd_thesis.pdf)
[2] The rsync algorithm (https://rsync.samba.org/tech_report/)
[3] rsync(1) - Linux man page (https://linux.die.net/man/1/rsync)