Duplikuotų failų aptikimo programa

Programa yra skirta linux operacinei sistemai

0.1. Programos veikimas

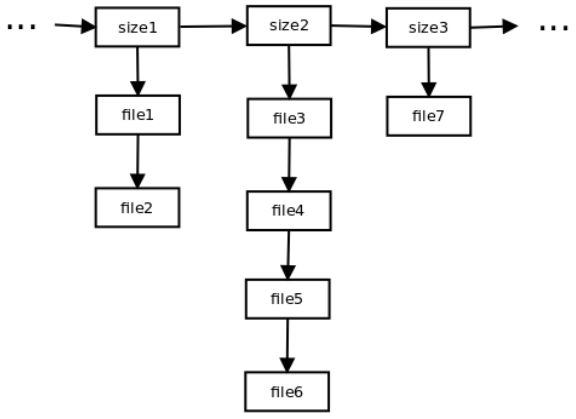
Pačiame abstrakčiausiame lygmenyje programa atlieka tris žingsnius duplikuotų failų radimui

1. Sudaro potencialiai sutampančių failų sąrašą pagal failo dydį
2. Kiekviename potencialiai sutampančių failų saraše, kuriame yra nemažiau kaip 3 failai skaičiuojama murmur3 128bit hash funkcija
3. Kiekviename potencialiai sutampančių failų saraše randama failų poros, kurių suskaičiuota murmur3 hash reikšmė sutampa, tuomet jie palyginami pagal jų turinį baitas po baito

0.1.1. Potencialiai sutampančių failų sarašo sudarymas

Šis sarašas sudaromas pagal failų dydį. Kiekviena gija skanuoja vis kitą aplanką, kiekvieną rastą failą pridėdama į map struktūrą, kur pridėjimo raktas yra failo dydis.

Dėl būtent šios map duomenų struktūros ypatumų (žr.: Duomenų struktūra map) tuo pačiu sudaromas failų linked-list'as sudėliotas pagal failo dydžius:



1 pav. Potencialiai sutampančių failų linked-list'as pagal dydį.

Verta pastebėti, kad map yra įprastas hash kontaineris galintis kiekvienam raktui priskirti tiktai vieną reikšmę, todėl ši reikšmė yra eilė. Gija norėdama įdėti naują failą į map struktūrą pirmiausia patikrina ar jau yra kas nors su atitinkamu raktu, ir jeigu yra - paimama tai, kas yra tuo raktu išsaugota (o tai yra eilė), ir į jos galą įdedamas naujas failas. Jeigu tuo raktu nieko nebuvo rasta, sukuriama nauja eilė, įdedamas pirmasis failas ir ši eilė įdedama į map struktūrą.

0.1.2. Murmur3 hash skaičiavimas

Pilnai sudarius potencialiai sutampančių failų sąrašą vykdomas hash skaičiavimas. Jeigu potencialiai sutampančių failų saraše to pačio dydžio failų yra nemažiau negu 3, hash skaičiavimas leidžia kiekvienam failui suskaičiuoti hash vertę ir ją palyginti tarpusavyje. Jeigu hash vertė sutampa failai gali (bet neprivalo) sutapti, tačiau, jeigu hash reikšmės nesutampa - garantuotai žinome, kad failai nesutampa. Šitaip, jeigu yra nemažiau kaip 3 potencialūs sutapimai, hash reikšmės leidžia kiekvieną failą perskaityti tik po kartą, ir greitai atmesti nesutampančius failus. Kitu atveju kiekvieną failą reiktų skaityti $2(n-1)$ kartų tam kad jis būtų palygintas su visais likusiais kandidatais.

Šioje programoje kiekviena gija skaičiuoja atskiro failo hash reikšmę.

0.1.3. Galutinis palyginimo žingsnis

Suskaičiavus hash reikšmes reikalingiems failams, vykdomas hash ir byte-by-byte palyginimas. Iš pradžių išrenkamos visos poros, kurioms arba hash nebuvo skaičiuotas (tam yra dvi galimybės: failo dydis yra 0 baitų, arba potencialių sutampančių failų yra mažiau negu 3) arba kurių hash reikšmės sutampa, ir tuomet atliekamas šių failų byte-by-byte palyginimas. Verta pastebėti, kad 0B failai be palyginimo iškarto laikomi sutampančiais

Šioje užduotyje vienai gijai tenka vienos failų poros palyginimas.

Rasti sutapimai yra dedami į eilę, iš kurios writer gija ima ir juos rašo į išeities failą.

0.2. Duomenų struktūros

Šioje programoje naudojamos trys pagrindinės duomenų struktūros:

1. map
2. queue
3. thread_pool

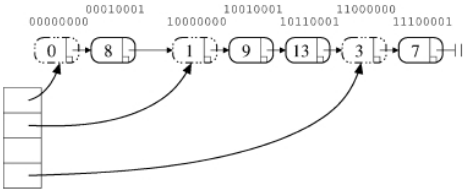
0.2.1. queue

Šiai programai buvo parinkta dvigubo žodžio ilgio Compare-and-Swap atominiais primityvais grįsta eilė. Todėl ji gali veikti daugiagijoje aplinkoje be jokių mutex‘ų ir panašių blokuojančių struktūrų.

Daugiau apie eilę: http://www.cs.rochester.edu/~scott/papers/1996_PODC_queues.pdf

0.2.2. map

Šiai programai buvo parinktas hash kontaineris veikiantis kiek priešingu principu negu įprasta. T.y. visi elementai dedami į vieną linked-list‘ą specialia tvarka, o patys elementai sudaromi nurodant linked-list‘o elementus tame "kibirėlyje"(ang.: bucket).



(a) Buckets 0,1 and 3 are initialized. Bucket 2 is uninitialized

2 pav. map. Paveiksliukas iš http://www.interdb.jp.

Šiuo atveju tai yra labai didelis pliusas, nes hash konteinerio elemento įdėjimo sudėtingumas yra O(1), bet paskui skaičiuojant murmur3 nereikia vykdyti raktų paieškos. Galima tiesiog keliauti per pirminį linked-list'ą.

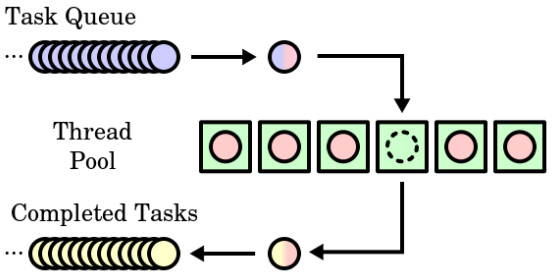
Taip pat šis hash kontaineris ir jį palaikantis linked-list'as veikia to pačio dvigubo žodžio ilgio Compare-and-Swap primityvais, tad jis gali veikti daugiagijoje aplinkoje be jokių mutex'ų ar panašių blokuojančių struktūrų.

Daugiau apie hash konteinerį:
http://www.cs.ucf.edu/~dcm/Teaching/COT4810-Spring2011/Literature/SplitOrderedLists.pdf

0.2.3. thread_pool

Thread_pool jau nevisai *duomenų* struktūra, bet šio aprašymo tikslais thread_pool bus įkeltas prie duomenų struktūrų.

Thread_pool įgyvendina standartinį thread pool pattern'ą. Yra sukuriama eilė, kurioje dedamos užduotys gijoms. Gijos viena po kitos ima užduotis iš eilės ir jas vykdo.



3 pav. Thread_pool. Paveiksliukas iš http://www.wikipedia.com.

Thread_pool naudojama ta pati eilė kaip ir potencialų failų sąrašui sudaryti.

0.3. Išeities kodo struktūra

Išeities kodas skirstomas į tokias failų poras:

- *mpmc_lf_queue.[c,h]* - duomenų struktūrą queue įgyvendinantys failai

- *lf_map.[c,h]* - duomenų struktūrą map įgyvendinantys failai
- *thread_pool.[c,h]* - thread_pool implementacija
- *dir_trav_task.[c,h]* - thread_pool pritaikyta potencialiai sutampančių failų sąrašo sudarymo užduotis
- *calc_hash_task.[c,h]* - thread_pool pritaikyta hash verčių skaičiavimo užduotis
- *compare_task.[c,h]* - thread_pool pritaikyta hash ir turinio palyginimo užduotis
- *free_map_task.[c,h]* - thread_pool pritaikyta resursų atlaisvinimo užduotis
- *writer.[c,h]* - writer gijos implementacija
- *jauler_lsdup.c* - pagrindinis programos failas
- *list_utils.h* - Macro skirti linked list'ams
- *file_desc.h* - programos viduje naudojama struktūra failams apibūdinti

0.4. Vartotojo sąsaja

Ši programa turi minimalią vartotojo sąsają. Paleidus programą, jai galima pateikti komandas stop arba start (tiesiog parašius stop arba start ir paspaudus enter). komanda stop atitinkamai sustabo thread_pool ir writer gijas, o start jas vėl paleidžia.

Programa priima tris komandinės eilutės argumentus. Jie būtinai turi būti pateikti tokia tvarkia:

1. kelias iki direktorijos
2. išeities failo pavadinimas
3. thread_pool gijų skaičius