Správa projektu do kurzu Přenos dat, počítačové sítě a protokoly

Pokorný Fridolín
xpokor32@stud.fit.vutbr.cz

10. marca 2014

1 Úvod

Cieľom projektu bolo vytvoriť efektívny nástroj pre agregáciu a triedenie sieťových tokov. Ako referenčný nástroj pre optimalizovanie bol vybraný voľne dostupný nástroj *nfdump*, ktorý záznamy sieťových tokov agreguje a triedi sekvenčne bez akéhokoľvek paralelizmu.

2 Analýza problému a návrh algoritmu

2.1 Datové štruktúry

Pre optimalizáciu je vhodné vybrať vhodnú datovú štruktúru pre ukladanie jednotlivých dát. Vzhľadom na charakteristiku sieťových tokov, je vhodné pre ukladanie jednotlivých záznamov zvoliť datovú štruktúru, ktorá minimalizuje dobu vyhľadania záznamu podľa kľúča (zdrojový port, cieľový port, zdrojová adresa alebo cieľová adresa). Práve vyhľadávanie záznamu je jednou z najčastejších operácii v programe. V ideálnom prípade je vhodné zvoliť datovú štruktúru s asociatívnym prístupom (časová zložitosť vyhľadania je $\mathcal{O}(1)$). Pre agregáciu podľa portov by bolo teda vhodné vytvoriť 65536 položiek (počet portov), ktoré by uchovávali jednotlivé záznamy. Pre adresy IP je vytvorenie takej štruktúry obtiažnejšie vzhľadom na rozsah IPv4, či dokonva IPv6 adries. Tu by bolo vhodné zvoliť hash tabuľku, kde je však nutné vhodne implementovať hash algoritmus pre maximálnu elimináciu hashovacích kolízií a tvorby tak synonym. V tomto prípade by časová zložitosť mohla degradovať až na $\mathcal{O}(n)$, čo je viac než nežiadúce.

Vzhľadom na vyššie uvedené problémy, ktoré by mohli byť predmetom ďalšieho vývoja, boli ako datové štruktúry pre ukladanie záznamov zvolené stromy pre agregáciu podľa IP adrieas a tabuľku pre agregáciu podľa portov. Pre agregáciu podľa IP adries bol vybraný *red-black* strom, ktorý v testoch dosahoval najlepšie výsledky. Pre samotné radenie bol vybraný binárny storm so spätnými ukazovateľmi.

Strom typu *red-black* bol vybraný na základe testovania rýchlosti programu. Pri výbere boli zohľadňované stromy typu *red-black*, binárny strom so spätnými ukazovateľmi, *AVL* a stromy *Splay*.

Pre zoraďovanie bol vybraný binárny strom so spätnými ukazovateľmi. Tu bol okrem dôrazu na rýchlosť kladený aj dôraz na efektívne vypisovanie a uvolňovanie alokovaných systémových zdrojov. Vzhľadom na fakt, že priechod *inorder* v zoradenom binárnom strome dáva zoradenú postupnosť podľa kľúča, je tak možné v lineárnom čase $\mathcal{O}(n)$ vypísať požadovaný výsledok agregácie, resp. radenia. Tu nie sme limitovaný čakaním na diskové operácie vzhľadom na fakt, že všetky položky sú alokované a pripravené v pamäti RAM.

Implementácia využíva upravené zdrojové kódy z knižnice `libtree` [1], pre ktorú je charakteristické to, že nie je potrebné alokovať pre každú položku samostatný uzol, ale položky už sú uzlami (podobne ako v jadre operačného systému Linux). Ukazateľ na začiatok datovej štruktúry sa dopočína na základe relativnej pozície k štruktúre uzlu (viac viď *bstree.h* a *rbtree.h*). Tým odpadá nutnosť alokácie a tým spojené volania jadra operačného systému pre pridelenie pamäte. Výrazne sa tak zefektívňuje doba výpočtu. V implementovanom algoritme pre každú položku v záznamoch o sieťových tokoch pripadá menej než jedna alokácia vzhľadom na možnosť znovupoužitia alokovanej štruktúry v prípade existencie záznamu v strome (pakety a bajty sa len sčítajú).

2.2 Algoritmus

Po spracovaní parametrov sú otvorené všetky vstupné súbory pre prípadnú detekciu chyby a prípravu na paralelný beh programu. Tým je odľahčené samotné jadro algoritmu pre načítanie položiek.

Otvorené súbory sú uchovávane v zozname. Ten sa postupne prechádza a vytvárajú sa vlákna, kde každé vlákno agreguje jeden súbor do datovej štrukúry pre agregáciu – *red-black* strom. Po ukončení agregácie sú výsledky v hlavnom vlákne agregované do jedného *red-black* stromu (ak záznam už existuje, upraví sa). Paralelne s agregáciou je možné spustiť ďalšiu sadu vlákien pre spracovanie ďalších súborov. Tento algoritmus sa postupne opakuje pre všetky vstupné súbory. Vzhľadom na fakt, že sú využívané súbory uložené na disku, agregácia niekoľkých medzivýsledkov, ktorá prekrýva agregáciu jedného súboru môže agregovať viacej položiek. Nie je totiž nutné čakať na pomalé diskové operácie, všetky položky sú alokované a prístupné v pamäti RAM. Celý algoritmus je ilustrovaný na obrázku 1.

Počet vlákien spustených paralelne pre agregáciu (v obrázku j[1]) je vhodné zvoliť s ohľadom na rýchlosť. Celkový počet paralelných vlákien $(j + 1)$ je preto vhodné zistiť napríklad empirickým pozorovaním. Pri testoch bol zistený optimálny počet vlákien $4 + 1$ (4 pre agregáciu zo súborov, 1 pre agregáciu výsledkov z prechádzajúcej iterácie).

2.3 Filtrovanie záznamov na základe masky IP

Ukázalo sa neefektívne prekladať každý záznam na textovú reprezentáciu a následne filtrovať záznamy. Ako efektívnejší spôsob bol vybraný vytvorenie predpočítanej masky v *network byte order* a záznamy porovnávať na základe bitového súčinu predpočítanej masky a uloženého záznamu.

2.4 Pamäťová náročnosť

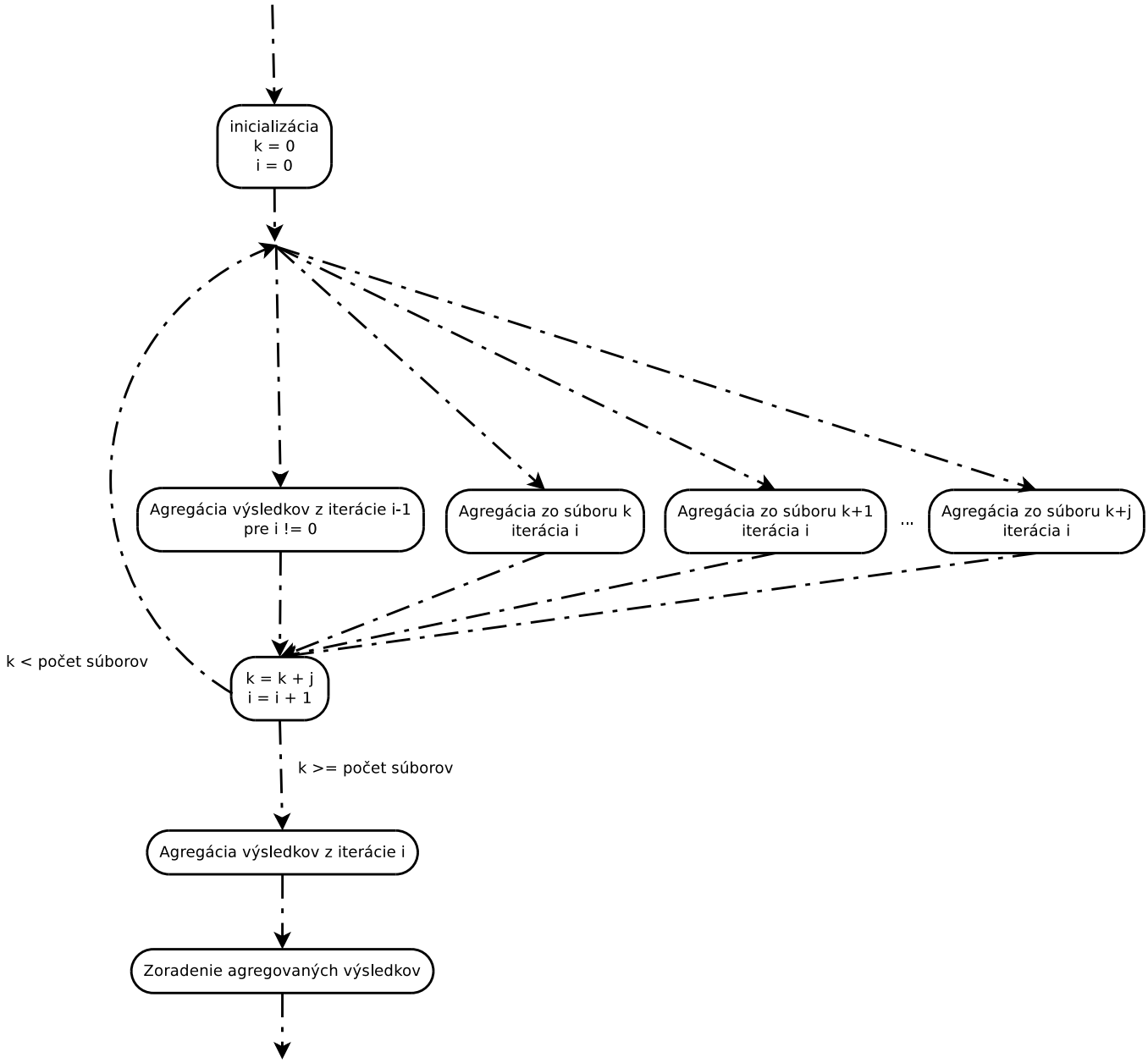
2.4.1 Agregácia pomocou IP adries

Program používa pre ukladanie záznamov z tokov strom. V najhoršom prípade bude obsahovať strom pre radenie všetkých n záznamov zo vstupu n uzlov. K tomu program uchováva j *red-black* stromov (j je počet vlákien pre agregáciu zo súborov, konštantanta), ktoré uchovávajú m položiek, zároveň platí relácia $m < n$. Pamäťová zložitosť je teda lineárna.

2.4.2 Agregácia pomocou portov

Pamäťová náročnosť je kontštantná $\mathcal{O}(1)$ vzhľadom na alokáciu poľa pre ukladanie záznamov podľa portov.

[1] V programe charakterizuje počet vlákien konštanta `THREAD_COUNT`.



Obr. 1: Ilustrácia algoritmu.

2.5 Časová náročnosť

2.5.1 Agregácia pomocou IP adries platí

$$((j+1)_p \cdot \mathcal{O}(\log n)) + \mathcal{O}(\log n) + \mathcal{O}(n) = \mathcal{O}(n)$$

$\mathcal{O}(\log n)$	–	uloženie položky v binárnom vyhľadávacom strome
$(j+1)_p \cdot \mathcal{O}(\log n)$	–	vyhľadanie položky v *red-black* strome pre $j+1$ vlákien, paralelne
$\mathcal{O}(n)$	–	prechod inorder binárnym stromom, n je počet agregovaných záznamov

Agregácia v *red-black* stromoch je uskutočnená paralelne. Časová náročnosť je teda lineárna $\mathcal{O}(n)$.

2.5.2 Agregácia pomocou portov

$$((j+1)_p \cdot \mathcal{O}(1))) + \mathcal{O}(n) = \mathcal{O}(n)$$

$(j+1)_p \cdot \mathcal{O}(1)$ – zápis do tabuľky záznamov podľa portu
$\mathcal{O}(n)$ – prechod inorder binárnym stromom, n je počet agregovaných záznamov

Časová náročnosť je teda lineárna $\mathcal{O}(n)$.

3 Experimentálne výsledky

Testy boli uskutočňované na procesore Intel®Core™i5-2540M 2.60GHz s jednotlivými veľkosťami cache – L1d cache: 32K, L1i cache: 32K, L2 cache: 256K, L3 cache: 3072K a SSD diskom. Pred spustením každého testu bola invalidovaná pamäť cache[2]. Nasledovná ukážka 1 poukazuje na spôsob spustenia testu. Štandardný výstup bol presmerovaný do `/dev/null` pre počítanie času algoritmu bez času potrebného pre výpis výsledkov (napr. eliminácia závislosti výpisu na pomalý terminál).

Ukážka 1: Ukážka spustenia testu

```
$ time ./flow -f 2014-01-27/00/ -s bytes -a srcip4/21 > /dev/null

real  0m2.202s
user  0m3.020s
sys   0m0.391s
```

Tabuľa testov sprehľadňuje dosiahnuté výsledky. Namerané výsledky sú mediánom troch meraní pre elimináciu vedľajších vplyvov.

Test	Doba behu programu	Mil záznamov/s
`./flow -f 2014-01-27/00/ -s bytes -a srcport`	2.929s	4.56
`./flow -f 2014-01-27/00/ -s bytes -a srcip`	5.358s	2.49
`./flow -f 2014-01-27/00/ -s bytes -a srcip4/32`	4.757s	2.80
`./flow -f 2014-01-27/00/ -s bytes -a srcip4/24`	4.120s	3.24
`./flow -f 2014-01-27/00/ -s bytes -a srcip4/16`	2.937s	4.55
`./flow -f 2014-01-27/00/ -s bytes -a srcip6/128`	2.544s	3.77
`./flow -f 2014-01-27/00/ -s bytes -a srcip6/96`	3.069s	4.35
`./flow -f 2014-01-27/00/ -s bytes -a srcip6/64`	2.847s	4.69
`./flow -f 2014-01-27/00/ -s bytes -a srcip6/32`	2.849s	4.68
`./flow -f 2014-01-27/00/ -s bytes -a srcip6/16`	2.811s	4.75

Tabuľka 1: Výsledky testov

V tabuľke nie je uvedené radenie na základe paketov. Vzhľadom na implementovaný algoritmus, toto radenie odpovedá časom nameraným pre radenie na základe počtu prenesených bajtov. Podobne je tomu tak u agregácii na základe cieľových portov, či cieľových IP adries.

[2] `sudo su -c "sync; echo 3 > /proc/sys/vm/drop_caches"`

4 Zhrnutie

Projekt bol implementovaný a otestovaný. Agregácia a triedenie datových tokov dosahuje výrazne rýchlejšie výsledky v porovnaní s referenčným nástrojom *nfdump*. Optimalizácia bola preto úspešná. Pre ďalší vývoj by bolo vhodné zvážiť použitie inej datovej štruktúry s časovou zložitosťou vyhľadania $\mathcal{O}(1)$ pre IP záznamy a dynamické zistenia počtu použitých vlákien pre paralelizmus vzhľadom na aktuálne používaný procesor.

Literatúra

[1] Knižnica `libtree`. Dostupné on-line: `https://github.com/fbuihuu/libtree` [cit. 03.03.2014].