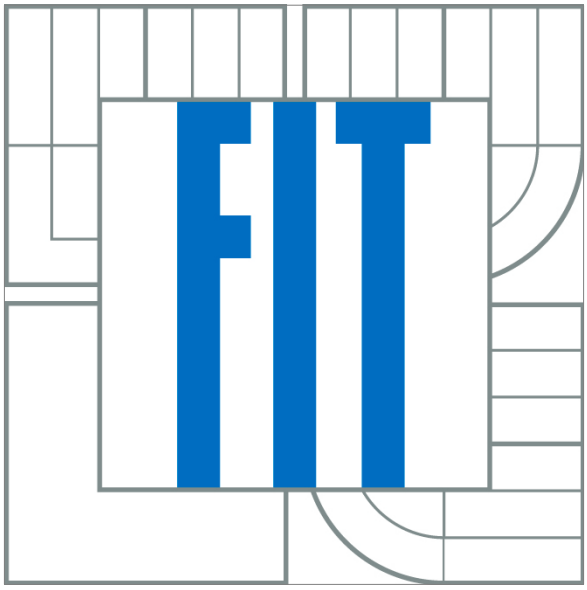

Dokumentácia k projektu do predmetu PDS

Projekt: „Varianta 1: Agregácia a triedenie flow dát“

Matúš Fedorko,
xfedor01@stud.fit.vutbr.cz

Brno, 27. apríla, 2014

1 Úvod

Účelom projektu bolo urýchlenie spracovania dát z denného sieťového prevozu, tzv. netflow dát. Cieľom bolo pritom dosiahnuť rýchlejšieho spracovania ako štandardná utilita *nfdump*, ktorá dokáže agregovať a zotriediť približne 1.5 milióna záznamov za sekundu pri agregovaní podľa IP adresy a približne 2 milióny záznamov za sekunudu pri agregovaní a triedení podľa portov.

V nasledujúcich kapitolách sú bližšie popísané použité nástroje, postup riešenia a zhrnuté dosiahnuté výsledky.

2 Použité nástroje

Program bol implementovaný v jazyku C++ a to najmä pre jeho natívnu, kompilovanú rýchlosť, možnosť použitia pokročilých programovacích techník ako sú šablóny a objektové programovanie a v neposlednom rade aj kôli bohatej štandardnej knižnici, ktorá obsahuje už hotové hešovacie kontajnery a radiace algoritmy.

Ako kompilátor bol zvolený program GCC, ktorý je bežnou súčasou mnohých unixových distribúcií a vo svojej najnovšej verzii disponuje aj kvalitnou podporou aktuálnej normy jazyka C++11. Program bol vyvíjaný a testovaný v GCC verzie 4.8 a využíva niektoré z jeho neštandardných optimalizačných rozšírení. Ostatné prekladače nie sú momentálne podporované.

Z operačných systémov sú podorované len unixové operačné systémy, ktoré implementujú funkcie sysconf, write a mapovanie súborov do pamäte. Program bol vyvíjaný a testovaný v systéme Ubuntu 13.10.

Okrem toho na rekurzívne prechádzanie priečinkov súborového systému je použitá knižnica boost::filesystem vo verzii 1.53, ktoré je však prakticky štandardom v jazyku C++ a je bežne dostupná v repozitároch všetkých hlavných linuxových distribúcií.

Výkon bol meraný nástrojom Intel VTune Amplifier XE 2013, ktorý je na Linuxe pre nekomerčné účely zdarma a okrem klasického zobrazovania časovo najnáročnejších funkcií umožňuje zistiť aj mnohé HW štatistiky ako napr. cache miss rate, memory stalls, alebo využitie jadier procesoru v konkrétej funkcii. Taktiež dokáže priamo pre každy riadok kódu zobraziť dĺžku jeho trvania a tým presnejšie lokalizovať problematické miesta v programe.

3 Implementácia

Samotná implementácia je z veľkej časti postavená na využití šablón a šablónového programovania. Týmto spôsobom bolo možné docieliť optimálneho spracovania pre jednotlivé

pípady agregácie a ich ľubovoľné kombinácie. Z dôvodu výrazne odlišného prístupu pri spracovní portov a IP adries je však možné sledovať rozdelenie programu na dve hlavné časti. Prvá je implementovaná v súbore ports.cpp a zaoberá sa agregáciou a triedením portov a druhá je implementovaná v súbore ips.cpp a zaoberá sa spracovaním IP adries.

Spracovanie IP adries

Na agregáciu IP adries bol použitý štandardný kontajner std::unordered_map, ktorý je na rozdiel od std::map interne implementovaný ako hešovacia tabuľka a teda garantuje amortizovanú konštantnú časovú zložitosť pri prístupe k prvkom, ktoré sú v ňom uložené. K degradácii výkonu však môže dôjsť častým prehešovávaním celého kontajneru. Aby sa tomuto predišlo, tak program vždy ešte pred začatím agregácie zavolá funkciu reserve, ktorou si v predstihu, na základe počtu záznamov v súbore, alokuje dostatočný počet voľných bucket-ov.

Okrem predalokovávania pamäte však bolo potrebné vyriešit aj samotné hešovanie IP adries, pretože štandardná knižnica neobsahuje pre tento prípad podporu. Najlepším riešením sa ukázalo byť počítanie heše IPv6 adries ako logickej funkcie XOR jednotlivých 32 bitových slov. V prípade IPv4 adries je funkcia triviálna a jednoducho len kopíruje hodnotu 32 bitového slova adresy.

Ďalšie urýchlenie prinieslo paralelné spracovanie viacerých súborov naraz, pričom k vytváraniu vlákien bola použitá štandardná trieda std::Thread.

Spracovanie portov

Agregácia portov je z pohľadu optimalizácie zaujímavejšia, pretože množstvo rôznych portov je v porovnaní s počtom rôznych IP adries relatívne malé. Tento fakt umožnuje implementovať agregáciu priamo pomocou lineárneho poľa o 65536 prvkoch a vyhnúť sa dodatočnej réžii spojenej s udržiavaním hešovacej tabuľky.

Vznikne tak algoritmus, kde hodnoty jednotlivých portov priamo určujú index do poľa, v ktorom sú počty bytov a paketov pre daný port. Táto schéma je typickým príkladom pre použitie SW prefetch inštrukcií, pretože indexácia poľa má v zásade náhodný charakter, čo dnešné procesory nie sú schopné detekovať, ale z pohľadu konkrétneho algoritmu je jasné, ktorá pamäťová bunka bude potrebná v ďalšej iterácii.

Problém s týmto algoritmom je v tom, že vo vstupnom súbore sa môžu nachádzať aj záznamy s nulovým počtom bajtov a paketov a teda nie je možné rozoznať, či daný port bol v súbore obsiahnutý, alebo nie. Toto bolo vyriešené využitím navýznamnejšieho bitu v položke udávajúcej množstvo množstvo paketov na informáciu o platnosti daných dát. Nie je to síce čisté riešenie, ale vychádza z dostupných testovacích súborov, pre ktoré by malo byť 63 bitov dostačujúcich. Navyše je využitý bit v políčku s informáciou o množstve paketov, ktoré by malo byť vždy menšie alebo rovné počtu prenesených bytov.

Vzhľadom na výrazne rýchlešiu agregáciu portov ako agregáciu IP adries sa paralelizácia nepreukázala byť až tak výrazne výhodná. Namiesto toho sa ukázalo byť výhodným vytvorenie špeciálneho vlákna s malou prioritou, ktorého jedinou úlohou je prečítať prvý bajt z každej stránky namapovaného súboru a prinútiť tak operačný systém, aby stránku nahral z disku do RAM pamäte skôr ako ju bude hlavné spracovateľské vlákno potrebovať.

K radeniu portov bola využitá štandardná funkcia std::sort podobne ako v prípade IP

adries, ale s tým rozdielom, že namiesto reálnych dát sa radí pole indexov na tieto dáta, aby sa znížilo množstvo presunov v pamäti.

Čítanie zo súboru

V oblasti čítania z disku a zapisovania na disk bolo skúšaných niekoľko postupov. Prvým boli pokusy o paralelné prednačítanie súboru do pamäte pomocou funkcií posix_fadvise a readahead, avšak tie nepriniesli očakávané zlepšenie výkonu. Mierny nárast výkonu prišiel až s použitím mapovania do pamäte, ktoré bolo ďalej vylepšené pomocou prefault-ovacieho mechanizmu popísaného vyššie.

Na zapisovanie na stdout bola pôvodne použitá štandardná funkcia fprintf, ktorá však bola nahradená vlastnou funkciou, ktorá výstup najprv vygeneruje do buffer-u a rozdelí na bloky o veľkosti presne 256KB a potom ich na stdout zapíše pomocou funkcie write.

Zapisovanie a hlavne čítanie z klasického pevného disku však aj naďalej zostáva jedným z najužších miest v programe.

4 Záver

Ako vidno z priložených grafov a tabuliek a podľa toho čo bolo napísané v úvode sa spracovanie oproti utilite nfdump podarilo v prípade IP adries urýchliť viac ako 2-krát a v prípade portov viac ako 5-krát. Priložené grafy ukazujú porovnanie medzi prvotnou neoptimalizovanou verziou programu a verziou po aplikovaní optimalizácií a postupov popísaných v tomto dokumente.

Na počítači s klasickým diskom je však spracovanie brzdené najmä rýchlosťou čítania. Toto by sa dalo zlepšiť napríklad komprimáciou vstupných súborov, čím by sa znížil objem dát, ktoré treba načítať.

Prvotná verzia	Optimalizovaná verzia
0.677	0.524
0.771	0.595
0.726	0.565
0.727	0.557
0.746	0.587
0.7294	0.5656

Tabuľka 1: Časy optimalizovanej a prvotnej verzie programu v sekundách pri agregavaní podľa ip adries (srcip) a triedení podľa bajtov (bytes)

Prvotná verzia	Optimalizovaná verzia
0.305	0.088
0.313	0.069
0.308	0.071
0.316	0.068
0.308	0.066
0.31	0.0724

Tabuľka 2: Časy optimalizovanej a prvotnej verzie programu v sekundách pri agregavaní podľa portov (srcport) a triedení podľa bajtov (bytes)