PDS Přenos dat, počítačové sítě a protokoly
FIT VUT Brno

Agregace a třídění flow dat - dokumentace

30. dubna 2014

Jan Bednařík

1 Úvod

Tato dokumentace popisuje návrh a realizaci programu `flow`, jehož úkolem je agregovat a třídit síťové toky zachycené nástrojem *nfdump*. Kromě získání korektních výstupů bylo hlavním cílem dosažení co nejefektivnějšího řešení, následující text se tedy zaměřuje především na implementační detaily kritické pro rychlý běh programu. Kapitola 2 popisuje návrh programu, způsob reprezentace klíčových dat a využité algoritmy, kapitola 3 vysvětluje princip optimalizačního řešení a kapitola 4 nakonec shrnuje výsledky testů nad reálnými daty.

2 Implementace

Při výběru programovacího jazyka byla zohledňována předně časová efektivnost výsledného spustitelného programu, ale také možnost rychlé implementace a prototypování. Zvolen byl jazyk C++, neboť díky kompilaci do strojového kódu obecně vykazuje vyšší rychlost běhu programu, než interpretované jazyky, navíc oproti potenciálně rychlejším jazykům (jako C) nabízí objektově orientovaný přístup a pokročilé abstraktní datové struktury.

Funkčně i časově významnými částmi programu je načítání vstupních souborů, agregace toků a jejich následné řazení. Načítání i zpracování souborů probíhá sekvenčně. Agregace je prováděna za pomocí vyhledávací struktury *hashovací tabulka*, v níž je asociována reprezentace IP adresy nebo síťového portu (v závislosti na vstupních parametrech programu) s hodnotou v podobě dvojice počet bajtů - počet paketů. Řazení je následně provedeno nad vektorem dvojic klíč - hodnota z původní hashovací tabulky za pomocí metody *quicksort*.

2.1 Datové struktury

IP adresa Z experimentálních výsledků (viz 4) vyplývá, že na rychlosti běhu programu se značnou měrou podílí výběr vhodné datové struktury pro reprezentaci IP adresy (případně portu) a pro tabulku s asociativním vyhledáváním.

Program podporuje agregaci podle IP adresy verze IPv4 i IPv6, pro jednu libovolnou adresu je tedy nutné vyhradit nejméně 128 bitů. Současně je nezbytné zachovat informaci o verzi IP. S ohledem na standardní podporované datové typy jazyka C++ byla zvolena následující datová struktura:

```
struct IP {
    uint64_t ip128bit[2];
    uint32_t sa_family;
};
```

Jak ukazují experimentální výsledky, binární reprezentace IP adresy na co nejmenším počtu bitů může například oproti nabízející se textové reprezentaci (pomocí datového typu `string`) zrychlit běh až 3,7krát.

Hashovací tabulka Pro implementaci hashovací tabulky je využit standardní kontejner `std::unordered_map` jazyka C++, který je efektivní díky konstantní časové složitosti *O(1)* přístupu k prvku [1].

```
unordered_map<IP, PacketsBytes, Hasher, EqualFn>
    *aggregatedFlows;
```

`PacketsBytes` zde představuje dvojici počet paketů - počet bajtů, `Hasher` je funkční objekt realizující vlastní hashovací funkci a `EqualFn` je funkční objekt realizující porovnání dvou klíčů.

Původní implementace programu `flow` pracovala s kontejnerem `std::map`, nicméně ten je interně implementován jako binární vyhledávací strom a jak ukazují experimentální výsledky, využití této struktury značně zpomaluje běh programu.

Datový typ `uint64_t` pro reprezentaci adresy IP byl zvolen z důvodu efektivnější implementace hashovací funkce. Delší pole složené z jednotek menšího datového typu (např. `uint8_t[16]`) vede na více kroků cyklu (hashuje se iterativně po položkách pole).

2.2 Algoritmy

Řazení již agregovaných dat probíhá nad kontejnerem `std::vector<pair<IP, PacketsBytes> >` za pomocí algoritmu `std::sort` ze standardní C++ knihovny `<algorithm>`. Řazení dosahuje lineárně logaritmické časové složitosti $O(N * logN)$ [2].

2.3 Spuštění programu a omezení

```
flow -f adresar -a agregace -s razeni
```

```
    -a
            srcip = IPv4/32 nebo IPv6/128
            srcip4/mask = IPv4/1-32
            srcip6/mask = IPv6/1-128
    -s
            bajty
            pakety
```

Nutno zdůraznit, že hodnota `directory` parametru `-f` opravdu musí zastupovat cestu k *adresáři*, nikoliv konkrétnímu souboru pro zpracování.

3 Optimalizace

Kromě optimalizací programu prostřednictvím vybraných datových struktur a algoritmů byla rovněž dodatečně doplněna optimalizace maskování IP adresy (podle zadaného parametru). Maskování probíhá iterativně po jednotlivých bajtech IP adresy. Namísto průchodu celou adresou se však začíná maskovat až od toho bajtu, na němž je v dílčí 8bitové masce přítomna aspoň jedna 0. Tato úprava algoritmu přinesla další časové zrychlení (viz kapitola 4).

4 Experimentální výsledky

Program byl testován z hlediska rychlosti běhu při zpracování referenčních vstupních souborů zachycujících anonymizované záznamy z páteřní sítě 27.1.2014 v době od 07:00 do 08:00. Daný časový interval obsahuje 1 123 353 zaznamenaných datových toků. Testování proběhlo na referenčním testovacím stroji `test-nfsen.net.vutbr.cz` (OS Red Hat Enterprise Linux Server release 6.5 (Santiago), 12 x86_64 CPU, 44 GB RAM, vstupní data uložena v RAM disku).

Tabulka 1 shrnuje výsledky měření při agregaci podle IP adresy a podle portu a porovnává efektivnost implementací ve čtyřech kombinacích, kdy je IP adresa reprezentována binárně (*uint64_t[2]*) či textově (*string*) a kdy je tabulka s asociativním vyhledáváním interně implementována jako binární vyhledávací strom (*map*) nebo jako hashovací tabulka (*unordered_map*).

Jedná se o implementaci bez použití optimalizace popsané v kapitole 3. Čas byl měřen nástrojem `time` s tím, že výstup je přesměrován do souboru:

```
./flow -f /mnt/ramfs/2014-01-27/ -a srcip -s packets
./flow -f /mnt/ramfs/2014-01-27/ -a srcport -s packets
```

Tabulka 1: Výsledky měření doby běhu programu `flow`.

TEST	AGREGACE	ČAS [s]	TOKŮ/s
map, string	ip	35,063	32095
	port	4,564	246572
map, uint64_t[2]	ip	9,402	119693
	port	4,567	246410
unordered_map, string	ip	12,567	89548
	port	1,147	981127
unordered_map, uint64_t[2]	ip	5,223	215461
	port	1,126	999425

Z výsledků plyne, že kombinace kontejneru `std::unordered_map` a binární reprezentace IP adresy na 128 bitů vede k nejlepším výsledkům.

Výsledek testování finální implementace programu `flow` nad celodenním záznamem síťových toků na zachycených na páteřní síti shrnuje tabulka 2. Výrazný rozdíl v počtu zpracovaných toků za sekundu oproti testům shrnutým v tabulce 1 může být dán částečně zahrnutím optimalizace maskování IP adresy, kolísající přesností měření závislé na množství vstupních dat a rovněž aktuální vytížeností testovacího stroje, který v době testování zatěžovalo více uživatelů.

Tabulka 2: Výsledek testování finální implementace porogramu `flow`

TEST	AGREGACE	ČAS [s]	TOKŮ/s
flow (celodenní záznam)	ip	128	3408416
	port	37,175	11735770

5 Závěr

V textu byl detailně nastíněn návrh a implementace programu `flow`. Vysvětleny byly vlivy využitých datových struktur i algoritmů na časovou efektivitu programu a byly shrnuty provedené optimalizace. Z uvedených výsledků experimentů provedených na reálných datech vyplývá, že program dosáhl nejlepších výsledků při binární reprezentaci IP adresy a využití tabulky s asociativním vyhledáváním implementované pomocí hashovací tabulky.

Finální výsledek, tedy téměř 3,5 milionu zpracovaných toků za sekundu při agregaci podle IP adresy převyšuje rychlost, jíž dosahuje existující řešení v podobě programu `nfdump`, nicméně je nutné brát v potaz, že testování proběhlo na velice výkonném stroji a data se nečetla z pevného disku, nýbrž z operační paměti, tudíž lze očekávat, že na běžném PC či slabším serveru nebudou výsledky zdaleka tak zářné.

Program lze dále urychlovat. Jako další optimalizace se nabízí předně paralelizace na úrovni čtení souborů, agregace i řazení.

Reference

[1] Network, T. C. R.: Standard C++ Library reference [online]. 2013. URL http://www.cplusplus.com/reference/unordered_map/unordered_map/

[2] Network, T. C. R.: Standard C++ Library reference [online]. 2013. URL http://www.cplusplus.com/reference/algorithm/sort/