Profilování haldy s nízkou latencí

Ing. Jan Tománek, Ing. Daniel Langr, Ph.D.
České vysoké učení technické v Praze

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Problém

S rostoucí komplexitou programu zpravidla rostou i jeho požadavky na paměť. Programy napsané v jazycích C/C++ obvykle získávají paměť potřebnou pro své výpočty pomocí **dynamických alokací** z haldy. Vysoká četnost provádění dynamických alokací se nicméně může negativním způsobem projevit na době běhu programu a stát se tak původcem jeho **neefektivity**.

Ve vícevláknových programech je tato situace ještě kritičtější – halda je **sdíleným prostředkem** všech vláken. I v případě, že každé z vláken programu jinak pracuje nad svými vlastními daty, může mít paralelní provádění dynamických alokací a uvolnění za následek problémy se **škálovatelností** celé aplikace.

Profilování haldy

Interakce programu s haldou:

- C (libc) – **`malloc`**, **`calloc`**, **`realloc`**, **`free`** atd.
- C++ (libstdc++/libc++) – **`operator new`**, **`operator delete`** atd.

Příklady sledovatelných veličin:

- **Počty** provedených dynamických **alokací** a uvolnění
- **Velikost** dynamicky alokované **paměti**
- **Histogram** počtu alokací a jejich velikostí
- **Místa volání** alokačních funkcí
- **Vývoj** dynamicky alokované paměti **v čase**

Existující profilery

Existující nástroje pro profilování haldy dokáží velmi přesně monitorovat její využití. S jejich pomocí lze **odhalit problematická místa** programu a ta následně **optimalizovat**.

Nejrozšířenější profilery haldy v prostředí **OS Linux**:

- **Heaptrack**
- Valgrind (nástroj **Massif**)
- Heap profiler v **gperftools**

Nedostatky existujících profilerů:

- Přidávají značnou **časovou režii (latenci)** k běhu programu.
- Generují příliš **velké výstupní soubory** (Heaptrack).
- **Omezují škálovatelnost** profilovaného programu.

Důsledkem těchto nedostatků je **obtížné profilování** paměťově náročných vícevláknových aplikací.

Vlastní profiler

Práce navrhuje nový profiler haldy s názvem **Heaprof**, který svým zaměřením cílí na profilování vícevláknových aplikací. Svou konstrukcí se snaží **eliminovat nedostatky** ostatních profilovacích nástrojů a klade přitom důraz na **minimální latenci** profilování.

Toho je dosaženo aplikací dvou základních pravidel:

1. **Vzájemně neblokovat programová vlákna**
   Každé vlákno profilovaného programu ukládá nezbytná profilovací data do své **vlastní cache** (tzv. **profil vlákna**). Volání (de)alokační funkce tak nezpůsobí, že by jedno vlákno programu muselo čekat na dokončení činnosti jiného vlákna.
2. **Agregovat nasbíraná data**
   Profiler disponuje **separátním vláknem**, které v pravidelných časových intervalech sbírá data z profilů vláken a v agregované podobě je zapisuje na konec výstupního souboru.

Realizace

Jádrem profileru Heaprof je **sdílená knihovna**, jejíž kód je při spuštění profilování nahrán pomocí proměnné prostředí **`LD_PRELOAD`** do profilovaného programu.

Knihovna definuje vlastní verze funkcí `malloc`, `free` a spol. Díky tomu je schopna **odchytit jejich volání**. Minimalistickou ukázku této techniky ilustruje kód 1.

```c
#include <dlfcn.h>  // dlsym

extern "C" void* malloc(size_t size) {
    using malloc_t = decltype(&::malloc);
    static auto real_malloc
        = (malloc_t) dlsym(RTLD_NEXT, "malloc");

    void* result = real_malloc(size);
    fprintf(stderr, "malloc(%zu)->%p\n", size, result);
    return result;
}
```

Kód 1: Odchycení volání funkce `malloc`

Klíčovou schopností profileru je získání **backtrace**. Pro podporu této funkcionality jsou použity knihovny:

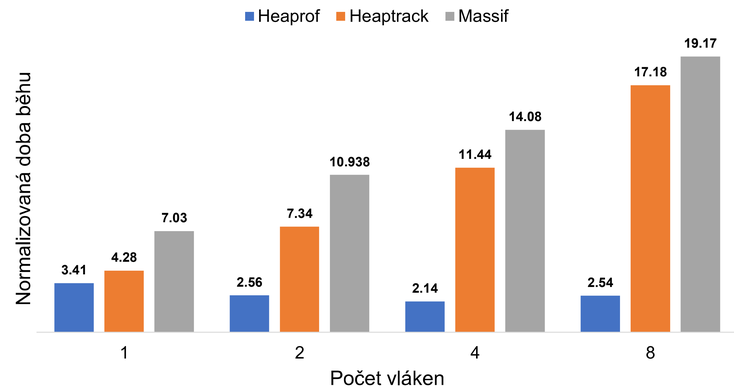
- **libunwind** – `https://github.com/libunwind/libunwind`
- **backward-cpp** – `https://github.com/bombela/backward-cpp`

Heaprof podporuje **3 režimy** profilování, které se od sebe liší úrovní detailu sbíraných dat. Implementace je provedena v jazyce C++. Součástí projektu je dále samostatný program určený k vizualizaci získaných dat.

Dosažené výsledky

Heaprof byl ve svém nejdetailnějším režimu profilování porovnán s ostatními profilovacími nástroji na sadě paměťově náročných vícevláknových benchmarků. Primární oblastí zájmu byl dopad profilování na celkovou dobu běhu programu:

- $T_B(P)$ – Doba běhu benchmarku B na P vláknech
- $T_B^H(P)$ – Doba běhu při profilování profilerem H
- $R_B^H(P)$ – Poměr $T_B^H(P)/T_B(P)$
- $R^H(P)$ – Průměr $R_B^H(P)$ přes všechny benchmarky



Obrázek 1: Závislost průměrné doby profilování na počtu vláken

Naměřené hodnoty $R^H(P)$ – normalizovaná doba běhu – jsou zaneseny do grafu na obrázku 1. Celkově se ukázalo, že:

- Heaprof má v jednovláknových programech **srovnatelnou nebo nižší latenci**, než ostatní profilery.
- **Latence** navrženého profileru **neroste s počtem vláken**. Profilovaný program tak **může škálovat**.
- Heaprof má **řádově menší výstupní soubory**, než Heaptrack.

Budoucí práce a vylepšení

- **Zefektivnění** procesu získávání **backtrace**
- **Komprimace** výstupních souborů
- **GUI prohlížeč** nasbíraných dat

Odkaz na práci

Tománek, Jan. *Profilování haldy s nízkou latencí*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.