Profiliranje programske kode

Sistemska programska oprema
Nejc Hirci, 63180335
2020/2021

1. Uvod

Profiliranje je oblika dinamične analize programa, pri kateri želimo pridobiti informacije o bolj natančnem poteku izvajanja kot ga lahko predvidimo s teoretično analizo. Nikakor pa ne gre za novodobni koncept, saj prva orodja za profiliranje segajo že v leto 1970 na platformah IBM/360 in IBM/370, kjer pa so bila v večini osnovana na časovnih prekinitvah izvajanja, ob katerih je orodje shranjevalo "vroče točke" v izvajalni programski kodi. Kasneje se je s pojavom Unixa prvič pojavilo nekoliko bolj razvito orodje `prof` in še kasneje njegov naslednik `gprof`, ki je poleg ATOM platforme prav tako namenjene profiliranju po PLDI-ju eden od 50 najpomembnejših prebojev med leti 1980 in 2000.

Orodja za profiliranje so izjemno pomembna za razumevanje obnašanja programa v njegovem končnem okolju, kar se uporablja na večih nivojih računalniškega razvoja:

- pri razvoju računalniških arhitektur za ocenjevanje izvedbe programov na novih platformah,
- razvijalci programske opreme jih pogosto uporabljajo za analizo programov in iskanje kritičnih sekcij,
- pri pisanju prevajalnikov razvijalcem pomagajo preučiti zanesljivost algoritmov za razporejanje ukazov in napovedovanje vejitev

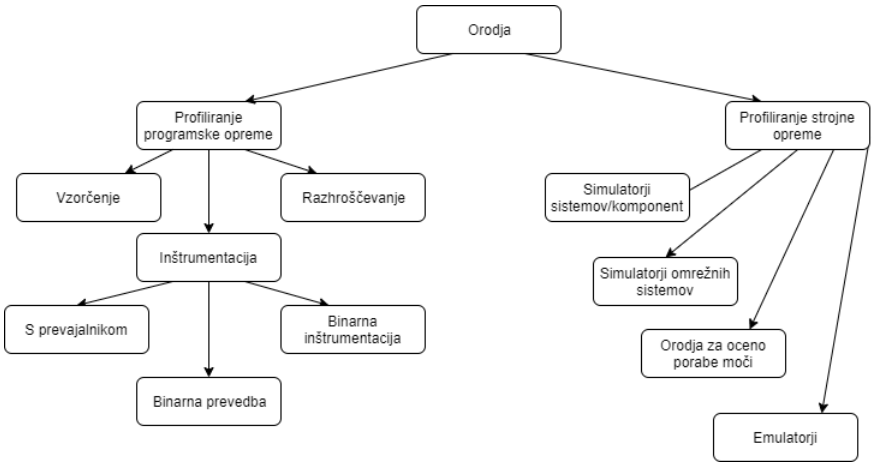
Uporabne informacije orodij za profiliranje

V grobem nam orodja za profiliranje lahko ponujajo več različnih izhodnih informacij:

- **profil** ali statistični povzetek dogodkov programa, ki nam zgolj preda informacijo o številu klicev posamezne vrstice programa v njegovi izvirni ali zborni obliki,
- **sled** ali tok posnetih dogodkov programa, ki je predvsem pomemben za analizo paralelnih programov, da lažje razumemo časovne odnose med dogodki (npr.

čakanje na sporočila ali različne oblike sinhronizacije) in njegove kritične sekcije
- **graf klicev** je razširjena oblika sledi, ki nam pokaže čase, frekvence in verige funkcijskih klicev za lažje razumevanje konteksta, v katerem so bili klici izvedeni

Pregled skupin orodij



Slika 1: Pregled skupin orodij za profiliranje

Glede na njihov obseg na najvišjem nivoju ločimo orodja za profiliranje namenjena programski opremi in strojnji opremi. V nadaljevanju se bomo posvetili samo orodjem za programsko opremo. Znotraj njih kategoriziramo orodja predvsem glede na njihove funkcionalnosti ter predvsem metodologijo pridobivanja informacij (npr. lahko temeljijo zgolj na že uveljavljenih knjižnicah ali posegajo zgolj po platformah za inštrumentacijo kot je ATOM), saj lahko te močno povečajo režijske stroške v primeru pridobivanja enakih informacij.

2. Orodja za programsko opremo

Ko uporabnik želi razumeti obnašanje programa med izvajanjem, kar je pomembno predvsem za optimizacijo med razvojem, lahko uporabi različne skupine orodij. Preprosti primer tega je, da uporabnik želi zmanjšati porabo pomnilnika programa, ki je zelo pogosti nivo optimizacije za vgrajene aplikacije, in izboljšati hitrost izvajanja, ali pa preprosto želi izboljšati zanesljivost in pravilnost programa, s tem da sledi aktivnim in potencialnim napakam. Seveda se ob tem uporabnik poleg orodij za profiliranje zanaša tudi na različne razhroščevalnike in druge knjižnice. Sam potek analize programa pa lahko poteka s statično analizo ali dinamično tekom izvajanja.

Bolj podrobno poznamo naslednje skupine orodij:

1. **Profiliranje glede na dogodke**

- zbira informacije o dogodkih, ki jih je definiral uporabnik, ti pa lahko potrebujejo določene prestrežbe s strani operacijskega sistema (hooking)
- ko se zgodi željeni dogodek bo orodje zbralo vse z njim povezane karakteristike programa

2. **Profiliranje z vzorčenjem**
    - zbira informacije v določenih intervalih ali z določeno frekvenco
    - cilj je da pridobi dovolj vzorcev, da lahko sestavi čimbolj natančno statistično sliko programa tekom izvajanja
    - prednost zelo **nizkih dodatnih režijskih stroškov**, vendar zelo **težko natančno oceni število klicev** posamezne metode (ali celo povsem zgreši zelo kratke metode)
    - najbolj uporabno pri iskanju **ozkega grla** programa
    - dober primer ročnega vzorčenja je preprosto merjenje odsekov izvajanja koda, ki ste se ga tekom razvoja svojih programov že nedvomno uporabili
    - za primere ko želimo analizirati procesorsko zelo zahtevno in morda večnitno kodo, ki pa potrebuje veliko vhodnih podatkov (npr. procesiranje slik), lahko vzorčimo asinhrono in s tem izločimo zakasnitve zaradi pridobivanja podatkov
3. **Profiliranje z inštrumentacijo**
    - vsebuje več podkategorij, vendar v splošnem deluje z vstavljanjem ali spreminjanjem izvorne kode programa v pomembnih sekcijah s pomočjo posebnega orodja znotraj profilnika
    - to orodje za spreminjanje kode programa si lahko pomaga s **prevajalnikom** (npr. gprof), uporablja **binarno prevajanje**, **binarno inštrumentacijo** ali neki kombinacijo teh metodologij
    - skoraj vedno vpeljejo **visoke dodatne režijske stroške**, kar lahko popači relativni pomen ozkih grl na učinkovitost, tako zaradi časovnega dodatka kot tudi zaradi metod, ki bi jih prevajalnik že deloma izboljšal
    - smiselno uporabljati nad množico razredov ali funkcij, ki opravljajo visoko nivojske operacije
    - pomembno pridobimo natančno informacijo o številu klicev vseh inštrumentiranih metod

2.1 Profiliranje z inštrumentacijo

Binarno prevajanje (angl. binary translation)

Gre za postopek ponovnega binarnega prevajanja, kjer orodje kot vhod dobi vnaprej prevedeno binarno sliko programa in jo prevede v strojno kodo. Najpomembnejši del takšnega orodja je, da med procesom prevajanja orodje zbira dodatne informacije o programu s pomočjo tolmača (angl. interpreter). Med tem postopkom lahko orodje tudi vstavlja, odstranjuje ali drugače spremeni vhodno binarno sliko. Ta postopek lahko počnemo na dva načina:

- **statično** - konvertira vhodno kodo v izvršljivo strojno kodo za ciljno arhitekturo brez izvajanja
    - v praksi je lahko zelo težko izvedljivo, saj lahko prevajalnik zgreši določene vejitve, za katere bi morali bolj tesno simulirati izvajanje programa (npr. posredni skočni ukazi)
    - primer tega je bilo recimo prevajanje igre StarCraft iz leta 1998 iz x86 na ARM arhitekturo, kjer pa je bilo sicer potrebno še precej povratnega inženirjenja
- **dinamično** - konvertira vhodno kodo v izvršljivo med časom izvajanja
    - ponavadi se izvaja nad manjšimi bloki programa
    - mnogo bolj natančna pri pokrivanju kode in predvidevanju poteka vejitev (nikakor pa ne zagotavlja 100% natančnosti), vendar vpelje veliko dodatnih režijskih stroškov, ti se pojavijo predvsem, ker mora orodje tekom izvajanja večkrat shranjevati in nazaj nalagati kontekst izvajanja (programske registre, sklad, itd.)
    - ne glede na to je lahko dolgoročno mnogo bolj uporabna za avtomatsko ali ročno optimizacijo, saj ima mnogo več znanja o izvajanju programa
    - nekaj režijskih stroškov orodja zmanjšajo z memoizacijo prevedene kode, s katero izločimo ponavljajoča prevajanja segmentov

Binarna inštrumentacija (angl. binary instrumentation)

Pri tej metodi orodje vstavlja dodatno kodo v izvršljivi modul programa ter jo nato preda različnim orodjem za nadaljnjo analizo. Podobno kot pri binarnem prevajanju lahko postopek deluje statično ali dinamično. Za sam razvoj je bila posebej pomembna platforma ATOM, ki je precej olajšala gradnjo orodij z binarno inštrumentacijo.

Inštrumentacija s pomočjo prevajalnika (angl. compiler-assisted)

Kot pove že ime izkorišča infrastrukturo prevajalnika, da vstavi dodatne rutine za analizo (npr. inštrumentira funkcije ali razrede z dodatnimi analitičnimi rutinami pred začetkom in po koncu klicev).

2.2 Profiliranje z vzorčenjem

Vzorčenje s časovnimi števci (angl. timer-based)

Gre za najbolj preprosto obliko vzorčenja, ki predstavlja osnovo v večini sodobnih komercialnih profilnikov. Ponavadi uporabljajo že vgrajene števce operacijskega sistema ali strojne opreme. Lahko pridobijo le grobo sliko izvajalnega časa posameznih sekcij v programu, vendar nudijo dovolj informacij za odkrivanje večjih ozkih grl.

Vzorčenje z dogodki (angl. event-based)

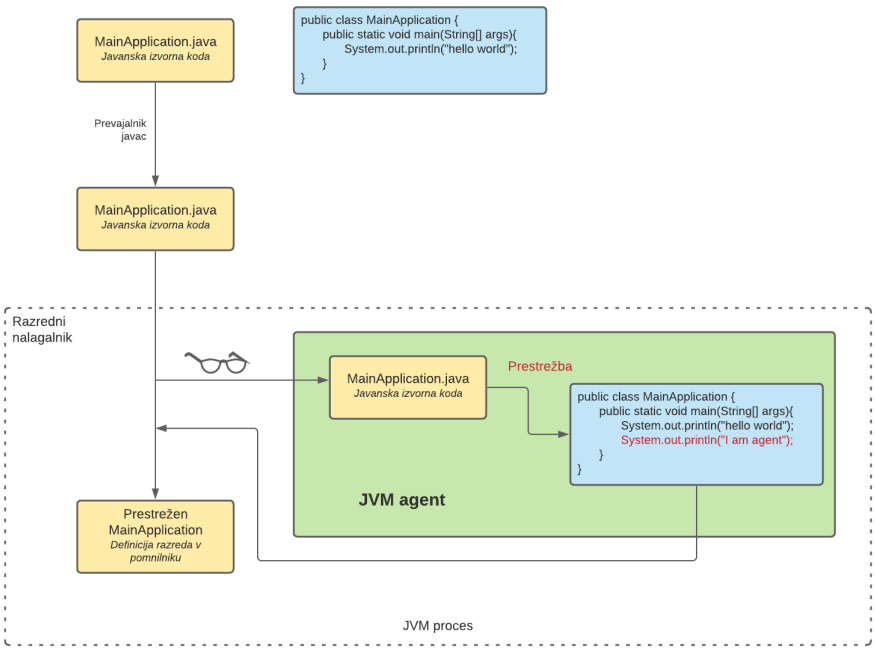
Podatke vzorčijo ob določenih programskih ali strojnih dogodkih. Uporabnika lahko recimo zanima število zgrešitev strani v predpomnilniku med izvajanjem programa, ali pa število določenih sistemskih klicev, ob katerih lahko profilnik vzorči informacije. Da se ti dogodki ujamejo v past in jih lahko profilink pravilno pomni seveda potrebujemo primitivne knjižnice operacijskega sistema.

Vzorčeneje po ukazih (angl. instruction-based)

Med načini vzorčenja nam podajo najbolj natančen profil programa. Delujejo tako, da prekinjajo program med izvajanjem po določenem številu izvedenih ukazov in pridobijo informacije o stanju strojnih števcev. Te lahko potem višjenivojska orodja uporabijo za bolj natančno analizo učinkovitosti programa. Njihova zanesljivost je seveda odvisna od periode vzorčenja.

3. Profiliranje v Javi

Ko aplikacija že teče v produkcijskem okolju, se lahko pojavijo težave, ki jih zelo težko simuliramo med razvojem. V programskem jeziku Java imamo nekoliko drugačno pozicijo, kjer lahko zaradi načina izvajanja programov v JVM (Javinem Virtualnem Stroju), izkoristimo že vključen Instrumentation API, s katerim lahko ustvarimo **java agente**. Java agenti se lahko dinamično pripnejo na nek nabor programske kode aplikacije in jo spremenijo za pridobivanje dodatnih informacij v nadaljnji analizi, ali preprosto shranjujejo vmesna stanja aplikacije. Večina Javinih že razvitih profilnikov kot je JProfiler prav tako uporabljajo java agente.



Slika 2: Diagram delovanja java agenta na preprostem primeru

Java agenti so posebna vrsta razreda, ki lahko z Instrumentation API-jem modificirajo bajtno kodo aplikacije v JVM. Njegov najpomembnejši del je implementacija vmesnika `ClassFileTransformer`. Agenta lahko ustvarimo na več načinov.

- **Statični agent**, ki ga zgradimo in zapakiramo v jar datoteko in ga nato povežemo z aplikacijo ob zagonu. Razred agenta mora vsebovati `premain(String args, Instrumentation instr)` funkcijo, ki se bo izvedla takoj po inicializaciji JVM

  ```
  $ java -javaagent:<pod do jar datoteke agenta> -jar <pot do jar datoteke za prestrežbo>
  ```

- **Dinmični agent**, ki se poveže na obstoječi že zagnani JVM

4. Viri

https://en.wikipedia.org/wiki/Profiling_(computer_programming)
https://en.wikipedia.org/wiki/Binary_translation
http://developforperformance.com/PerformanceProfilingWithAFocus.html
https://course.ece.cmu.edu/~ece548/tools/atom/man/wrl_94_2.pdf
Janjusic, T. & Krishna, K. (2016). Chapter Three - Hardware and Application Profiling Tools *Advances in Computers* , *92*, 105-160. https://doi.org/10.1016/B978-0-12-420232-0.00003-9
Primer agenta Javi: https://github.com/fredang/agent-metric-example