

Verifikacija dinamičke detekcije trke

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Marko Crnobrnja
mi12024@alas.matf.bg.ac.rs

26. april 2018.

Sažetak

Česta greška višenitnih programa sa konkurentnim izvršavanjem su trrke za resursima, stoga su razvijeni algoritmi koji automatski ispituju prisustvo ovog problema. Ovaj rad obrađuje formalnu verifikaciju ispravnosti jednog takvog algoritma. Ova proverava obuhvata dokaz tačnosti i potpunosti samog algoritma kao i njegove implementacije.

Sadržaj

1	Uvod	2
2	Algoritam	2
2.1	Algoritam vektorskog sata	2
2.2	FASTTRACK algoritam	3
3	Implementacija	4
4	Zaključak	5
	Literatura	5

1 Uvod ¹

Trka za resursima (nadalje **trka**) je pojava do koje dolazi kada tokom konkurentnog izvršavanja dve niti potencijalno pristupaju istoj memorij-skoj lokaciji istovremeno. Trka može dovesti do nedefinisanog ponašanja programa i nedoslednosti podataka. Kako bi se ispitala ispravnost višenitnih programa, neophodno je utvrditi da ne može doći do trke tokom izvršavanja koda na potpuno automatizovan način koji nije podložan ljudskoj grešci. Radi toga razvijeno je više programa sa algoritmima za verifikaciju od koji će jedan (FASTTRACK) biti razmatran i dokaz njegove ispravnosti prikazan.

2 Algoritam

Obradeni algoritam, FASTTRACK [?], dinamički je algoritam što znači da je zasnovan na istovremenom izvršavanju sa ispitivanim kodom pomoću instrumentalizacije, odnosno izmene izvornog koda dodavanjem algoritma.

Kako bi dokazali ispravnost ovog, počecemo sa jednostavnijim, takozvanim algoritmom vektorskog sata, čija se ispravnost lakše dokazuje, a koji zatim proširujemo u FASTTRACK i dokazujemo njihovu ekvivalentnost.

2.1 Algoritam vektorskog sata

Kako bi detektovali trke, potrebno je ustanoviti vremenski poredak između izvršenih naredbi u višenitnom programu. Da bi čuvali ove podatke, definišemo strukturu **vektorskog sata**: niza V koji čuva po jedan nenegativan ceo broj po niti t što zapisujemo $V(t)$.

Nad ovom strukturom definišemo i dve operacije i relaciju:

1. Unija:

$$\forall t V_1 \sqcup V_2(t) = \max(V_1(t), V_2(t))$$

2. Inkrement:

$$\forall t \text{ inc}_k(V)(t) = \begin{cases} V(t) + 1, & \text{ako } t = a \\ V(t), & \text{inače} \end{cases}$$

3. Poređenje:

$$V_a \sqsubseteq V_b \iff \forall t V_a(t) \leq V_b(t)$$

Algoritam čuva četiri skupa vektorskih satova:

1. Satove C_t gde je t indeks niti koji predstavljaju poslednje momente koji su prethodili trenutnom vremenu niti t za sve niti.
2. Satove L_m gde je m broj muteksa koji čuvaju poslednje vreme za svaku nit koje prethodi poslednjem otpuštanju muteksa m
3. Satove R_x gde je x lokacija koji označavaju najskorije vreme kada je data nit zadnji put čitala sa lokacije x .

¹Ovaj rad je neformalna obrada [1].

4. Satove W_x gde je x lokacija koji znače vremena poslednjeg upisa na lokaciju x od strane date niti.

Celokupno stanje izvršavanja algoritma zapisujemo kao $(C, L, R, W) = S$ nadd kojim definišemo semantiku prelaza $S \Rightarrow S'$ koja je definiisana na slici 1. Ukoliko operacija programa ne odgovara nijednom od dozvoljenih prelaza znamo da je potencijalno došlo do istovremene upotrebe memorijske lokacije i trka je detektovana.

$$\begin{array}{l}
\text{READSAMEEPOCH} \frac{\mathcal{R}_x = E(t)}{(C, \mathcal{L}, \mathcal{R}, W) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}', W)} \\
\text{READEXCLUSIVE} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Epoch} \\ \mathcal{R}_x \preceq C_t \quad W_x \preceq C_t \\ \mathcal{R}' = \mathcal{R}[x := E(t)] \end{array}}{(C, \mathcal{L}, \mathcal{R}, W) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}', W)} \\
\text{READSHARE} \frac{\begin{array}{c} W_x \preceq C_t \\ V = \perp_V[t := C_t(t), u := c] \\ \mathcal{R}' = \mathcal{R}[x := V] \end{array}}{(C, \mathcal{L}, \mathcal{R}, W) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}', W)} \\
\text{READSHARED} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Vector Clock} \\ W_x \preceq C_t \\ \mathcal{R}' = \mathcal{R}[x := \mathcal{R}_x[t := C_t(t)]] \end{array}}{(C, \mathcal{L}, \mathcal{R}, W) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}', W)} \\
\text{WRITESAMEEPOCH} \frac{W_x = E(t)}{(C, \mathcal{L}, \mathcal{R}, W) \xrightarrow{wr(t,x)} (C, \mathcal{L}, \mathcal{R}, W)} \\
\text{WRITEEXCLUSIVE} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Epoch} \\ \mathcal{R}_x \preceq C_t \quad W_x \preceq C_t \\ W' = W[x := E(t)] \end{array}}{(C, \mathcal{L}, \mathcal{R}, W) \xrightarrow{wr(t,x)} (C, \mathcal{L}, \mathcal{R}, W')} \\
\text{WRITESHARED} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Vector Clock} \\ \mathcal{R}_x \sqsubseteq C_t \quad W_x \preceq C_t \\ W' = W[x := E(t)] \\ \mathcal{R}' = \mathcal{R}[x := \perp_e] \end{array}}{(C, \mathcal{L}, \mathcal{R}, W) \xrightarrow{wr(t,x)} (C, \mathcal{L}, \mathcal{R}', W')}
\end{array}$$

Slika 1: Pravila prelaska.

Dokaz ispravnosti ovog algoritma (koji je ovde preskočen) sledi iz korespondencije između relacije poretka \sqsubseteq nad satovima i vremenskog redosleda izvršavanja instrukcija u predstavljenom programu.

2.2 FASTTRACK algoritam

Kako je primećeno da su svi upisi u kodu bez trke potpuno uređeni, odnosno da su svi ili prvi ili imaju prethodnika, predloženo je poboljšanje algoritma takvo da se čuva samo vreme i indeks niti poslednjeg upisa na lokaciju umesto upisa za sve niti. Kako bi se ovo implementiralo uvodi se nova struktura, **epoha**, koja je se može shvatiti kao poseban slučaj sata oblika:

$$(c@t)(u) = \begin{cases} c & \text{ako } t = u \\ 0 & \text{inače} \end{cases}$$

Poređenje između epoha ili između epohe i sata pišemo $c@t \preceq V$, sa značenjem analognim poređenju satova. Minimalnu epohu $0@t_0$ pišemo \perp_e . Često upotrebljavanu vrednost $C_t(t)@t$ pišemo kao $E(t)$.

Izvršavanje ovog algoritma teče slično kao prethodnog, sa n-torkom (C, L, R, W) osim što je u ovom slučaju R_x epoha ili sat a W_x uvek epoha. Pravila prelazaka između stanja, prikazana na slici 2, nešto su složenija kako bi se obuhvatili slučajevi kad je R_x sat odnosno epoha.

Dokaz ispravnosti ovog algoritma svodi se na dokazivanje ekvivalencije sa prvobitnim algoritmom, odnosno postojanje relacije bisimulacije između njihovih stanja tako da epohe i satovi FASTTRACK algoritma koji se odnose na čitanje i pisanje predstavljaju gornju granicu odgovarajućih satova osnovnog algoritma na način koji garantuje da se drugi algoritam

$$\begin{array}{c}
\text{READSAMEEPOCH} \frac{\mathcal{R}_x = E(t)}{(C, \mathcal{L}, \mathcal{R}, \mathcal{W}) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}, \mathcal{W})} \\
\\
\text{READEXCLUSIVE} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Epoch} \\ \mathcal{R}_x \preceq C_t \quad \mathcal{W}_x \preceq C_t \\ \mathcal{R}' = \mathcal{R}[x := E(t)] \end{array}}{(C, \mathcal{L}, \mathcal{R}, \mathcal{W}) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}', \mathcal{W})} \\
\\
\text{READSHARE} \frac{\begin{array}{c} \mathcal{W}_x \preceq C_t \\ \mathcal{R}_x = c \otimes u \\ V = \perp_V[t := C_t(t), u := c] \\ \mathcal{R}' = \mathcal{R}[x := V] \end{array}}{(C, \mathcal{L}, \mathcal{R}, \mathcal{W}) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}', \mathcal{W})} \\
\\
\text{READSHARED} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Vector Clock} \\ \mathcal{W}_x \preceq C_t \\ \mathcal{R}' = \mathcal{R}[x := \mathcal{R}_x[t := C_t(t)]] \end{array}}{(C, \mathcal{L}, \mathcal{R}, \mathcal{W}) \xrightarrow{rd(t,x)} (C, \mathcal{L}, \mathcal{R}', \mathcal{W})} \\
\\
\text{WRITESAMEEPOCH} \frac{\mathcal{W}_x = E(t)}{(C, \mathcal{L}, \mathcal{R}, \mathcal{W}) \xrightarrow{wr(t,x)} (C, \mathcal{L}, \mathcal{R}, \mathcal{W})} \\
\\
\text{WRITEEXCLUSIVE} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Epoch} \\ \mathcal{R}_x \preceq C_t \quad \mathcal{W}_x \preceq C_t \\ \mathcal{W}' = \mathcal{W}[x := E(t)] \end{array}}{(C, \mathcal{L}, \mathcal{R}, \mathcal{W}) \xrightarrow{wr(t,x)} (C, \mathcal{L}, \mathcal{R}, \mathcal{W}')} \\
\\
\text{WRITESHARED} \frac{\begin{array}{c} \mathcal{R}_x \in \text{Vector Clock} \\ \mathcal{R}_x \sqsubseteq C_t \quad \mathcal{W}_x \preceq C_t \\ \mathcal{W}' = \mathcal{W}[x := E(t)] \\ \mathcal{R}' = \mathcal{R}[x := \perp_e] \end{array}}{(C, \mathcal{L}, \mathcal{R}, \mathcal{W}) \xrightarrow{wr(t,x)} (C, \mathcal{L}, \mathcal{R}', \mathcal{W}')}
\end{array}$$

Slika 2: Pravila prelaska za FASTTRACK. FORK, JOIN, ACQUIRE i RELEASE identični su kao u osnovnom algoritmu.

zaglavljuje samo kada i prvi i obratno. Većina dokaza (ovde izostavljenog, kao i definicije, radi sažetosti) odnosi se na pokazivanje očuvanja ove ekvivalencije prilikom prelazaka.

3 Implementacija

Kako bi se uverili u ispravnost detekcije trka ovim putem, potrebno je verifikovati ne samo algoritam već i njegovu implementaciju, odnosno instrumentalizaciju koda nad kojim operiše. Da bi se ovo postiglo potrebno je definisati semantiku programskog jezika na kome je pisan instrumentalizovan kod, zatim dodati instrukcije koje odgovaraju ažuriranjima satova i epoha pri prelascima stanja algoritma prilikom svake instrukcije na koju se ti prelasci odnose (operacije sa muteksima, nitima, pisanje i čitanje na deljenu memorijsku lokaciju).

Problem koji se javlja pritom je prisustvo trka u kodu algoritma koje se javljaju upravo kada i u originalnom kodu. Zato je potrebno uvesti dodatne mutekse za svaku memorijsku lokaciju koji će zaštititi instrumentalizaciju.

Ispravnost implementacije se dokazuje ekvivalencijom (bisimulacijom) između prvobitnog i instrumentalizovanog programa. Ovo uključuje posmatranje instrumentalizovanog programa u "velikim koracima" odnosno kao sačinjenog od atomičnih operacija koje uključuju prvobitne instrukcije i instrumentalizaciju koja im odgovara kao i dokaz da je ovakva semantika ekvivalentna prvobitnoj semantici pojedinačnih instrukcija.

4 Zaključak

Verifikacija softvera morala bi podrazumevati ne samo garanciju ispravnosti koju daju korišćeni alati već i garanciju za te same alate. U jednostavnim slučajevima ova bi možda mogla biti data ručnim dokazom ili čak neformalno, ali za složenije algoritme potrebno je formalizovati njihovo izvršavanje kako bi se podvrglo mašinskoj proveru. Svi postupci opisani ovde mogu se izraziti precizno i biti provereni automatskim dokazivačima.

Literatura

- [1] William Mansky, Yuanfeng Peng, Steve Zdancewic, and Joseph De-vietti. Verifying dynamic race detection. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 151–163. ACM, 2017.