DISTRIBUIRANI ALGORITMI I SISTEMI

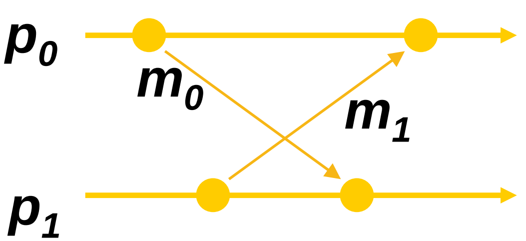
Iz kursa CSCE 668
Proleće 2014

Autor izvorne prezentacije:
Prof. Jennifer Welch

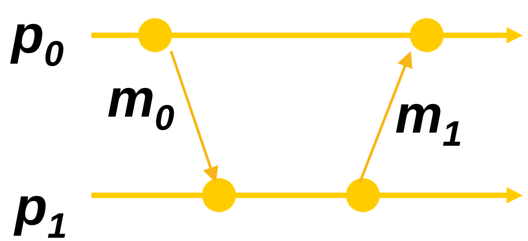
Motivacija za logičke satove

- U asinhronim sistemima, često ne možemo orediti koji od dva događaja se desio pre onog drugog:

Primer A



Primer B



U primeru A, procesori neznaju koja poruka je prva poslata. Verovatno to nije važno.

U primeru B, procesori *znaju* koja poruka je prva poslata. Može biti važno.

Pokušajmo da oderimo relativni redosled *nekih* (ne svih) događaja.

Delimičan redosled „desio se pre“

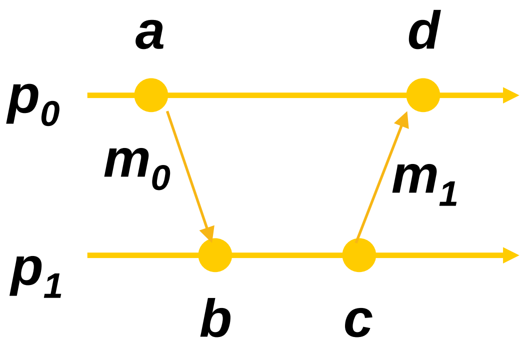
- U datom izvršenju, događaj računanja a **desio se pre** događaja računanja b, što se označava sa:

  $a \rightarrow b$, ako

1. a i b se desili na istom procesoru i a prethodi b, ili
2. a je rezultat slanja m i b uključuje prijem m, ili
3. postoji događaj računanja c takav da $a \rightarrow$ c i c $\rightarrow$ b (tranzitivna zatvorenost)

Delimičan redosled „desio se pre“

- Desio se pre znači da informacija može teći od *a* do *b*, tj., da *a* može uzrokovati *b*.



$a \to b$

$b \to c$

$c \to d$

$a \to c$

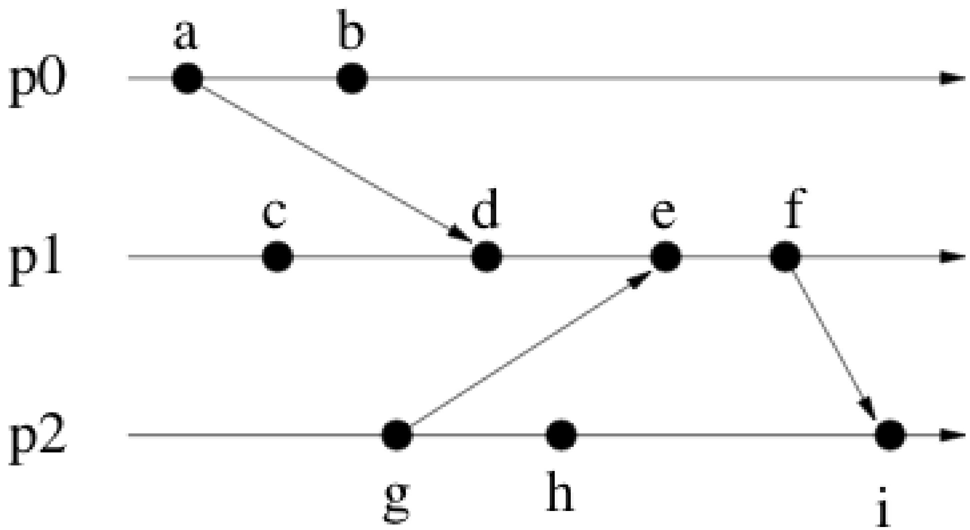
$a \to d$

$b \to d$

Konkurentni događaji

- Ako se *a* nije sedio pre *b*, i *b* se nije desio pre *a*, onda su *a* and *b* **konkurentni**, oznaka *a* || *b*.

Primer relacija desio se pre



Pravilo 1: $a \rightarrow b$, $c \rightarrow d \rightarrow e \rightarrow f$, $g \rightarrow h$, $h \rightarrow i$

Pravilo 2: $a \rightarrow d$, $g \rightarrow e$, $f \rightarrow i$

Pravilo 3: $a \rightarrow e$, $c \rightarrow i$, …

$h \mid\mid e$, …

Logički satovi

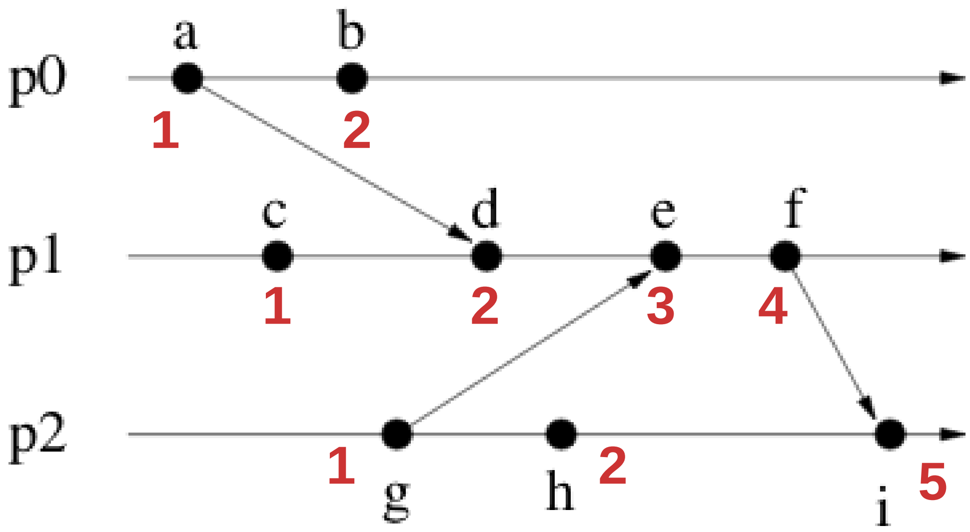
- **Logički satovi** su vrednosti dodeljene događajima da bi se obezbedila info. o redosledu događaja.
- Ceo broj $L(e)$ se dodeljuje svakom događaju računanja e u nekom izvršenju tako da

  ako $a \rightarrow b$, onda $L(a) < L(b)$.

Algoritam logičkih vremenskih pečata (Timestamps)

- Svaki p_i održava brojač (**logički vrem. pečat**) L_i, inicijalno 0
- Svaka poruka koju p_i šalje je pečatirana sa tekućom vrednošu od L_i
- p_i inkrementira L_i u svakom koraku tako da bude veći od
  - njegove tekuće vrednosti, i
  - vrem. pečata on svih poruka primljenih u tom koraku
- Ako je *a* događaj u p_i, onda se $L(a)$-u dodeljuje vrednost od L_i na kraju *a*.

Primer logičkih vrem. pečata



$a \rightarrow b : L(a) = 1 < 2 = L(b)$

$f \rightarrow i : L(f) = 4 < 5 = L(i)$

$a \rightarrow e : L(a) = 1 < 3 = L(e)$

itd.

Dobijanje totalnog redosleda

- Ako je potreban totalni redosled, ID procesora se mogu koristi za razlikovanje pečata (break ties).
- U primeru, $L(a) = (1,0)$, $L(c) = (1,1)$, itd.
- Vrem. pečati su uređeni leksikografski.
- U primeru, $L(a) < L(c)$.

Nedostaci logičkih satova

- $a \rightarrow b$ implicira $L(a) < L(b)$, ali $L(a) < L(b)$ ne mora da implicira $a \rightarrow b$.
- U predhodnom primeru, $L(g) = 1$ i $L(b) = 2$, ali g se nije desio pre b.
- Razlog je što „desio se pre" daje delimičan redosled, dok su vrednosti logičkih satova celi br., koji su totalno uređeni.

Vektorski satovi

- Generalizuju logičke satove tako da obezbede uzročnu i neuzročnu informaciju.
- Koriste se vrednosti iz delimično uređenog skupa umesto iz totalno uređenog skupa.
- Vrednost $V(e)$ se dodeli svakom događaju računanja e u nekom izvršenju tako da
  $a \rightarrow b$ ako i samo ako $V(a) < V(b)$.

Algoritam vektorskih vremenskih pečata

- Svaki p_i održava n-vektor V_i, inicijalno svi el. 0
- Elem j u V_i je p_i-jeva procena koliko koraka je p_j izveo
- Svaka poruka koju p_i šalje se pečatira sa tekućom vrednosti od V_i
- U svakom koraku, $V_i[i]$ se povećava za 1
- Po prijemu poruke sa vektorskim pečatom T, ažurira se svaki elem $j \neq i$ od V_i tako da

  $V_i[j] = max(T[j], V_i[j])$
- Ako je a događaj u p_i, onda postavi $V(a)$ da ima vrednost V_i na kraju a.

Rukovanje vektorskim vremenskim pečatima

Neka su V i W dva n-vektora celih brojeva.

Jednakost: $V = W$ iff $V[i] = W[i]$ za svako i.

Primer: $(3,2,4) = (3,2,4)$

Manje ili jednako: $V \leq W$ iff $V[i] \leq W[i]$ za svako i.

Primer: $(2,2,3) \leq (3,2,4)$ i $(3,2,4) \leq (3,2,4)$

Manje od: $V < W$ iff $V \leq W$ ali $V \neq W$.

Primer: $(2,2,3) < (3,2,4)$

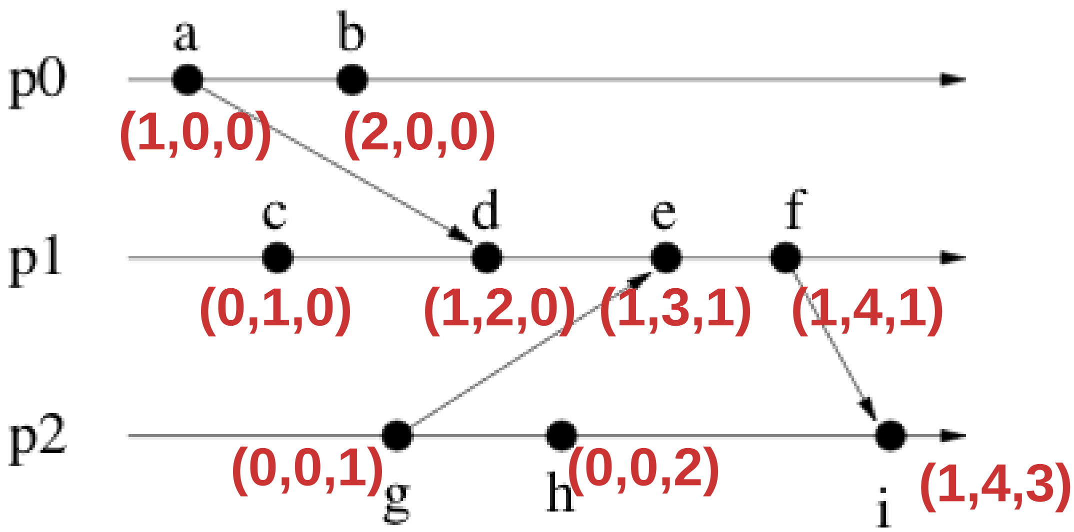
Neuporediv: $V \,||\, W$ iff $!(V \leq W)$ i $!(W \leq V)$.

Primer: $(3,2,4) \,||\, (4,1,4)$

Rukovanje vektorskim vremenskim pečatima

- Ovako def. delimično uređenje **nije** isto kao leksikografsko uređenje.
- Leksikografsko uređenje je totalno uređenje vektora.
- Uporedi (3,2,4) sa (4,1,4) u ova dva pristupa.

Primer vektorskih vrem. pečata



V(g) = *(0,0,1)* i *V(b)* = *(2,0,0)*, koji su neuporedivi.
Uporedi sa logičkim satovima *L(g)* = *1* i *L(b)* = *2*.

Korektnost vektorskih vrem. pečata

Teoreme (4.5 & 4.6): Vektorski vrem. pečati implementiraju vektorske satove.

Dokaz: Prvo, pokažimo: $a \rightarrow b$ implicira $V(a) < V(b)$.

Slučaj 1: a i b se desili u p_i, i to a prvi. Pošto se V_i povećava za svaki korak, $V(a) < V(b)$.

Korektnost vektorskih vrem. pečata

Slučaj 2: *a* se desi u p_i i uzrokuje slanje *m*, dok se *b* desi u p_j i uključuje prijem *m*.

- Tokom *b*, p_j ažurira svoj vektor vrem. pečat tako da $V(a) \leq V(b)$.
- p_i-jeva procena broja koraka koje je izveo p_j nikada nije prevelika. Pošto se *m* ne može primiti pre nego je poslata, procenjen br. koraka je uvek manji ili jednak od stvarnog broja. Tj. $V(a)[j] < V(b)[j]$.
- Sledi da je $V(a) < V(b)$.

Korektnost vektorskih vrem. pečata

Slučaj 3: Postoji c takav da $a \rightarrow$ c i c $\rightarrow b$.

Na osnovu indukcije (iz Slučajeva 1 i 2) i tranzitivnosti relacije <, $V(a) < V(b)$.

Zatim, pokažimo: $V(a) < V(b)$ implicira $a \rightarrow b$.

Ekvivalentno sa: $!(a \rightarrow b)$ implicira $!(V(a) < V(b))$.

Korektnost vektorskih vrem. pečata

- Predpostavimo: *a* se desio u p_i, *b* se desio u p_j, i *a* se nije desio pre *b*.
- Neka je $V(a)[i] = k$.
- Pošto se *a* nije desio pre *b*, ne postoji lanac poruka od p_i do p_j sa izvorištem u p_i-jevom *k*-tom koraku, ili kasnije, i krajem u p_j pre *b*.
- Sledi $V(b)[i] < k$.
- Sledi $!(V(a) < V(b))$.

Veličina vektorskih vrem. pečata

- Vektorski vrem. pečati su veliki:
  - *n* elemenata u svakom od njih
  - vred. elemenata rastu bez ograničenja
- Da li postoji efikasniji način implementacije vektorskih satova?
- Odgovor je NE, barem pod nekim uslovima.

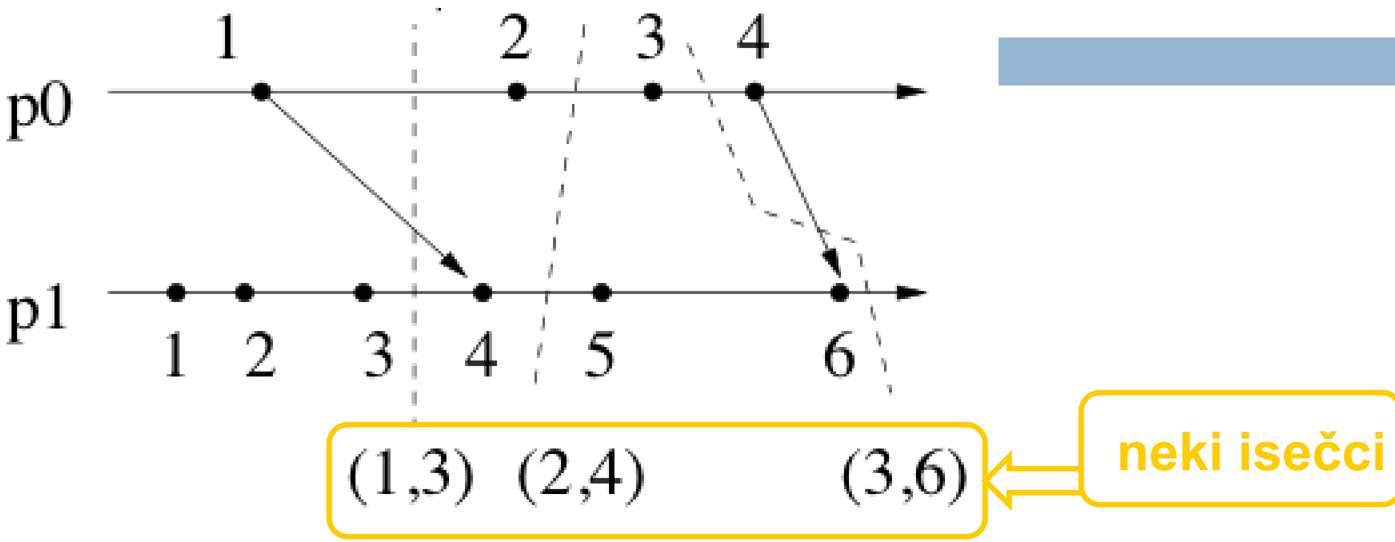
Donja granica veličine vekt. sata

Teorema (4.7): bilo koja izvedba vektorskog sata pomoću vektora realnih brojeva zahteva vektore dužine *n* (broj procesora).

Primena uzročnosti: Konzistentni isečci

- Razmotrimo sistem sa asinhronim slanjem poruka sa:
  - FIFO isporukom poruka po kanalu
  - prijemom najviše 1 por po koraku računanja
- Brojmo korake računanja za svaki procesor 1,2,3,…
- **Isečak** nekog izvršenja je $K = (k_0,\ldots,k_{n-1})$, gde k_i indicira broj koraka računanja koje je izveo p_i

Konzistentni isečci



U **konzistentnom** isečku $K = (k_0,\ldots,k_{n-1})$, ako se korak s na p_j desio pre koraka k_i na p_i, onda je $s \leq k_j$.

(1,3) i (2,4) su konzistentni.

(3,6) je nekonzistentan: korak 4 na p_0 se desio pre koraka 6 na p_1, ali 4 je veće od 3.

Pronalaženje skorašnjeg konzistentnog isečka

Problem, Verzija 1: Svim procesorima je dat isečak K i oni treba da odrede max konz. isečak koji je $\leq K$.

Aplikacija: Oporavak zasnovan na log datoteci.

- Procesori periodično zapisuju svoje stanje u stalnu mem.
- Kada se proc oporavlja od ispada, on proba da uđe u poslednje zapisano stanje, ali mora da se koordinira sa drugim procesorima

Rešenje sa vektorskim satovima

- Implementirati vekt. satove koristeći vekt. vrem. pečate koji se dodaju u aplikacionim porukama.
- Sačuvati vekt. sat za svaki korak računanja u lokalni niz store[1,…]
- Kada se p_i–ju zada ulazni isečak K:

    for x := K[i] down to 1 do

        if store[x] $\leq$ K then return x

    return x (elem za p_i u globalnom odgovoru)

Šta je sa stanjem kanala?

- Stanja procesora nisu dovoljna da snime celokupno stanje sistema.
- Poruke u tranzitu se moraju izračunati.
- Rešenje ovde zahteva
  - dodatno skladište (broj poruka)
  - dodatno računanje u vreme oporavka (uključujući reprodukciju orig. izvršenja da bi se snimile poruke koje su poslate a nisu primljene)

Drugi način uzimanja skorašnjeg konzistentnog stanja

Problem, Verzija 2: Podskup procesora inicira (u proizvoljnim vremenima) traženje konzistentnog isečka sa stanjem barem jednog od inicijatora u trenutku iniciranja.

- Naziva se *distribuirani snimak*.
- Info. za snimak se može skupiti na jednom proc. i tu se može analizirati.

Aplikacija: detekcija završetka

Algoritam sa markerima (Marker)

- Umesto dodavanja ekstra informacije na svaku aplikacionu poruku , umeću se kontrolne poruke („markeri") u kanale.
- Kod za p_i:

inicijalno answer = -1 i num = 0

kada stigne aplikaciona poruka:

    num++; uradi aplikacionu akciju

kada stige marker ili kada se inicira uzimanje snimka:

    if answer = -1 then

        answer := num // p_i-jev deo u konačnom odgovoru

        pošalji marker svim susedima

Šta je sa stanjima kanala?

- p_i zapisuje sekvencu poruka primljenih od p_j između trenutka kada p_i zapiše svoj odgovor i trenutka kada p_i dobije marker od p_j
- Ovo su poruke u tranzitu od p_j do p_i u isečku koji vraća ovaj algoritam.