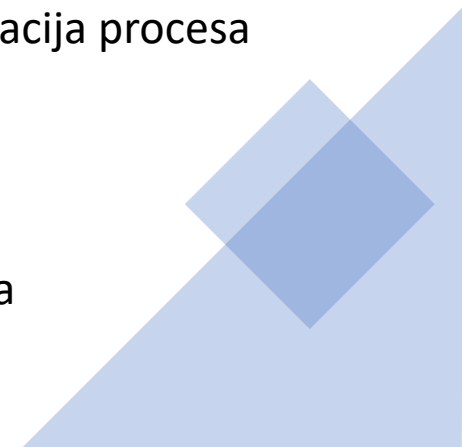


Paralelno i distribuirano računarstvo

Predavanja
Računarstvo u oblaku (Cloud Computing)
nastavnik: Miroslav Zarić



Sadržaj

- Zašto ovde pričamo o paralelnom i distribuiranom računarstvu
 - Paralelno računarstvo
 - Osnovni koncepti
 - Amdalov zakon
 - Izazovi
 - Komunikacija
 - Arhitekture
 - Distribuirani sistemi
 - Osnovni koncept
 - Poželjna svojstva
 - Globalno stanje grupe procesa
 - Komunikacioni protokoli i koordinacija procesa
 - Runs, cuts, i causal history
 - Konkurentnost
 - Atomičnost operacija
 - Protokoli za postizanje konsenzusa
- 

Zašto ovde pričamo o paralelnom i distribuiranom računarstvu

- Računarstvo u oblaku je rezultat višedecenijskog razvoja raznih koncepata i ideja
- Koncepti paralelnih i distribuiranih računarskih sistema su važni za razumevanje izazova koji se postavljaju pred dizajn i korišćenje sistema za računarstvo u oblaku
- Računarstvo u oblaku je blisko povezano sa paralelnim i distribuiranim sistemima
 - Aplikacije u oblaku su bazirane na paradigmi klijent-server gde relativno jednostavan klijentski softver „trči“ na klijentskom računaru, dok se logika aplikacija (*computation*) izvršava „u oblaku“
 - Veliki broj aplikacija u oblaku obavlja operacije koje zahtevaju obradu velike količine podataka (*data intensive*), i koriste veliki broj instanci koje rade konkurentno.
 - Sistemi sa transakcionim procesiranjem, bazirani na web servisima, predstavljaju često korišćen tip aplikacija u oblaku, a one se često izvršavaju na više instanci koje koordinisano rade zahvaljujući pouzdanom slanju poruka i isporuci poruka „po redosledu“

Zašto ovde pričamo o paralelnom i distribuiranom računarstvu (nastavak)

- Koncepti koji se vezuju za paralelne i distribuirane sisteme imaju veliki praktični značaj i za razumevanje računarstva u oblaku
 - Komunikacioni protokoli
 - Koncept „konzistentnog preseka“ (*consistent cuts*) i distribuiranih “zamrznutih” snimaka (*snapshots*) – obezbeđuje mehanizam “kontrolnih tačaka” (*checkpoints*) i restartovanja aplikacija
 - Upotreba monitora – mnoge funkcije *clouda* počivaju na upotrebi sistemskih komponenti koje prikupljaju i omogućavaju uvid u stanje pojedinačnih sistema

Paralelno računarstvo – osnovni koncepti

- I u prirodi, sposobnost paralelnog rada u grupi je veoma efikasan način da se postigne zajednički cilj (često i brže)
- Ideja da se individualni sistemi organizuju da koordinirano rade na rešavanju kompleksnih zadataka pojavila se dosta rano i u računarstvu.
- Paralelno računarstvo omogućava da se kompleksni problemi rešavaju njihovim razbijanjem na jednostavnije, koji se zatim konkurentno (paralelno) rešavaju.
- Dugo godina je paralelizacija bila „sveti gral“ za rešavanje zadataka koji su zahtevali kompleksnu obradu velikih količina podataka (*data intensive*).
- Paralelno računarstvo podstaklo je značajne napretke u više oblasti – algoritmi, arhitektura računara, mreže...
- Hardver i softver za paralelno izvršavanje omogućava da se reše problemi koji zahtevaju više resursa nego što bilo koji pojedinačni sistem može da obezbedi, a istovremeno omogućavaju vršu obradu

Paralelno računarstvo – ubrzanje izvršavanja

- Ubrzanje koje se postiže paralelizacijom izvršavanja može se izraziti kao

$$S(N) = T(1) / T(N)$$

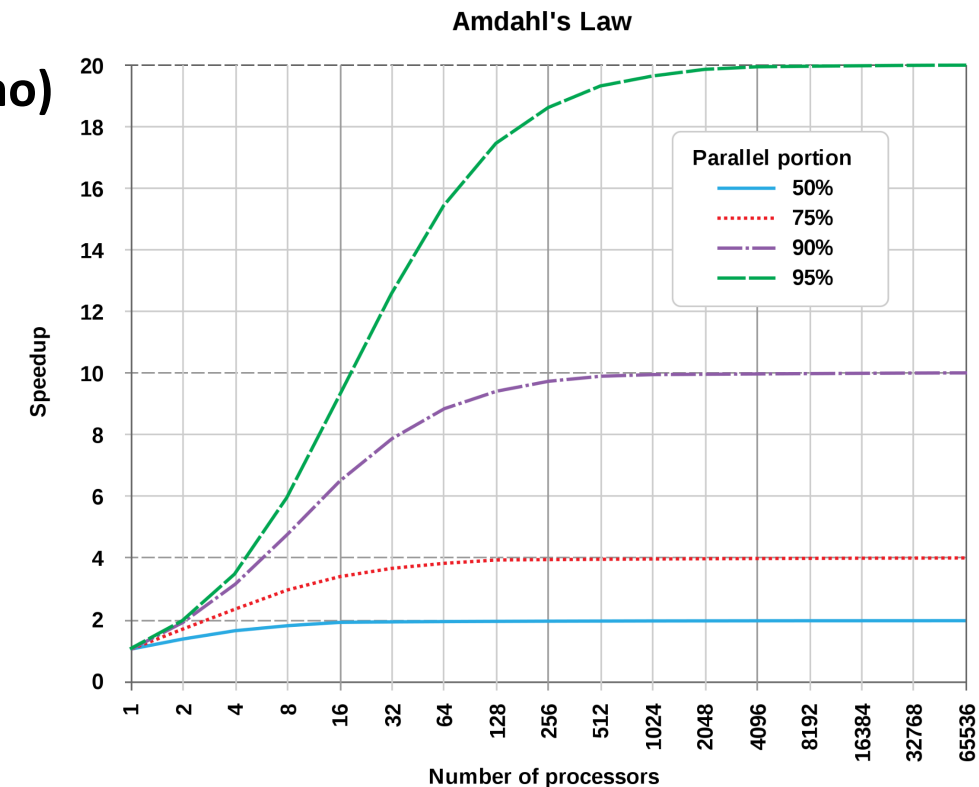
- Gde je

$T(1)$ – vreme potrebno serijskom algoritmu da izvrši obradu

$T(N)$ – vreme potrebno paralelizovanom algoritmu da izvrši obradu na N instanci

Paralelno računarstvo – Amdalov zakon

- Amdalov zakon (Amdahl's Law) definiše teoretski moguće ubrzanje koje se može dobiti paralelizacijom
 - "opšte ubrzanje koje se može dobiti optimizacijom jednog dela sistema ograničeno je delom ukupnog vremena u kome se optimizovani segment zaista koristi u samom sistemu"
- Ako je α deo vremena koje program provodi u segmentu koda koji **nije paralelizovan (ne može se obavljati konkurentno)** onda je teoretski limit za ubrzanje
$$S = 1 / \alpha$$
- Često se koristi u paralelnom računarstvu da se predvidi teoretsko ubrzanje kada se koriste multiprocesorski sistemi
- Primenjiv je kod problema „fiksne“ veličine – gde se količina posla koji svaki paralelizovan process smanjuje sa povećanjem broja procesa i svi procesi dobijaju istu količinu posla



Paralelno računarstvo – izazovi

- Koordinacija konkurentnog izvršavanja je složen problem
- Zadaci u paralelizovanom programi su posebni procesi, a manji zadaci su često niti
 - potreban je (nekad dosta veliki) *overhead* za sinhronizaciju, što ograničava učinak paralelizacije
 - *barrier synchronization* - paralelno izvršavanje se često radi po fazama (*stages*) – sve konkurentne aktivnosti moraju sačekati da se završi jedna faza pre nego se pređe na sledeću gde se ponovo posao paralelizuje
 - *race conditions* – neželjeni efekat kada ukupni rezultat paralelne obrade postane zavisn od redosleda događaja u sistemu
 - *deadlock* – situacija u kojoj se procesi ili niti takmiče za zauzimanje resursa i pri tome svaki od njih uspe da zauzme deo potrebnih resurse, ali je prinuđen da čeka na druge, koje je zaključao drugi proces/nit koji takođe nije uspeo da ostvari lock nad svim resursima koji su mu neophodni i nijedan proces/nit više nije u stanju da nastavi.

Paralelno računarstvo – izazovi (nastavak)

- *deadlock* (nastavak) – Coffmanovi kriterijumi za upadanje u *deadlock* (svi moraju da se dese istovremeno):
 - Međusobno isključivanje (*mutual exclusion*) – barem jedan resurs mora biti nedeljiv
 - Držanje i čekanje (*hold and wait*) – barem jedan proces mora zadržati resurs, a čekati na neki drugi
 - Nema prinudnog oslobađanja resursa (no preemption) – scheduler ili monitor nema mogućnost da prisli proces/thread da oslobodi zaključan resurs
 - Cirkularno čekanje – ako postoji skup procesa/niti {P1, P2, P3, ... Pn} – postoji kružna uslovljenost koji proces čeka na koji resurs P1 čeka na resurse koje drže P2 i P3, a Pn čeka na resurs koji drži P1
- *livelock* - kada dva ili više procesa stalno menjaju svoje stanje i uslovljavaju druge da posledično promene svoje stanje kao posledicu te promene
- *priority inversion* – procesi imaju prioritete pri raspoređivanju, problem kada proces manjeg prioriteta izbací onaj višeg iz izvršavanja

Paralelno računarstvo – komunikacija

- Procesi/niti mogu komunicirati razmenom poruka ili preko deljenih memorijskih lokacija
- Procesori s više jezgri ponekad koriste deljene memorijske lokacije, ali moderni „superračunari“ retko koriste ovaj princip jer se teško skalira
 - sistemski softver često koristi deljenu memoriju (stek, tabele procesora i jezgri za upravljanje raspoređivanjem, tabele procesa/niti, tabele stranica za upravljanje virtuelnom memorijom...)
 - debugovanje je kompleksno
- Prosleđivanje poruka
 - tipično koristi u velikim distribuiranim sistemima
 - debugovanje je lakše

Paralelno računarstvo – arhitekture

- Paralelizam se može posmatrati na više nivoa
- Paralelizam na nivou bita (*bit-level parallelism*)
 - 8-bit, 16-bit, 32-bit, 64-bit – broj bita koji se obrađuje po ciklusu procesora. Redukuje broj instrukcija koji se mora obaviti da bi se obradili veći operandi, povećavao se broj adresnih bita a time i adresni prostor (količina memorije) kojim se može upravljati
- Paralelizam na nivou instrukcija (*instruction-level parallelism*)
 - procesni *pipeline*-i sa više faza. RISC – tipično 5 faza (dohvatanje instrukcija, dekodiranje instrukcija, izvršavanje instrukcija, upravljanje pristupom memoriji, *write back*). CISC – veći broj faza.
- Paralelizam na nivou podataka ili ciklusa (*data or loop parallelism*)
 - programske petlje (ciklusi) mogu se izvršavati u paraleli
- Paralelizam zadatka (*task parallelism*)
 - problem se može dekomponovati u međusobno nezavisne zadatke koji se onda mogu obavljati paralelno

Paralelno računarstvo – arhitekture (nastavak)

- Klasifikacija računarskih arhitektura na osnovu broja konkurentnih tokova podataka i instrukcija
- SISD – Single instruction, single data
 - Ovde paralelizma nema
- SIMD - Single instruction, multiple data
 - Vektorsko procesiranje – ista instrukcija može da se izvrši paralelno nad svim komponentama vektora
- MIMD – Multiple instruction, multiple data
 - Sistem sa nekoliko procesora (jezgri) koji funkcionišu asinhrono i nezavisno, različite jezgre mogu izvršavati potpuno različite instrukcije nad različitim podacima u istom trenutku
 - Procesori mogu deliti zajedničku memoriju - Uniform Memory Access (UMA), Cache Only Memory Access (COMA), i Non-Uniform Memory Access (NUMA)
 - Može imati distribuiranu memoriju – tada procesori i memorija komuniciraju preko mrežnih sistema kao što su hypercube, 2D torus, 3D torus, omega mreže...

Distribuirani sistemi – osnovni koncepti

- Distribuirani sistem – kolekcija autonomnih računara koji se povezuju preko mreže i softvera za upravljanje distribucijom (*middleware*) koji omogućava računarima da koordinišu aktivnosti i da dele resurse.
- Korisnik distribuirani sistem doživljava kao jedan integrisani računarski sistem.
- Karakteristike:
 - Komponente su autonomne
 - Raspoređivanje (*scheduling*) i upravljanje resursima se implementira na svakom od sistema
 - Postoji više tačaka upravljanja, ali i više tačaka u kojima može doći do otkaza
 - Resursi u opštem slučaju ne moraju biti dostupni sve vreme
 - Skaliranje se obavlja dodavanjem novih komponenti u sistem
 - Mogu biti tako dizajnirani da obezbede (relativno) visok nivo dostupnosti čak i u uslovima slabe pouzdanosti hardvera/softvera/mreže

Distribuirani sistemi – osnovni koncepti (nastavak)

- Koncept postoji već decenijama
 - Distribuirani fajl sistemi
 - RPC – remote procedure call – interprocesna komunikacija koja omogućava pozivanje procedura koje se nalaze u drugom adresnom prostoru, pa i na drugom udaljenom računaru

Distribuirani sistemi – poželjna svojstva

- Transparentan pristup (*Access transparency*)
 - lokalnim i udaljenim informacionim objektima se pristupa putem istih operacija.
- Transparentnost lokacije (*Location transparency*)
 - informacionim objektima se pristupa na takav način da korisnik ne mora znati gde se on nalazi.
- Transparentnost konkurentnog izvršavanja (*Concurrency transparency*)
 - Više procesa može da se izvršava konkurentno, pristupajući deljenim informacionim objektima, a da to ne izazviva međusobno ometanje
- Transparentnost replikacije (*Replication transparency*)
 - Koriste se više instanci informacionih objekata kako bi se povećala pouzdanost, a da toga ne moraju biti svesni ni korisnici ni aplikacije

Distribuirani sistemi – poželjna svojstva (nastavak)

- Transparentnost pojave otkaza (*Failure transparency*)
 - Otkazi komponenti su skriveni od korisnika
- Transparentnost premeštanja (*Migration transparency*)
 - Informacioni objekti mogu biti premešteni na drugu lokaciju u sistemu a da to ne utiče na operacije nad njima.
- Transparentnost performansi (*Performance transparency*)
 - Sistem se može rekonfigurisati na osnovu trenutnog opterećenja i u skladu sa zahtevima za kvalitet servisa.
- Transparentnost skaliranja (*Scaling transparency*)
 - Sistem i aplikacije se mogu skalirati bez izmene strukture sistema ili ometanja rada aplikacija.

Distribuirani sistemi – globalno stanje grupe procesa

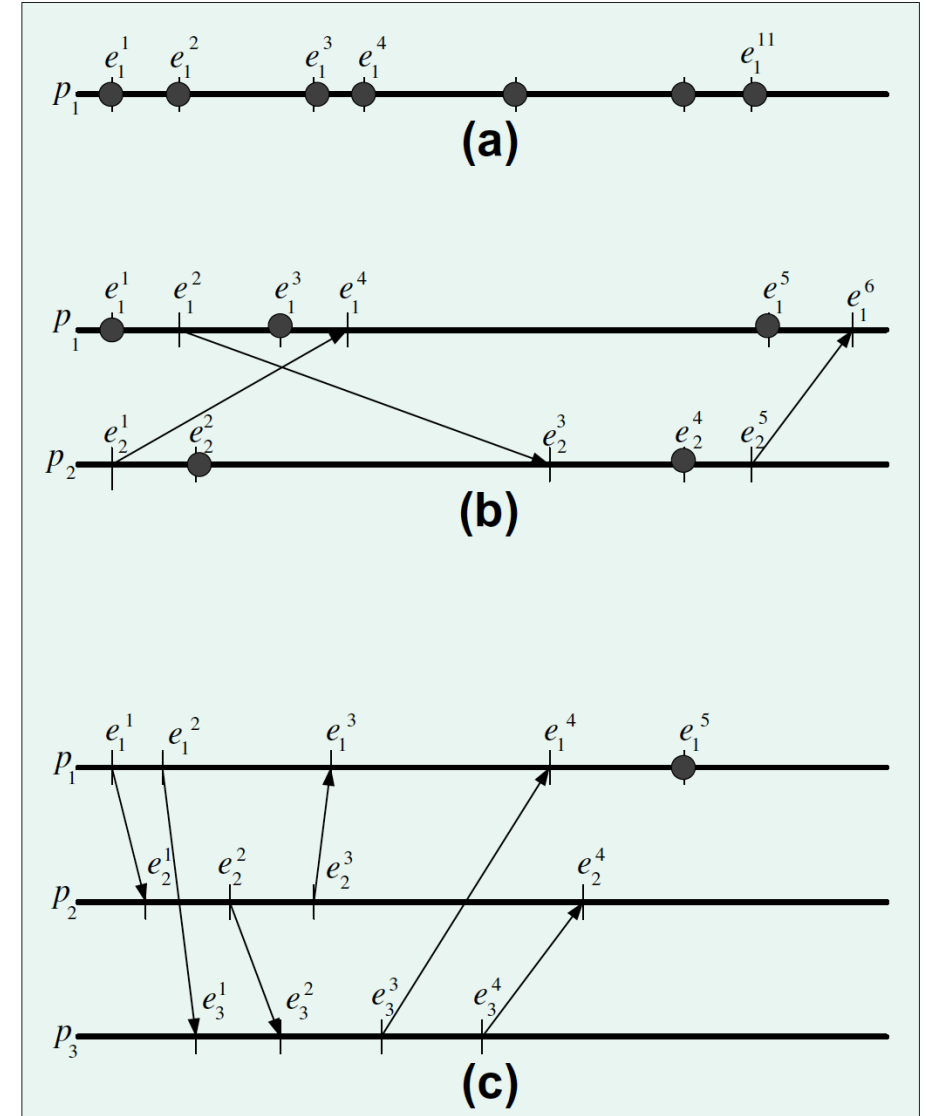
- Za razumevanje distribuiranih sistema može se koristiti model - apstrakcija koji se zasniva na dve ključne komponente: procesi i komunikacioni kanali.
- **Proces** je program koji se izvršava, a niti (*threads*) predstavljaju “manje, lakše” procese. Nit je najmanja procesna jedinica koju operativni sistem može da raspoređuje.
- **Grupa procesa** (*process group*) – predstavlja procese koji sarađuju – koordinisano rade kako bi se postigao određen zajednički cilj
- **Komunikacioni kanal** (*communication channel*) – obezbeđuje mogućnost komunikacije između procesa/niti putem razmene poruka. U najopštijem slučaju možemo smatrati da se komunikacija obavlja putem komunikacionih događaja slanja (*send*) i prijema (*receive*) poruke.
 - Apstrahujemo ga kao jednosmerni kanal za prenos bita, beskonačnog propusnog opsega i nulte latencije, **ali nepouzdan**

Distribuirani sistemi – globalno stanje grupe procesa (2)

- Proces je u nekom momentu okarakterisan svojim **stanjem** (*state*). Stanje predstavlja sveukupnost informacija koje su nam potrebne da(uspešno) restartujemo proces nakon što je bio suspendovan.
- **Događaj** (*event*) je promena stanja procesa.
 - Događaji koji izazivaju promenu stanja procesa p_1 se označavaju sekvencijalno sa $e_i^1, e_i^2, e_i^3 \dots$
 - Neposredno nakon pojave događaja e_i^j , proces (p_1) dospeva u jedno od svojih mogućih stanja σ_i^j , u kome i ostaje dok se ne pojavi događaj e_i^{j+1}
- **Stanje komunikacionog kanala** – Može se definisati na sledeći način: Za dva posmatrana procesa p_i i p_j , stanje komunikacionog kanala ξ_i^j od p_i ka p_j sastoji se od poruka koje je p_i poslao, a p_j još nije primio.

Distribuirani sistemi – globalno stanje grupe procesa (3)

- Pojašnjenje slike:
 - U procesu p_1 svi događaji su lokalni
 - Komunikacija dva procesa: događaj e_1^2 je komunikacioni događaj kojim p_1 šalje poruku p_2 . Događaj e_2^3 reprezentuje komunikacioni događaj prijema poruke
 - Komunikacija tri procesa preko komunikacionih događaja



Distribuirani sistemi – globalno stanje grupe procesa (4)

- Apstrakcije procesa i komunikacionog kanala omogućavaju da se kritična svojstva distribuiranog sistema posmatraju bez ulaska u fizičke detalje pojedinih implementacija i entiteta
- **Protokol** (*protocol*) – konačan skup poruka koje procesi razmenjuju da bi koordinisali svoje aktivnosti
- Globalno stanje grupe procesa predstavlja uniju svih stanja procesa i komunikacionih kanala date grupe
 - Za grupu procesa $p_1, p_2, \dots, p_i, \dots, p_n$ globalno stanje je n-tuple

$$\sum^{(j_1, j_2, \dots, j_n)} = (\sigma_1^{j_1}, \sigma_2^{j_2}, \dots, \sigma_i^{j_i}, \dots, \sigma_n^{j_n})$$

- Stanje kanala u globalnom stanju nije eksplicitno prisutno, već posredno – lokalno stanje pojedinih procesa je uslovljeno obavljenim komunikacijama

Distribuirani sistemi – globalno stanje grupe procesa (5)

- Globalna stanja oformljavaju n dimenzionalu rešetku mogućih stanja grupe procesa
- Komunikacioni događaji su ti koji definišu koja globalna stanja grupa procesa može dostići
- Zanimljiv je i problem koliko različitih putanja u n-dimenzionalnoj rešetci postoji da bi se obavila tranzicija iz jednog u drugo globalno stanje
- Za primer dva procesa za prelazak iz $\sum^{(0,0)}$ u stanje $\sum^{(m,n)}$ taj broj je $N_p^{(m,n)} = \frac{(m+n)!}{m!n!}$
- Za veći broj od q procesa/niti $N_p^{(n_1, n_2, \dots, n_q)} = \frac{(n_1 + n_2 + \dots + n_q)!}{n_1! n_2! \dots n_q!}$
 - Ovo pokazuje da je broj mogućih tranzicija ogroman, što čini debugovanje sistema sa velikim brojem konkurentnih procesa/niti jako komplikovanim

Distribuirani sistemi – komunikacioni protokoli i koordinacija procesa

- Jedan od glavnih problema u paralelnim i distribuiranim sistemima je komunikacija nepouzdanim komunikacionim kanalima – tj. u prisustvu otkaza komunikacionih kanala
- Ukoliko procesi p_1 i p_2 komuniciraju preko kanala koji ima neku verovatnoću otkaza $\varepsilon > 0$, ne postoji protokol koji garantuje da će procesi postići konsenzus ma koliko ε bilo malo
- U praksi mehanizmi otkrivanja i ispravljanja grešaka omogućavaju procesima da komuniciraju pouzdano i preko nepouzdatih kanala

Distribuirani sistemi – komunikacioni protokoli i koordinacija procesa (2)

- Komunikacioni protokoli implementiraju ne samo mehanizme z kontrolu grešaka, već i mehanizme za kontrolu toka i zagušenja komunikacionog kanala.
 - Kontrola toka obezbeđuje povratnu informaciju od primaoca, i dozvoljava da pošiljaca šalje samo onoliko informacija koliko je primalac u stanju da prihvati i obradi.
 - Kontrola zagušenja komunikacionog kanala se brine da količina podataka koja se stavlja na raspolaganje ne prevaziđe kapacitet mreže. Kada je mreža preopterećena, ruteri mogu gubiti pakete, a onda pošiljalac mora da radi retransmisiju paketa. Pošiljalac može korišćenjem RTT da proceni da li je mreža zagušena i smanji brzinu slanja da bi se tome prilagodio.
- Implementacija ovih mehanizama zahteva merenje vremenskih intervala pa je neophodan i koncept *globalog vremena*

Distribuirani sistemi – komunikacioni protokoli i koordinacija procesa (3)

- Svaki proces ima lokalnu istoriju i svaka promena je povezana sa lokalnim vremenom koji obezbeđuju relativno merenje vremena
- Poruke koje procesi šalju jedni drugima mogu se izgubiti ili biti oštećene i bez dodatnih mehanizama zaštite ne postoji način da se obezbedi savršena sinhronizacija lokalnih satova, a ni način da se utvrdi jednoznačan globalni sled događaja koji su nastali u različitim procesima
- U distribuiranim sistemima neophodno je postojanje *globalne saglasnosti o vremenu* kako bi se omogućilo okidanje akcija koje su vremenski uslovljene. Neophodan je konsenzus oko toga *kada se događaj dogodio* – npr. da bi se utvrdilo koji događaj je prethodio nekom drugom - vremenski poredak događaja.
- Da bi se obezbedilo da sistem radi korektno – moramo biti u stanju da utvrdimo da se neki događaj desio PRE nego što se promenilo stanje koje treba da je posledica tog događaja.

Distribuirani sistemi – komunikacioni protokoli i koordinacija procesa (4)

- *Time stamps* se često koriste za određivanje redosleda događaja tako što se uzme globalno vreme na koje se sinhronizuju lokalni satovi.
 - Ali svi lokalni satovi su neprecizni i „driftuju“ tokom vremena
- Ukoliko lokalni satovi ne greše više od π u odnosu na globalno vreme – ovu meru zovemo preciznost
- Neka je g granaularnost fizičkog sata (i ona ne bi trebala biti manja od π)
- Ukoliko imamo dva događaja koja se dešavaju u dva procesa t_b i t_a ,
- Kada je $t_b - t_a \leq \pi + g$ postaje nemoguće utvrditi koji događaj je prethodio onom drugom

Distribuirani sistemi – komunikacioni protokoli i koordinacija procesa (5)

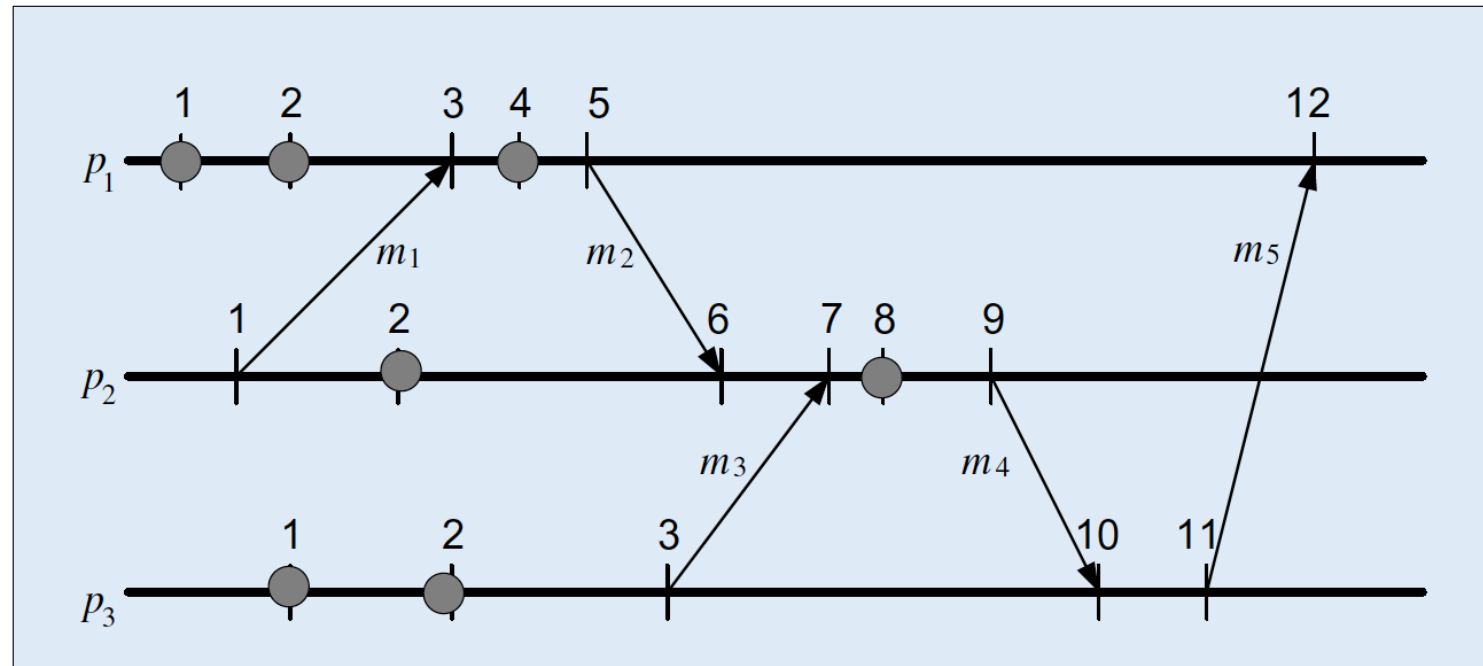
- Dizajn i analiza distribuiranog sistema zahtevaju jasnu vezu uzrok-posledica
- Često se tokom faze dizajna koristi koncept mašine prelaza stanja i utvrđuju se akcije koje dovode do prelaska u sledeće stanje
- Aktivnost bilo kog procesa se modeluje preko sekvence događaja koji se u njemu dešavaju
 - Događaji menjaju stanje – mora postojati veza uzrok-posledica (causality)
 - lokalni događaji
 - (njihov redosled se može utvrditi iz istorije stanja procesa)
 - Komunikacioni događaji
 - Njihov redosled je uslovljen time da poruka ne može biti primljena pre nego je poslata
 - Za događaje važi tranzitivnost
- Događaji koji u globalnoj istoriji nemaju vezu su *konkurentni događaji*

Distribuirani sistemi – Logički sat

- *Logički sat* je apstrakcija koja obezbeđuje suglasnost oko vremenskog redosleda u odsustvu pouzdanog globalog vremena
- Svaki proces mapira događaje na celobrojne vrednosti
- $LC(e)$ – lokalna varijabla koja je izračunata za event e
- Svaki proces označava svaki poslanu poruku sa vrednošću logičkog sata u momentu slanja $TS(m) = LC(send(m))$
- Pravila kako se azurira stanje logičkog sata
- $$LC(e) = \begin{cases} LC + 1, & \text{ako je } e \text{ lokalni događaj} \\ \max(LC, TS(m)) + 1, & \text{ako je događaj } e \text{ prijem poruke} \end{cases}$$

Distribuirani sistemi – Logički sat (2)

- Slika prikazuje upotrebu logičkog sata
- Logički sat ne omogućava globalno utvrđivanje redosleda svih događaja (npr. lokalnih događaja e_1^1 , e_2^1 , e_3^1)
- Ali omogućava da procesi koordinišu svoje logičke satove putem komunikacionih događaja
- *Gap detection* nije moguć s ovim pristupom



Distribuirani sistemi – pravila za isporuku poruka, causal delivery

- Apstrakcija koju smo koristili za komunikacioni kanal ne pretpostavlja ništa u pogledu redosleda isporuke poruka
- Ali realne mreže mogu izazvati da poruke ne stižu po redosledu
- Prijem poruke i **isporuka** poruke procesu su dve različite operacije koje su u uzročno posledičnoj vezi

$\text{receive}(m) \rightarrow \text{delivery}(m)$

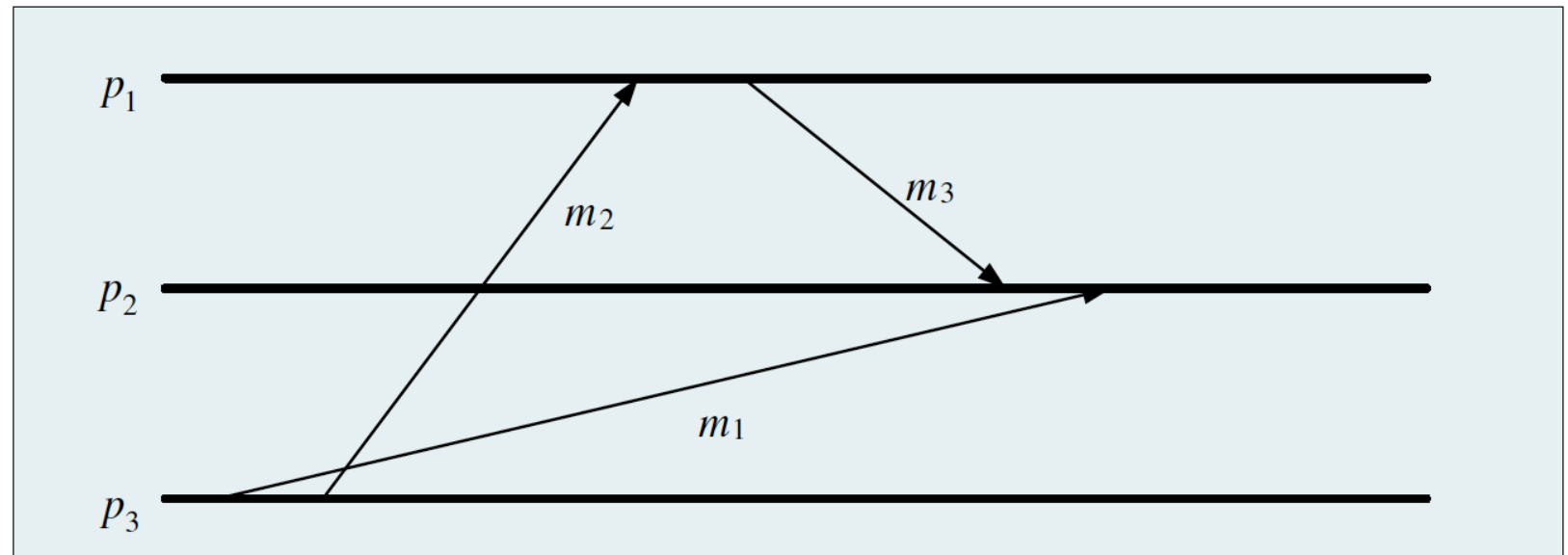
- FIFO isporuka – garantuje isporuku poruka onim redosledom kojim su poslane
 - Čak i kada sam komunikacioni kanal ne garantuje FIFO ona se može forsirano postići tako da se svakoj poruci doda broj sekvence
- Causal delivery – ekstenzija FIFO principa na situacije kada proces može primiti poruke iz različitih izvora

Distribuirani sistemi – pravila za isporuku poruka, causal delivery (2)

- Causal delivery – pretpostavlja da ako imamo dve poslane poruke od strane dva procesa (i, j), koje se isporučuju procesu k, redosled prijema mora odgovarati redosledu slanja između te dve poruke

$$\text{send}_i(m) \rightarrow \text{send}_j(m) \Rightarrow \text{deliver}_k(m) \rightarrow \text{deliver}_k(m)$$

- Kada postoji više procesa moguća je situacija da isporuka bude FIFO ali ne i uzročna (causal)



Distribuirani sistemi – pravila za isporuku poruka, causal delivery (3)

- Ako poruka m nosi *timestamp* $TS(m)$
- Poruka koju proces p_i primi je *stabilna* ukoliko ne postoji ni jedna poruka sa manjim *timestamp-om* koja bi mogla biti primljena naknadno
- Ukoliko se koristi mehanizam logičkog sata, onda proces p_i može izgraditi konzistentnu sliku sistema ukoliko primenjuje pravilo da se ***sve stabilne poruke isporuče po redosledu rastućih timestamp-ova***

Distribuirani sistemi – pravila za isporuku poruka, causal delivery (4)

- *Consistent message delivery* –
 - procesi imaju pristup globalnom realnom vremenu, i nema „driftovanja“ lokalnih satova
 - Kašnjenje poruka nije veće od δ ,
 - $RC(e)$ je timestamp poruke u vreme slanja i ugrađuje se u poruku
- Pravilo za isporuku u ovom slučaju je ***u momentu t isporuči sve primljene poruke sa timestampom do $(t - \delta)$ u rastućem poretku timestamp-a***
- Ovo pravilo garantuje da će uz ograničeno kašnjenje (do δ) sve poruke biti isporučene po redosledu
 - *clock condition* – ukoliko je događaj e prethodio e' onda je njegov $RC(e) < RC(e')$
 - *strong clock condition* - uspostavlja jednakost između redosleda događaja i njihovog timestamp-a

Distribuirani sistemi – pravila za isporuku poruka, causal delivery (5)

- Causal delivery - je jako bitan koncept jer omogućava procesu da stekene uvid u stanje sistema oslanjajući se pri tome samo na lokalne informacije (lokalni timestamp i poruke koje su do njega stigle)
- Ovo je tačno samo u „zatvorenim sistemima“ gde su svi kanali komunikacije poznati

Distribuirani sistemi – runs and cuts, causal history

- Postoji potreba da se u određenom momentu zna stanje određenih ili svih procesa
 - Proces za nadgledanje treba da bude u mogućnosti da detektuje pojavu *deadlock-a*
 - Proces može da migrira na drugu lokaciju ili da bude repliciran
- Monitor - proces koji je zadužen da formira „sliku“ globalnog stanja sistema
 - Kontaktira sve procese i traže da mu jave svoje trenutno stanje, a zatim te informacije kombinuje u globalno stanje sistema
 - Ovo je ekvivalent pravljenja *snapshota* svakog sistema i njihovo kombinovanje u globalnu „sliku“
 - Međutim kombinovanje snapshotova je jednoznačno samo ako svi procesi imaju pristup istom globalnom satu i snapshoti su napravljeni u istom trenutku

Distribuirani sistemi – runs and cuts, causal history

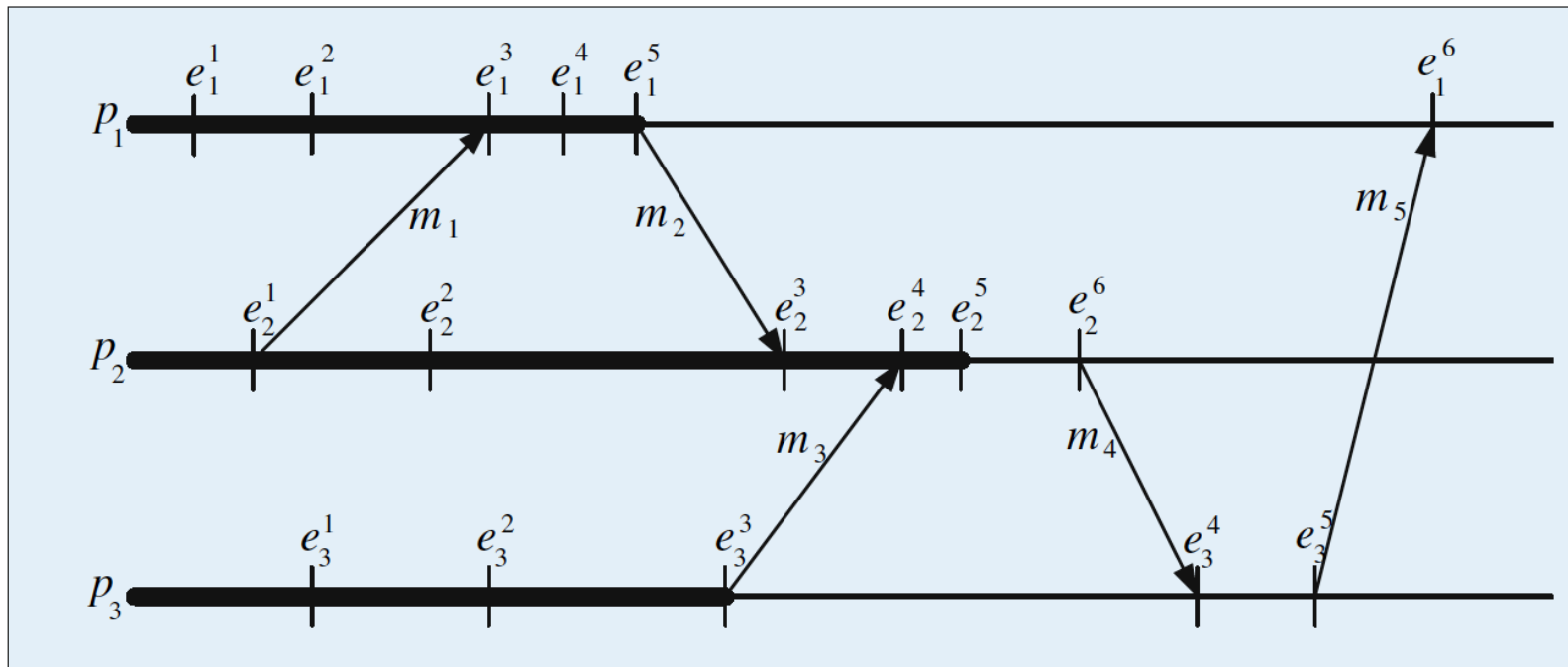
- Postoji potreba da se u određenom momentu zna stanje određenih ili svih procesa
 - Proces za nadgledanje treba da bude u mogućnosti da detektuje pojavu *deadlock-a*
 - Proces može da migrira na drugu lokaciju ili da bude repliciran
- Monitor - proces koji je zadužen da formira „sliku“ globalnog stanja sistema
 - Kontaktira sve procese i traže da mu jave svoje trenutno stanje, a zatim te informacije kombinuje u globalno stanje sistema
 - Ovo je ekvivalent pravljenja *snapshota* svakog sistema i njihovo kombinovanje u globalnu „sliku“
 - Međutim kombinovanje snapshotova je jednoznačno samo ako svi procesi imaju pristup istom globalnom satu i snapshoti su napravljeni u istom trenutku

Distribuirani sistemi – runs and cuts, causal history

- *Run* – R predstavlja uređen torku svih događaja u globalnoj istoriji koji je konzistentan sa svakom lokalnom istorijom procesa
 - Implicitno daje sekvencu događaja i sekvencu globalnih stanja sistema
- *Cut* – podskup lokalnih historija svakog procesa
- Granica reza (*frontier of the cut*) - n -tuple koja sadrži sve poslednje događaje u svim procesima koji su uključeni u rez.
 - Ova granica predstavlja „zamrznuto“ stanje sistema u nekom momentu
 - Obezbeđuje mogućnost generisanja globalnog stanja na osnovu razmene poruka između monitora i grupe procesa
- Neki rezovi nisu smisleni jer narušavaju pravilo causalnosti (C1 na sledećem slajdu)

Distribuirani sistemi – runs and cuts, causal history

- Consistent cut – onaj koji je obuhvatio događaje koji zadovoljavaju uzročno-posledičnu vezu
 - *Predstavlja instancu stanja celokupnog sistema*



Distribuirani sistemi – Konkurentnost

- Sposobnost da se nekoliko aktivnosti izvršava istovremeno
- Da bi se iskoristile prednosti konkurentnosti često problem mora da se sagleda iz drugačije perspektive i da bi se osmislio algoritam koji omogućava paralelizaciju
- Konkurentnost je ključni element dizajna softvera i celog sistema
 - Kernel sistema koristi konkurentnost kako bi virtuelizovao hardverske resurse i omogućio što bolje iskorišćenje
- Ponekad je konkurentnost prividna jer se obezbeđuje raspoređivanjem vremena resursa između procesa/niti koje se prividno izvršavaju istovremeno

Distribuirani sistemi – Konkurentnost (2)

- Često je cilj poboljšanje performansi sistema
- Donosi sa sobom i izazove
 - Zamena konteksta
 - Povećanje kompleksnosti sistema
 - Pažljivo usklađivanje vremenskih tokova
 - Neophodnost postojanja komunikacionih kanala za razmenu podataka i koordinaciju
 - Razmena poruka forsira i modularnost sistema kako bi se izbeglo da svi moduli „dele sudbinu“ ukoliko se desi neka greška
 - Obrasci komunikacije moraju biti prilagođeni arhitekturi
 - Više su strukturirani za paralelne sisteme,
 - manje strukturirani, ali više dinamični za distribuirane

Distribuirani sistemi – Atomske aktivnosti

- Koncept neophodan za uspešno korišćenje deljenih resursa
- Mora se obezbediti da se kompleksne operacije koje se suštinski sastoje od nekoliko jednostavnijih koraka završe bez prekidanja i u celini (transakcija) – operacija mora biti spolja gledano atomična (nedeljiva)
 - Promenjeno stanje sistema se ne vidi dok se akcija ne završi
 - Skrivanje stanja zapravo redukuje broj stanja u kome se sistem može naći - pojednostavljuje dizajn sistema
- Atomičnost se ne može implementirati bez neke vrste podrške za to od strane hardvera
 - *test-and-set* instrukcije, *compare-and-swap* instrukcije

Distribuirani sistemi – Atomske aktivnosti (2)

- Atomičnost se može obezbediti u dva oblika:
- *all-or-nothing*
 - ili je sve uspelo i stanje sistema promenjeno, ili sistem ostaje u stanju pre operacije
 - razlikuju se dve faze izvršavanja – pripremna i faza nepovratne izmene. Tokom prve faze moguće je vratiti se u prethodno stanje, tokom druge se mora dozvoliti neprekidnost i uspešnost operacije izmene.
 - Prelazak iz prve u drugu fazu – *commit point*
 - Zlatno pravilo – nikad ne menjajte jedinu kopiju – mora postojati barem log akcija / istorija izmena koji nam omogućava da se obezbedi konzistentnost i u slučaju grešaka

Distribuirani sistemi – Atomske aktivnosti (3)

- Atomičnost se može obezbediti u dva oblika:
- *Before-or-after*
 - Zasniva se na konceptu da se efekat višestrukih aktivnosti od strane korisnika vidi kao da su se one obavile sekvencijalno
 - Jači zahtev je da se uspostavi sekvencijalni red tranzicija između stanja
 - Transfer novca mora prvo povući novac s jednog računa, a zatim ga deponovati na drugi (ne može obrnuto), a ako ne uspe stanje oba računa mora ostati nepromenjeno

Distribuirani sistemi – Atomske aktivnosti (4)

- Atomičnost je kritičan koncept koji omogućava da se izgradi pouzdan sistem na osnovu nepouzdatih komponenti
- Upravljanje atomičnošću mora da reši sledeće probleme
 - Kako garantovati da samo jedna atomska akcija pristupa deljenom resursu
 - Kako se vratiti u prethodno stanje u slučaju neuspeha
 - Kako obezbediti da redosled pojedinačnih atomskih operacija vodi do konzistentnog celokupnog rezultata
- Rešenja za ove probleme donose dodatnu kompleksnost i dodatne probleme u sistem
 - Locks, critical sections, semaphores, monitors

Distribuirani sistemi – Protokoli za postizanje konsenzusa

- Konsenzus je proces postizanja saglasnosti o validnosti određene alternative (izbor) u prisustvu više njih
- Nijedan protokol koji je potpuno imun na greške (*fault tolerant*) ne može garantovati progresiju, ali postoje protokoli koji garantuju stepen slobode u odnosu na greške (*safety*)
- Familija protokola koja se zasniva na mašini prelaza stanja – konačnim automatima – *Paxos*
- Leslie Lamport je predložio nekoliko protokola
 - Disk Paxos, Cheap Paxos, Fast Paxos, Vertical Paxos, Stoppable Paxos, Byzantizing Paxos by Refinement, Generalized Consensus and Paxos, and Leaderless Byzantine Paxos.

Distribuirani sistemi – Protokoli za postizanje konsenzusa (2)

- Konsenzus se postiže između n servisa (procesa)
- Osnovni *Paxos* polazi od sledećih pretpostavki
 - Procesi se izvršavaju na procesorima i komuniciraju preko mreže. Procesori i mreža mogu doživeti otkaze.
 - Procesori rade na proizvoljnim brzinama; imaju stabilan (trajan) storage i mogu se ponovo pridružiti protokolu nakon otkaza; mogu slati poruke drugim procesorima;
 - Mreža može izgubiti ili duplirati poruke ili im presložiti redosled; poruke slati asinhorno a vreme do isporuke može biti proizvoljno dugo

Distribuirani sistemi – Protokoli za postizanje konsenzusa (3)

- Nadalje osnovni *Paxos* pretpostavlja postojanje entiteta u različitim ulogama
 - *Client* – agent koji šalje zahtev i čeka odgovor
 - *Proposer* – agent koji „zatupa“ zahtev koji je klijent poslao i pokušava da uveri ostale da ga prihvate i nastupa kao koordinator
 - *Acceptor* – agent koji se ponaša kao *fault tolerant* memorija protokola (pamti sve ranije usvojene odluke)
 - *Learner* – agent koji nastupa kao replikator i koji preduzima akcije kada se konsenzus usvoji
 - *Leader* – istaknuti predlagač. Onaj koji u datom momentu upravlja protokolom
- Cilj protokola je da se u konačnom vremenu dođe do konsenzusa oko toga koji predlog kog agenta postaje važeći i to postaje prihvaćeno stanje sistema

Paralelni i distribuirani sistemi

- Pitanja?

