CLOUD PITANJA 1

1. Šta je računarstvo u oblaku?
2. Koje sve resurse nudi na iznajmljivanje računarstvo u oblaku?
3. Koji su to rani modeli
4. Glavne karakteris>ke?
5. Koje su ključne prednos> računarstva u oblaku?
6. Izazovi računarstva u oblaku?
7. Koja su tri glavna modela isporuke ponuđenih usluga (delivery models)?
8. Koje su osobine svakog od ovih modela isporuke?
9. Koji koncep> razvijeni za paralelno i distribuirano računarstvo su ključni i za računarstvo u oblaku?
10. Osnovni koncep> paralelnog računarstva?
11. Čime je ograničeno maksimalno ubrzavanje koje se paralelizacijom može pos>ći? Amdalov zakon.
12. Da li povećanje broja procesorskih jedinica garantuje stalno povećanje performansi sistema?
13. Koji su ključni izazovi/problemi paralelnih sistema? Šta tačno predstavlja svaki od njih?
14. Koji su >pični modeli komunikacije između procesa?
15. Na kojim nivoima možemo posmatra> paralelizaciju?
16. Koje su >pične arhitekture paralelnih sistema?
17. Šta su distribuirani sistemi?
18. Glavne karakteris>ke distribuiranih sistema?
19. Koja poželjna svojstva treba da ima distribuirani sistem?
20. Globalno stanje grupe procesa? Koji model sa koje dve ključne apstrakcije se koriste za sagledavanje distribuiranih sistema?
21. Šta su procesi i ni>?
22. Šta su komunikacioni kanali?
23. Koja je ključna pretpostavka koju moramo uze> u obzir kada koris>mo komunikacione kanale?
24. Kako sagledavamo stanje procesa?
25. Kako posmatramo stanje samog komunikacionog kanala?
26. Šta predstavlja protokol?
27. Kako sagledavamo globalno stanje grupe procesa?
28. Koji događaji definišu globalno stanje grupe procesa?
29. Kako broj mogućih tranzicija u>će na složenost analize/debagovanja sistema?
30. Zašto su neophodni mehanizmi za otkrivanje i uklanjanje grešaka u komunikacionim kanalima?
31. Svaki proces ima lokalnu istoriju promena povezanih sa lokalnim vremenom? Šta je problem sinhronizacije?
32. Da li je moguća apsolutna globalna suglasnost o vremenu u velikim distribuiranim sistemima?
33. Da li je apsolutni konsenzus o tačnom vremenu neophodan ili je za koordinaciju sistema potreban mehanizam koji omogućava utvrđivanje redosleda događaja?
34. Zašto je bitan koncept uzročno-posledičnih veza (causality)
35. Koji događaji se smatraju konkurentnim?

1. Računarstvo u oblaku je isporuka računarskih resursa i usluga preko interneta, što omogućava brz i jednostavan pristup deljenim resursima. Ovi resursi mogu uključivati servere, prostor za čuvanje podataka, baze podataka, mrežnu infrastrukturu, softver, analitiku i druge računarske resurse. Kompanije koje nude ove usluge nazivaju se “cloud provider”, a tipični model naplate usluga je “po trošku”, slično obračunu potrošnje električne energije ili komunikacionih usluga.

   Računarstvo u oblaku omogućava korisnicima da pristupe resursima i uslugama na zahtev, u kratom roku i uz minimalan napor za podešavanje. Usluga se pruža putem internet, što olakšava pristup i skalabilnost za korisnike. Ovaj pristup računarstvu omogućava fleksibilnost, smanjenje troškova i povećanje efikasnosti za različite korisnike i organizacije.

2. Računarstvo u oblaku nudi sledeće resurse na iznajmljivanje:
   a. Server
   b. Prostor za čuvanje podataka
   c. Baze podataka
   d. Mrežne infrastrukture
   e. Softver
   f. Analitiku
3. Razlog za pojavu ranih modela je to što se računarska obrada može brže izvršiti ukoliko se koriste farme računarskih resursa i sistema za čuvanje podataka. Rani modeli se dele na:
   a. **Grid computing** - Infrastruktura kod koje se povezuje i koordiniše rad geografski dislociranih računarskih resursa radi postizanja cilja. Neiskorišćeni računarski kapaciteti na povezanim računarima se stavljaju na raspolaganje za izvršavanje zajedničkog zadatka.
   b. **Utility computing** - Omogućava korisnicima da računarske resurse skaliraju po svojim potrebama. Korisnici unajmljuju resurse kao što su procesna snaga, prostor za čuvanje podataka, aplikacije, ili stvarno iznajmljivanje fizičkog hardware-a.
4. Glavne karakteristike računarstva u oblaku:
   a. Cloud computing obezbeđuje skalabilno i elastično računarsko okruženje. Resursi koji se koriste za pružene usluge se mere i naplaćuju po utrošku.
   b. **Deljeni resursi i upravljanje resursima** - Koristi se veliki bazen deljenih resursa. Koriste se internet tehnologije da bi se resursi učinili dostupnim kao skalabilne i elastične usluge. Elastičnost se odražava tako što se resursi obezbeđuju dinamički, na zahtev i u količini koja je potrebna. Upotreba resursa se meri i ekonomično je zbog efekta multipleksiranja upotrebe resursa.
   c. **Čuvanje podataka** - Podaci se čuvaju (“u oblaku”) na različitim lokacijama, a nekad i bliže mestu upotrebe. Korisnik nike svestan lokacije gde su podaci, a strategije čuvanja podataka (višestruke kopije) povećavaju pouzdanost sistema.
   d. **Upravljanje** - Upravljanje i bezbednost resursa je odgovornost ponuđača usluge. Oni posao upravljanja mogu da obavljaju efikasno jer imaju nephodnu

specijalizaciju (kadrove, opremu) i jer je većina resursa centralizovana u data centre.

5. Ključne prednosti računarstva u oblaku:
    a. **Resursi** (radni ciklusi CPU, prostor za čuvanje podataka, mreža) se dele između korisnika.
    b. Kada više aplikacija dele određene resurse, period njihovog najvećeg opterećenja nije sinhronizovan, što omogućava da se korišćenje resursa multipleksira - mnogo bolje iskorišćenje.
    c. Resursi se mogu udruživati kako bi zadovoljili i najzahtevnije aplikacije.
    d. Deljenje podataka omogućava kolaborativni rad, koji mogu obavljati različite grupe ljudi bilo gde u svetu.
    e. Eliminišu se veliki početni troškovi za nabavku opreme.
    f. **Smanjenje troškova** - koncetracija resursa (u data centre), koji se dele između korisnika, omogućava plaćanje po principu “plati koliko trošiš” (pay as you go).
    g. **Elastičnost** - sposobnost za prilagođavanje količine resursa koji se koriste tako da i aplikacije koje imaju veliki odnos vrha opterećenja/regularno opterećenje mogu raditi efikasno.
    h. **Prilagođenost korisnicima** - virtuelizacija omogućava da se korisnicima kreira poznato okruženje.
6. Izazovi računarstva u oblaku su:
    a. **Dostupnost usluga** - šta se dešava ukoliko ponuđač usluge u nekom momentu nije u stanju da pruža usluge?
    b. **Privatnost, zaštita i analiza podataka** - ozbiljan razlog za zabrinutost.
    c. Širok spektar servisa, različiti načini organizacije podataka, korisnički interfejsi koje različiti ponuđači nude - smanjuje se mogućnost prelaska korisnika kod drugih ponuđača - jednom kada ste izabrali platformu teško je preći na drugu.
    d. Uska grla u prenosu podataka.
    e. **Nepredvidljive performanse** - posledica deljenja resursa.
    f. **Upravljanje resursima** - izazovno je optimalno upravljati resursima u uslovima promenljivih i velikih opterećenja.
    g. **Bezbednost i privatnost** - posebno za aplikacije koje rade sa osetljivim podacima.
7. Tri glavna modela isporuke ponuđenih usluga su:
    a. Software as a Service (SaaS)
    b. Platform as a Service (PaaS)
    c. Infrastructure as a Service (IaaS)
8. Osobine svakog od modela isporuke:
    a. IaaS - Obezbeđuje se infrastruktura (računarski resursi CPU, VMs, storage(serveri)), i na tu infrastrukturu korisnik može da instalira i pokreće proizvoljan softver pa i OS. Korisnik ne kontroliše cloud infrastrukturu koja mu je obezbedila virtuelizovane resurse, ali kontroliše sve od operativnog sistema, alociranog prostora, instaliranih aplikacija, u određenom obimu mrežnih uređaja. Usluge koje se nude u ovoj kategoriji: (Server hosting, storage, computing hardware, operating systems, virtual instances, load balancing, internet access, and bandwidth provisioning). Primer je Amazon EC2.

   b. **PaaS** - Omogućava korisniku da instalira svoje ili kupljene aplikacije korišćenjem različitih alata i programskih jezika koje ponuđač usluge podržava. Primeri su Google App Engine i Windows Azure.
      i. Korisnik:
         1. Kontroliše instalirane aplikacije i donekle izvršno okruženje u kome se aplikacije pokreću.
         2. Nema kontrolu nad cloud infrastrukturom niti nad operativnim sistemom ili prostorom za čuvanje podataka.
      ii. Nije posebno korisno u situacijama kada:
         1. Aplikacije moraju biti portabilne
         2. Koriste neki specifični programski jezici za razvoj
         3. Hardver i softver moraju da budu prilagođeni da bi se postigle željene performanse aplikacija
   c. **SaaS** - Aplikacije koje ponuđač stavlja na direktnu upotrebu korisnicima. Korisnik ne kontroliše niti cloud infrastrukturu niti same aplikacije, može samo da ih koristi u skladu sa uslovima koje je postavio ponuđač. Ovaj model nije pogodan za specijalizovane aplikacije koje rade sa podacima u realnom vremenu ili kada je zahtev da podaci ne smeju da se čuvaju eksterno. Usluge koje se nude su: (workflow management, communications, digital signature, customer relationship, management (CRM), desktop software, financial management, geospatial, and search). Primeri su Gmail i Salesforce.
9. Koncepti razvijeni za paralelno i distribuirano računarstvo koji su ključni i za računarstvo u oblaku:
   a. **Komunikacioni protokoli** - Komunikacioni protokoli su ključni aspekti paralelnih i distribuiranih sistema. Oni omogućavaju različitim delovima sistema da komuniciraju međusobno na efikasan način.
   b. **Koncept "konzistentnog preseka" (consistent cuts) i distribuiranih "zamrznutih" snimaka (snapshots)** - Konzistentni presek je stanje distribuiranog sistema u kojem se sve komponente sistema nalaze u konzistentnom stanju. U kontekstu distribuiranih sistema, ovo znači da je moguće obnoviti sistem na osnovu ovog stanje bez ikakvih konlikata ili nekonzistentnosti. Distribuirani zamrznuti snimci su fotografije stanja svih komponenti sistema u određenom trenutku. Ove fotografije omogućavaju kontrolne tačke i restartovanje aplikacije u slučaju grešaka ili otkaza
   c. **Upotreba monitora** - Monitori su sistemski alati koji prikupljaju podatke o stanju pojedinačnih komponenti i sistema u distribuiranom okruženju. Oni omogućavaju administratorima i softveru uvid u performanse, resurse i potencijalne probleme. U kontekstu računarstva u oblaku, mnoge funkcije i usluge zavise od informacija prikupljenih pomoću monitora. Na primer, automatizacija skaliranja resursa, analiza performanski, detekcija grešaka i održavanje sistema zavise od informacija dobijenih iz monitora.
10. **Paralelno računarstvo** predstavlja pristup u kojem se kompleksni problemi rešavaju razbijanjem na jednostavnije delove, koji se zatim rešavaju konkurentno (paralelno). Ovaj pristup je inspirisan efikasnošću paralelnog rada u grupi u prirodi. Paralelno računarstvo je omogućilo rešavanje zadataka koji zahtevaju kompleksnu obradu velikih količina podataka i podstaklo napredak u oblastima kao što su algoritmi, arhitektura računara i mreže.

Hardver i softver za paralelno izvršavanje omogućavaju rešavanje problema koji zahtevaju više resursa nego što bilo koji pojedinačni sistem može da obezbedi, a istovremeno omogućavaju bržu obradu podataka. Paralelno računarstvo je ključni koncept koji je doprineo razvoju distribuiranih računarskih sistema u oblaku.

11. Ubrzanje koje se postiže paralelizacijom je ograničeno:

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

    T(1) - vreme potrebno serijskom algoritmu da izvrši obradu

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

    Amdalov zakon definiše teoretski moguće ubrzanje koje se može postići paralelizacijom. Zakon tvrdi da je opšte ubrzanje koje se može dobiti optimizacijom jednog dela sistema ograničeno delom ukupnog vremena u kojem se optimizovani segment zaista koristi u samom sistemu.

    Ako je **α** deo vremena koje program provodi u segmentu koda koji nije paralelizovan, teoretski limit za ubrzanje je **S = 1/α**. Amdalov zakon se često koristi u paralelnom računarstvu kako bi se predvidelo teoretsko ubrzanje kada se koriste multiprocesorski sistemi.

    Najpririmenljiviji je za probleme fiksne veličine, gde se količina posla koji svaki paralelizovan proces obavlja smanmjuje sa povećanjem broja procesa, a svi procesi dobijaju istu količinu posla.

12. Povećanje broja procesorskih jedinica **ne garantuje** uvek stalno povećanje performansi sistema. **Amdalov zakon** ukazuje na teoretsko ograničenje ubrzanja koje se može postići povećanjem broja procesorskih jedinica. Ako deo koda nije paralelizovan, ubrzanje će biti ograničeno tim delom koda.
13. Izazovi paralelnog računarstva:
    a. **Koordinacija konkurentnog izvršavanja** - U paralelnim sistemima, više procesa ili niti radi istovremeno na rešavanju zadatka. Koordinacija tih procesa ili niti zahteva složene mehanizme za sihronizaciju i komunikaciju kako bi se osiguralo ispravno izvršavanje programa.
    b. **Overhead za sihronizaciju** - Kada se program pralelizuje, dodatni resursi i vreme se troše na uspostavljanje komunikacije i sihronizacije između procesa ili niti. Ovaj dodatni trošak može smanjiti ukupno ubrzanje koje se postiše paralelizacijom, posebno ako je potrebno mnogo sihronizacije.
    c. **Barrier synchronization** - Ovaj koncept se odnosi na fazno izvršavanje paralelnih programa, gde sve niti ili procesi moraju završiti jednu fazu pre nego što pređu na sledeću fazu. Barrier synchronization osigurava da se niti ne mešaju između faza, ali može ograničiti paralelizam ako niti moraju često čekati jedna na drugu.
    d. **Race conditions** - Ovaj problem nastaje kada rezultat programa zavisi od relativnog redosleda izvršavanja konkurentnih operacija. Race conditions mogu dovesti do nepredvidljivih i nekonzistentnih rezultata, što može biti teško dijagnostifikovati i ispraviti.

    e. **Deadlock** - Deadlock se javlja kada dva ili više procesa ili niti čekaju jedni na druge da oslobode resurse, što rezultuje blokadom u kojoj nijedan proces ne može da napreduje. Coffmanovi kriterijumi za upadanje u deadlock su: međusobno isključivanje, držanje i čekanje, nema prinudnog oslobađanja resursa i cirkularno čekanje. Deadlock može dovesti do potpunog zastoja sistema i zahteva posebne mehanizme za otkrivanje i oporavak.
    f. **Livelock** - Livelock je sličan Deadlocku, ali se javlja kada procesi kontinuirano menjaju svoje stanje u odgovoru na promene u stanju drugih procesa, umesto da budu potpuno blokirani. Livelock može dovesti do beskonačnog ciklusa promena stanja bez napretka u rešavanju zadatka.
    g. **Priority inversion** - U sistemima sa prioritetnim raspoređivanjem, procesi sa višim prioritetom trebaju da se izvršavaju pre onih sa nižim prioritetima. Međutim, priority inversion se javlja kada proces nižeg prioriteta prekida proces višeg prioriteta, što može dovesti do smanjenja performansi.
14. Procesi/niti mogu komunicirati razmenom poruka ili preko deljenih memorijskih lokacija:
    a. **Deljena memorija** - Procesori s više jezgra često koriste deljene memorijske lokacije za komunikaciju, ali moderni superračunari retko koriste ovaj princip zbog problema sa skalabilnošću (mnogo procesora bi trebalo da ima pristup istim memorijskim lokacijama pa bi se povećala konkurentnost). Sistemski softver često koristi deljenu memoriju za upravljanje raspoređivanjem, tabelama procesa/niti i upravljanje virtuelnom memorijom. Međutim, debagovanje u ovom slučaju može biti kompleksno.
    b. **Prosleđivanje poruka** - Ovaj pristup se tipično koristi u velikim distribuiranim sistemima, jer je lakše skalirati i debagovati u poređenju sa deljenom memorijom. Poruke se šalju između procesa ili niti kako bi se razmenjivali podaci ili koordinisale aktivnosti.
15. Paralelizam možemo posmatrati na više nivoa:
    a. **Paralelizam na nivou bita** - Odnosi se na broj bita koji se obrađuju po ciklusu procesora (8-bit, 16-bit, 32-bit, 64-bit). Ovo smanjuje broj potrebnih instrukcija za obradu većih operanada i povećava adresni prostor za upravljanje memorijom.
    b. **Paralelizam na nivou instrukcija** - Uključuje procesne pipeline-e sa više faza, kao što su RISC (5 faza) i CISC (veći broj faza). Ovo omogućava simultano izvršavanje i povećanje brzine obrade.
    c. **Paralelizam na nivou podataka ili ciklusa** - Omogućava izvršavanje programskih petlji (ciklusa) u paraleli, čime se poboljšava performansa za zadatke koji uključuju obradu velikih količina podataka.
    d. **Paralelizam zadataka** - Problem se dekomponuje u međusobno nezavisne zadatke koji se zatim mogu obavljati paralelno. Ovo omogućava efikasniju upotrebu resursa i povećava performanse sistema.
16. Tipične arhitekture paralelnih sistema:
    a. **SISD (Single Instruction, Single Data)** - Ova arhitektura nema paralelizam.
    b. **SIMD (Single Instruction, Multiple Data)** - SIMD arhitektura omogućava da se jedna instrukcija istovremeno primeni na više podataka. Ovaj pristup se često koristi u vektorskom procesiranju, gde ista instrukcija može biti izvršena paralelno nad svim komponentama vektora. SIMD je pogodan za zadatke sa

visokim stepenom podatkovanog paralelizma, kao što su grafička obrada i naučni proračuni.

c. **MIMD (Multiple Instruction, Multiple Data)** - MIMD arhitektura uključuje više procesora koji funkcionišu asinhrono i nezavisno, svaki obrađujući različite instrukcije nad različitim podacima istovremeno. Ova vrsta arhitekture je zastupljena u modernim paralelnim i distribuiranim sistemima.

   MIMD arhitektura može koristiti različite pristupe memoriji:

   1. Uniform Memory Access (UMA) - Procesori dele zajedničku memoriju sa uniformnim vremenom pristupa.
   2. Cache Only Memory Access (COMA) - Svaki procesor ima lokalnu keš memoriju, a glavna memorija je sastavljena od tih keševa.
   3. Non-Uniform Memory Access (NUMA) - Procesori dele zajedničku memoriju, ali vreme pristupa varira u zavisnosti od lokacije memorije.

   MIMD arhitektura takođe može koristiti distribuiranu memoriju, gde procesori i memorija komuniciraju preko različitih mrežnih topologija, kao što su hiperkocka, 2D torus, 3D torus i omega mreže.

17. **DISTRIBUIRANI SISTEMI** su 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. Koncept postoji već decenijama (distribuirani fajl sistemi, RPC - remote procedure call - komunikacija koja omogućuje pozivanje procedura koje se nalaze u drugom adresnom prostoru ili udaljenom računaru).
18. Karakteristike distribuiranih sistema su:
    a. Autonomne komponente
    b. Raspoređivanje (scheduling) i upravljanje resursima se implementira na svakom od sistema
    c. Više tačaka upravljanja i više tačaka u kojima može doći do otkaza
    d. U opštem slučaju, resursi ne moraju biti dostupni sve vreme
    e. Skaliranje se obavlja dodavanjem novih komponenti u sistem
    f. Mogu se dizajnirati tako da obezbede relativno visok nivo dostupnosti čak i u uslovima slabe pouzdanosti hardvera, softvera ili mreže
19. Poželjna svojstva distribuiranih sistema su:
    a. **Transparentan pristup** (Access transparency) - lokalnim i udaljenim informacionim objektima se pristupa putem istih operacija
    b. **Transparentnost lokacije** (Location transparency) - informacionim objektima se pristupa na takav način da korisnik ne mora znati gde se on nalazi
    c. **Transparentnost konkurentnog izvršavanja** (Concurrency transparency) - više procesa može da se izvršava konkurentno, pristupajući deljenim informacionim objektima, a da to ne izazviva međusbno ometanje
    d. **Transparentnost replikacije** (Replication transparency) - koristi se više instanci informacionih objekata kako bi se povećala pouzdanost, a da toga ne moraju biti svesni ni korisnici ni aplikacije

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

20. Za razumevanje distribuiranih sistema može se koristiti model apstrakcije sa dve ključne komponente: procesi i komunikacioni kanali. Apstrakcije procesa i komunikacionog kanala omogućavaju da se kritična svojstva distribuiranog sistema posmatraju bez ulaska u fizičke detalje pojedinih implementacija i entiteta.
21. Proces predstavlja program koji se izvršava, a niti predstavljaju manje zadatke unutar procesa (manji procesi). Nit predstavlja najmanju procesnu jedinicu koji operativni sistem može da raspoređuje. Grupa procesa predstavlja procese koji sarađuju tj. koordinisano rade kako bi se postigao određeni zajednički cilj.
22. Komunikacioni kanal obezbeđuje mogućnost komunikacije između procesa ili niti putem razmene poruka. U najopštijem slučaju možemo smatrati da se komunikacija obavlja putem komunikacionih događaja slanja i prijema poruke (apstrahovan je kao jednosmerni kanal za prenos bita, beskonačnog propusnog opsega i nulte latencije, ali je nepouzdan).
23. Ključna pretpostavka koja se mora uzeti u obzir prilikom korišćenja komunikacionih kanala je njihova nepouzdanost.
24. Proces je u nekom trenutku okarakterisan svojim stanjem koje predstavlja sve informacije koje su nam potrebne da uspešno restartujemo proces nakon što je bio suspendovan. Promena stanja procesa je događaj.
25. Stanje komunikacionog kanala se može definisati posmatrajući dva procesa pi i pj. Stanje komunikacionog kanala od pi do pj se sastoji od poruka koje je pi poslao, a pj jos nije primio.
26. Protokol predstavlja konačan skup poruka koje procesi razmenjuju da bi koordinisali svoje aktivnosti.
27. Globalno stanje grupe procesa sagledavamo kroz uniju stanja svih procesa i komunikacionih kanala date grupe. Stanje kanala u globalnom stanju nije eksplicutno prisutno, već posredno (lokalno stanje pojedinih procesa je uslovljeno obavljenim komunikacijama). Globalna stanja formiraju n dimenzionalnu rešetku mogućih stanja grupe procesa.
28. Komunikacioni događaji su ti koji definišu koja globalna stanja grupa procesa može dostići.
29. Broj mogućih tranzicija (putanja kroz rešetku koja vodi iz jednog u drugo globalno stanje) je ogroman, što samim tim čini analizu i debagovanje sistema sa velikim brojem konkurentnih procesa ili niti sve komplikovanijim.
30. Jedan od glavnih problema u paralelnim i distribuiranim sistemima je komunikacija nepouzdanim komunikacionim kanalima, odnosno komunikacionim sistemima koji imaju određenu verovatnoću otkaza. Kako bi procesi mogli pouzdano da komuniciraju, implementiraju se mehanizmi za otkrivanje i ispravljanje grešaka. Pored mehanizama za kontrolu grešaka, implementiraju se i mehanizmi za kontrolu

toka i kontrolu zagušenja komunikacionog kanala. Kontrola toka obezbeđuje povratnu informaciju od primaoca i ne dozvoljava pošiljaocu da pošalje više informacija od onoga koliko je primalac u stanju da prihvati i obradi. Kontrola zagušenja se brine o tome da količina podataka koja se stavi na raspolaganje ne prevaziđe kapacitet mreže. Pošiljalac može korišćenjem RTT da proceni da li je mreža zagušena i smanji brzinu slanja, kako bi se izbegla mogućnost gubljenja paketa.

31. Svaki proces ima lokalnu istoriju i svaka promena je povezana sa lokalnim vremenom koji obezbeđuju relativno merenje vremena. Poruke koje procesi šalju jedni drugima se mogu izgubiti ili biti oštećeni i bez dodatnih mehanizama zaštite, ne postoji način da se obezbedi savršena sinhronizacija lokalnih satova, a samim tim ni način da se utvrdi jednoznačan globalni sled događaja koji su nastali u različitim procesima.
32. U distribuiranim sistemima neophodno je postojanje globalne sagalasnosti 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, odnosno 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.
33. *ovaj pasus ima i iznad isto* -> Neophodan je konsenzus za 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.

    Mehanizam koji omogućava utvrdjivanje redosleda dogadjaja je Time stamps. Odredjuje se redosled tako što se uzme globalno vreme na koje se sinhronizuju lokalni satovi. Ukoliko lokalni satovi ne greše više od π (preciznost) u odnosu na globalno vreme, i g je granaularnost fizičkog sata (i ona ne bi trebala biti manja od π), za dva događaja koja se dešavaju u dva procesa tb i ta, ako važi $tb - ta \leq \pi + g$ nemoguće je utvrditi koji događaj je prethodio onom drugom.

34. Dizajn i analiza distribuiranog sistema zahtevaju jasnu vezu uzrok-posledica. Aktivnost bilo kog procesa se modeluje preko sekvence događaja koji se u njemu dešavaju.

    Događaji menjaju stanje pa mora postojati veza uzrok-posledica (causality) i za sve događaje važi tranzitivnost.

    Lokalni događaji - redosled se može utvrditi iz istorije stanja procesa.

    Komunikacioni događaji - redosled je uslovljen time da poruka ne može biti primljena pre nego je poslata.

35. Događaji koji u globalnoj istoriji nemaju vezu su **konkurentni događaji**
36. 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.

Pravila kako se ažurira stanje logičkog sata:

- LC(e) je LC + 1 ako je e lokalni dogadjaj ili
- max(LC, TS(m)) + 1 ako je dogadjaj e priijem poruke gde je TS(m) = LC(send(m))

Logički sat ne omogućava globalno utvrđivanje redosleda svih događaja, ali omogućava da procesi koordinišu svoje logičke satove putem komunikacionih događaja. Gap detection nije moguć s ovim pristupom

37. Apstrakcija komunikacionog kanala (da je jednosmerni kanal za prenos bita, beskonačnog propusnog opsega i nulte latencije) ne pretpostavlja ništa u pogledu redosleda isporuke poruka. Realne mreže mogu izazvati da poruke ne stižu po redosledu.
38. Prijem poruke i isporuka poruke procesu su dve različite operacije koje su u uzročno posledičnoj vezi. **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.
39. **Causal delivery isporuka** - proširenje FIFO isporuke na situacije kada proces može primiti poruke iz različitih izvora. 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. Kada postoji više procesa moguća je situacija da isporuka bude FIFO ali ne i uzročna (causal).

40. Poruka koju proces primi je stabilna ukoliko ne postoji ni jedna poruka sa manjim timestamp-om koja bi mogla biti primljena naknadno.

    Proces može izgraditi konzistentnu sliku sistema ukoliko primenjuje pravilo da se sve stabilne poruke isporuče po redosledu rastućih timestamp-ova.

41. Kod **Consistent message delivery** isporuke:

- procesi imaju pristup globalnom realnom vremenu, i nema driftovanja lokalnih sativa
- kašnjenje poruka nije veće od δ
- RC(e) je timestamp poruke u vreme slanja i ugradjuje se u poruku

Pravilo je da u momentu t isporučujemo sve primljene poruke sa timestampom do t-δ u rastućem poretku timestamp-a. Ovo garantuje da će uz ograničeno kašnjenje sve poruke biti isporučene po redosledu.

42. **Monitor** je proces koji je zadužen da formira sliku globalnog stanja sistema. Kontaktira sve procese i traži da mu jave svoje trenutno stanje, a zatim te informacije kombinuje u globalno stanje sistema.
43. **Run** predstavlja uredjenu torku svih dogadjaja u globalnoj istoriji. Konzistentan je sa svakom lokalnom istorijom procesa. Implicitno daje sekvencu događaja i sekvencu globalnih stanja sistema.

Cut – podskup lokalnih istorija svakog procesa.

Granica reza (frontier of the cut) - tuple koji sadrži sve poslednje događaje u svim procesima koji su uključeni u rez. Ova granica predstavlja zamrznuto stanje sistema u nekom momentu.

Neki rezovi nisu smisleni jer narušavaju pravilo causalnosti.
Consistent cut – onaj koji je obuhvatio događaje koji zadovoljavaju uzročno-posledičnu vezu.

44. Konkurentnost je sposobnost da se nekoliko aktivnosti izvršava istovremeno. Može biti prividna ako se vreme zauzetosti resursa raspoređuje između procesa/niti. Cilj konkurentnosti je poboljšanje peformansi sistema. Izazovi koje donosi konkurentnost
    a. Zamena konteksta - prebacivanje izvršavanja jednog zadatka na drugi
    b. Povećanje kompleksnosti sistema
    c. Pažljivo usklađivanje vremenskih tokova - bitno je u kom redosledu su se događaji desili
    d. Neophodnost postojanja komunikacionih kanala - da bi se osigurala pravilna i bezbedna upotreba deljenih resursa mora postojati mehanizam za koordinaciju i komunikaciju između procesa
    e. Modularnost sistema razmenom poruka umesto direktnog deljenja memorije što doprinosi da će usled neke greške samo jedan modul biti pogođen
    f. Komunikacija je više strukturirana za paralelne sisteme, a za distribuirane je manje strukurirana i dinamičnija
45. Kompleksne operacije koje se sastoje od niza jednostavnih operacija se moraju izvršiti bez prekidanja (transakcija). Promenjeno stanje sistema će biti vidljivo samo kada se akcija cela završi. Atomičnost se mora implementirati uz neku podršku hardvera. Može se obezbediti u dva oblika
    a. All or nothing - ako je čitava transakcija uspela, stanje sistema se menja. Prva faza izvršavanja je pripremna i u toku nje je moguće vratiti se u prethodno stanje, a druga faza je faza nepovratne izmene u kojoj se mora dozvoliti neprekidnost operacije izmene. Prelazak iz prve u drugu fazu je operacija commit.
    b. Before or after - promene se izvršavaju pre ili posle izvršavanja transakcije kako bi vreme blokiranja baze bio manji i transakcija bila efikasnija. U slučaju greške tokom neke od akcija u nizu, sistem će pokušati da reši problem tako što će ponovo izvršiti akcije pre ove problematične kako bi izbegao gubitak podataka.
46. Konsenzus se koristi za postizanje dogovora (odabira neke alternative) između čvorova koji zajedno treba da obave neku transakciju. Protokoli za ostvarivanje konsenzusa omogućavaju da sistem sa više čvorova radi kao celina i bude stabilan i pouzdan usled grešaka.
47. Pretpostavke od kojih polazi Paxos protokol
    a. Procesi se izvršavaju na procesorima i komuniciraju preko mreže. Procesori i mreža mogu doživeti otkaze
    b. Procesori rade proizvoljnim brzinama, imaju trajno skladište, mogu slati poruke drugim procesorima i mogu se pridružiti protokolu nakon otkaza
    c. Mreža može izgubiti, duplirati poruke ili im izmeniti redosled. Poruke se šalju asinhrono i vreme od slanja do isporuke može biti proizvoljno dugo.

48. Ključni učesnici u Paxos protokolu su:
    a. klijent - šalje zahtev i čeka odgovor
    b. proposer - zastupa poslati zahtev i pokušava da uveri ostale da ga prihvate
    c. acceptor - agent koji bira između predloženih opcija i prihvata jednu od njih
    d. learner - agent koji vrši replikaciju podataka kada se konsenzus usvoji tj širi informaciju o prihvaćenom pregledu
    e. leader - privremeni čvor koji upravlja glasanjem i predstavlja vezu između acceptora i proposera

    Cilj protokola je da se u konačnom vremenu postigne dogovor tj prihvati predlog nekog agenta čije stanje postaje važeće za ceo sistem

49. **VIRTUELIZACIJA** je tehnologija koja značajno pojednostavljuje upravljanje fizičkim resursima kreiranjem virtuelne, umesto realne verzije OS, servera, skladišnog prostora, mrežnih resursa. Koristi softver kako bi simulirala i pojednostavila funkcionalnost hardvera, izolovala međusobno korisnike i time formira virtuelni računarski sistem, te omogućava replikaciju koja povećava elastičnost sistema.
50. Značaj virtuelizacije u računarstvu u oblaku je to što omogućava:
    a. **Izolaciju performansi** - resursi se mogu dinamički dodjeljivati raznim korisnicama ili aplikacijama, po potrebi
    b. **Bezbednost sistema** - izolacija servisa koji se pokreću na istom hardveru
    c. **Bolje performanse i pouzdanost** - omogućava da se po potrebi aplikacije migriraju sa jedne na drugu hardversku platformu
    d. Ponuđač resursa je u mogućnosti da ponudi servise za upravljanje virtuelizovanim resursima
51. Virtuelizacija se može vršiti na sledeće načine:
    a. **Multipleksiranje** (many virtual to one physical) - kreira više virtuelnih objekata na osnovu jedne instance fizičkog objekta, a ti virtuelni objekti zatim dele pristup fizičkom objektu
    b. **Agregacija** (one virtual to many physical) - kreira jedan virtuelni objekat objedinjujući više fizičkih objekata
    c. **Emulacija** - konstruiše virtuelni objekat određenog tipa koristeći fizičke objekte drugog tipa (npr. SSD emulira ram)
    d. **Kombinovanje multipleksiranja i emulacije** (npr. virtuelna memorija straničenjem multipleksira RAM i disk, a virtuelno adresiranje emulira stvarne memorijske adrese)
52. Uloga **Hypervisor-a (VMM - Virtual Machine Monitor)** je da obezbedi emulacija stvarnih hardverskih resursa (CPU, IO, memorija, skladište, mreža). Može biti instaliran kao aplikacija na OS ili direktno na hardverskoj konfiguraciji. Hypervisor omogućava:
    a. Da više resursa dijele istu hardversku platformu
    b. Migraciju aktivnih VM sa jednog hardvera na drugi
    c. Modifikaciju sistema tokom održavanja
    d. Kompatibilnost unazad sa originalnim sistemom
    e. Forsira izolaciju između VM i time poboljšava bezbednost
53. Nivoi interfejsa kojima se koristi aplikativni sloj:
    a. **Application Programming Interface (API)** - skup instrukcija koje hardver može da izvršava i omogućava aplikacijama pristup ISA. Sadrži biblioteke

razvijene u višim programskim jezicima koje se oslanjaju na sistemske pozive tokom interakcije sa hardverom

   b. **Application Binary Interface (ABI)** - omogućava aplikacijama i bibliotekama da koriste hardverske resurse. Pristup obavlja posredno preko sistemskih poziva, ne sadrži pozive privilegovanih sistemskih instrukcija
   c. **Instruction Set Architecture (ISA)** - granica između hardvera i softvera. Mašinski kod koji se dobija kompajliranjem za specifične ISA i specifični OS nije prenosiv

54. Razlika između hipervizora tipa 1 i tipa 2 je to što se tip 1 izvršava se direktno na hardverskoj platformi, dok se tip 2 pokreće na host OS-u.
55. Da bi se uspešno postigla virtuelizacija procesora i memorije, VMM bi trebalo da:
   a. "hvata" privilegovane instrukcije koje izvršava gostujući OS i obezbeđuje korektnost i bezbednost operacija
   b. "hvata" prekide i prosleđuje ih gostujućim operativnim sistemima
   c. kontroliše upravljanje virtuelnom memorijom
   d. održava paging tabelu "u senci" (shadow page table) za svaki gostujući OS i replicira bilo koju promenu koju gostujući OS napravi u sopstvenoj paging tabeli (shadow page table sadrži pokazivače na stvarne memorijske lokacije i koristi se za dinamičko prevođenje adresa)
   e. nadgleda performanse sistema i vrši korektivne akcije kako performanse sistema ne bi bile ugrožene
56. Uslovi za uspešnu virtuelizaciju su:
   a. Program koji se izvršava u okruženju koje kontroliše VMM ima isto ponašanje kao kada bi se izršavao direktno na ekvivalentnoj hardverskoj platformi
   b. VMM ima potpunu kontrolu nad virtuelizovanim resursima
   c. Značajan deo mašinskih instrukcija se izvršava bez potrebe da VMM izvrši intervenciju nad njima
57. Izazov virtuelizacije x86 arhitekture je taj što se ona sastoji iz 4 sloja privilegija izvršavanja (rings). U 0. prstenu je OS, u 3. prstenu su korisničke aplikacije. Pitanje je gde izvršavati VMM ? Rešenje je da se VMM postavi na 0. prsten, a OS pomeri na 1. prsten.

    Još jedan izazov virtuelizacije ove arhitekture je što se samo privilegovane mašinske instrukcije lako virtuelizuju, dok neprivilegovane i osetljive teže.
58. **Puna virtuelizacija** je jedna od tehnika virtuelizacije x86 arhitekture. Gostujući OS se može neizmenjen izvršavati na VM pod kontrolom VMM. Svaka VM izvršava se na identičnoj kopiji realnog hardvera. Binarno prevođenje dinamički prepisuje delove koda tako da menja osetljive i neprivilegovane instrukcije za odgovarajuće privilegovane i emulira originalne instrukcije. Hipervizor kešira rezultat prevođenja za naknadnu upotrebu, a user mode instrukcije se izvršavaju direktno, istom brzinom kao na hardveru.

    Prednosti: radi i bez hardverske podrške, nisu potrebne modifikacije OS-a, izolacija, bezbednost

    Mane: sporo
59. **Paravirtuelizacija** je jedna od tehnika virtuelizacije x86 arhitekture, potpomognuta od strane OS-a. Zahteva modifikaciju OS kernela da bi se instrukcije koje nisu pogodne za virtuelizaciju zamenile hiperpozivima koji direktno komuniciraju sa slojem za virtuelizaciju.

Prednost: brzo, manji troškovi virtuelizacije
Mane: smanjena portabilnost

60. **Hardverski podržana virtuelizacija** x86 arhitekture je režim izvršavanja instrukcija na CPU koji omogućava da se VMM izvršava u root mode režimu, ispod prstena 0. Privilegovane i osetljive instrukcije se ovde hvataju automatski, od strane hipervizora.
    Prednosti: baš brzo
    Mane: malo ih je, ali uglavnom cena
61. (**Tipovi virtuelizacije**) Virtuelizovati se mogu:
    a. **Network** - kombinovanje raspoloživih mrežnih resursa u virtuelnu mrežu podelom propusnnog opsega fizičke mreže, gde je svako virtuelizovano okruženje nezavisno od ostalilh
    b. **Storage** - objedinjavanje fizičkih kapaciteta za skladištenje podatka u prividno jedinstven smeštajni kapacitet koji se može deliti i dodeljivati korisnicima
    c. **Server** - skriva od korisnika kompleksnost upravljanja serverskim instancama, a istovremeno omogućava efikasno deljenje hardverskih resursa
    d. **Data** - apstrakcija tehničkih detalja upravljanja podacima (lokacija, format zapisa, način pristupa) povećava dostupnost i otpornost sistema za gubitak podataka (zbog replikacije)
    e. **Desktop** - virtuelizacija radnih stanica (desktop mašina) koje mogu da se pokreću na udaljenim mašinama, a pristupa im se preko “tankih klijenata”. Radne stanice se pokreću na serverima u data centrima, pa ovo omogućava bolju administraciju i zaštitu pristupa
    f. **Application** - apstrakcija aplikativnog sloja operativnog sistema, tako da se aplikacije mogu izavršavati u zatvorenom okruženju, nezavisno od OS-a na kom se pokreću
62. NIST referentni model identifikuje nekoliko ključnih entiteta u Cloud computingu
    a. Service consumer
    b. Service provider
    c. Carrier - obezbeđuje povezivanje i transport podataka
    d. Broker - upravlja upotrebom, performansama i isporukom servisa i uspostavlja odgovarajući odnos između consumera i providera
    e. Auditor - revizor, obezbeđuje nezavisnu reviziju usluga, informacije o statusu i funkcionisanju sistema i obezbeđuje sigurnost usluga na cloudu
63. Globalna infrastruktura AWS-a se sastoji od
    a. Geografskih zona (24) gde svaka sadrži dve ili više availability zona
    b. Availability zone (oko 72) - može se sastojati od jednog ili više Data centara gde je svaki opremljen redudantnim napajanjem i mrežnom infrastrukturom
    c. Edge locations - endpoint za AWS koji služe za keširanje sadržaja kako bi se krajnjim korisnicima smanjilo vreme učitavanja sadržaja i povećala brzina isporuke
64. Ključni servisi AWS-a su
    a. EC2 serverske instance - omogućuju korisniku pristup virtualnom računaru na kom mogu da pokreću sopstveni softver. Skalabilne su
    b. SQS - servis za upravljanje porukama, omogućava decentralizovano slanje, skladištenje i primanje poruka između aplikacija ili klijenta i aplikacija
    c. CloudWatch - servis za nadgledanje i upravljanje AWS resursima. Omogućava logovanje, praćenje performansi aplikacija, alarmiranje i rešavanje nastalih problema.

    d. Management console - omogućava kreiranje, konfigurisanje i upravljanje AWS resursima (EC2, S3 skladišta, baze podataka)
    e. S3 Simple Storage Service - služi za skladištenje podataka u oblaku u vidu različitih vrsta datoteka (slike, video, dokumenti, baze podataka), kako bi oni bili dostupni putem interneta
    f. EBS - servis za skladištenje blokova podataka vezanih za EC2 instance.
    g. Razne vrste baza
65. Google-ovi cloud servisi: Storage, Compute, Machine learnig i Big Data servisi
66. MS Azure platforma se sastoji od:
    a. Connect komponente za povezivanje lokalne infrastrukture sa Azure cloud-om
    b. Applications and data komponente za razvoj, postavljanje i upravljanje aplikacijama u cloud-u
    c. CDN komponente koja služi za keširanje i distribuciju sadržaja korisnicima širom sveta
    d. Compute komponente koja se sastoji od virtuelnih mašina i ostalih servisa za obradu podataka i izvršavanje aplikacija u cloud-u
    e. Skladišta (Blob storage, queue storage, table storage)
    f. Fabric controller komponenta služi za upravljanje resursima i obezbeđuje njihovu dostupnost i skalabilnost
67. AWS S3 (Secured Storage Service) obezbeđuje prostor za sigurno smeštanje fajlova. Zasnovano je na objektima tj. ne postoji hijerarhija i podešavanja blokova podataka. Pogodno je za skladištenje nestrukturiranih podataka kao što su slike, video snimci, dokumenti itd.

    Podaci se čuvaju na više uređaja i više lokacija. Fajlovi mogu biti veličine max 5TB i smeštaju se u buckete (folder na cloudu).

    S3 objekat sadrži kljuc (naziv), vrednost (podaci), ID verzije, metapodatke i subresurse

    Konzistentnost: fajl je dostupan za čitanje odmah nakon potvrde uspešnog uploada. Pri izmeni ili brisanju fajlova neophodno je određeno vreme da promene postanu vidljive svima.

    S3 Lifecycle Management omogućava da rasporedimo objekte po slojevima (6 slojeva). Moguće je i postaviti pravila za proglašavanje objekta isteklim nakon nekog vremena. Podržano je verzioniranje objekata, enkripcija, multifactor autentikacija kod brisanja objekta iz bucketa, i obezbeđena je zaštita podataka.

    S3 nudi usluge u različitim klasama: Standard, Infrequently Accessed, One Zone Infrequently Accessed, Intelligent Tiering, Glacier, Glacier Deep Archive (najsporiji). Naplata se vrši prema količini podataka, broju pristupa, klasi itd

    Bucketi su na početku privatni, a prava pristupa se mogu kontrolisati na nivou njih ili na nivou pojedinačnih fajlova. Podaci se mogu kriptovati tokom prenosa (HTTPS), na server strani (AWS upravlja ključevima) ili na klijent strani.

Verzioniranje omogućava čuvanje svih verzija nekog objekta (izmene i brisanje). Nakon uključivanja, može se isključiti samo brisanjem celog bucket-a. Ukoliko su u bucket-u veliki fajlovi koji se često menjaju onda će bucket biti prevelik.

S3 Object Lock omogućava da se neki objekat ne sme menjati ili brisati (trajno ili u nekom periodu). Režimi: Governance mode (specijalnim korisnicima je dozvoljeno da menjaju ili brišu objekat), Compilance mode (niko ne može ništa menjati, ni root user)

S3 prefiksi - sve između imena bucketa i imena objekta. Korišćenjem njih aplikacija može da obradi više zahteva u sekundi jer S3 ima određen broj zahteva u sekundi po prefiksu.

KMS kvote se koriste za ograničavanje kriptografskih operacija (kreiranje ključeva, zahtevi za dekripciju…) u određenom vremenskom periodu što predstavlja dodatni nivo zaštite od zlonamernih akcija. Trenutno se ne mogu povećati na zahtev.

Multipart upload paralelizuje upload i preporučljiv je za fajlove preko 100MB, a obavezan preko 5GB. S3 Byte range fetches paralelizuje download tako što se specificira opseg bajtova na kojima se fajl deli na segmente. Neuspešni segmenti će ponovo biti download-ovani.

S3 select ubrzava preuzimanje podskupa podataka koji teba aplikaciji (umesto preuzimanja celog skupa). Glacier Select radi nad objektima u Glacier arhivama.

Deljenje bucket-a između AWS naloga je moguće uz postojanje AWS organizacije (skupa naloga koji su pod glavnim nalogom). Preporuka je da glavni nalog služi samo za upravljanje i obračun troškova, a ostali nalozi za instalirnje aplikacija.

Načini za deljenje bucketa:

- korišćenjem Bucket polisa i IAM: deljenje celog sadržaja Bucket-a, moguć samo programski pristup, ne preko AWS console.
- korišćenjem Bucket ACL i IAM: deljenje pojedinačnih objekata, samo programski pristup, ne preko AWS console.
- Cross Account IAM Roles: moguće i programski i preko AWS console.

68. AWS IAM omogućava upravljanje korisnicima i određivanje nivoa pristupa korisnicima na Management Console-i tako što je omogućeno kreiranje korisnika, grupa (kolekcija korisnika koji nasleđuju prava dodeljena grupi), uloga (određuju koje AWS resurse mogu da koriste) i prava.

    Prava se specificiraju u JSON dokumentima koji se zovu polise. Postoje predefinisane polise, a mogu se i same kreirati.

69. AWS Cloud front omogućava da se sadržaji isporučuju korisnicima preko distribuirane mreže servera u zavisnosti od geografske lokacije. Umesto direktne komunikacije korisnika sa serverom, korisnik uspostavlja vezu prema lokalnoj Edge Location na kojem se kešira sadržaj što pomaže u smanjenju opterećenja izvornog

servera i povećava brzinu isporuke. Zahtevi od korisnika se automatski rutiraju na najbližu Edge lokaciju i tako se postižu najbolje performanse.

Distribucija se sastoji od više Edge lokacija i može biti jednog od dva tipa: Web distribucija za websajtove i RTMP distribucija za streaming multimedijalnih sadržaja

Na Edge Location se može i slati sadržaj (ne samo čitati) sa vremenom zadržavanja na njemu. Može se i invalidirati poslat sadržaj.

Cloud front distribuciji je moguće ograničiti pristup korišćenjem SignedURL ili SignedCookie

Signed URL se dodeljuje jednom fajlu kada na određenom sajtu treba obezbediti posebna prava pristupa nekom sadržaju određenim korisnicima. Signed cookie se koristi za više od jednog fajla. Na taj URL se takođe doda i vreme isteka, opseg dozvoljenih IP adresa, skup naloga koji mogu da kreiraju URL.

CloudFrontSignedURL - pristup se obavlja preko Cloud Fronta tako što se klijent loguje na tu aplikaciju koja generiše SignedURL i prosleđuje ga korisniku.

S3SignedURL - pristup S3 bucketima se obavlja direktno. Svi korisnici koji imaju URL imaju ista prava kao onaj koji ga je kreirao (IAM user). Pristup je vremenski ograničen.

70. ANSI/TIA-942-2005 standard identifikuje:
    a. Soba za računare – glavni server
    b. Ulazna soba – za izlazne (spoljne) kablove
    c. Main distribution area: glavna soba u data centru, služi za distribuciju električne energije i mreže do HDA i EDA. U njoj se nalaze ruteri, switch-evi, firewall-ovi.
    d. Horizontal distribution area: nalazi se između MDA i EDA. U njoj se nalaze patch paneli. Omogućava protok podataka kroz mrežu, od mrežne opreme u EDA do mrežnih uređaja u MDA.
    e. Equipment distribution area: u nju se smeštaju serveri i sistemi za skladištenje podataka. Nalaze se na kraju horizontalnog kabliranja
    f. Zone distribution area: nalazi se opcionalno između HDA i EDA. Služi za optimizaciju, omogućava prilagođavanje specijalizovane opreme iz EDA sa ostatkom mreže. Koristi se u većim data centrima
71. Tri osnovna sloja- core, aggregation, access pružaju pouzdanost, efikasnost, i skalabilnost.
    a. Core sloj: nalazi se u središtu mreže, sadrži mrežne uređaje za brz i pouzdan prenos podataka. Sadrži switch-eve za prenos velike količine podataka.
    b. Aggregation sloj: nalazi se između core i access sloja i omogućava povezivanje različitih segmenata mreže. Sadrži switch-eve za povezivanje servera i skladišta sa mrežom
    c. Access sloj: nalazi se na rubu mreže i služi za povezivanje klijentskih uređaja sa mrežom (serveri, desktop računari…). Sadrži switch-eve za pouzdan prenos podataka uz različite tehnologije

72. Top of Rack – switch se smešta na vrh svakog ormara, a serveri u tom ormaru se povezuju na njega. Jednostavnija instalacija i održavanje mreže jer su kablovi kraći i lakše ih je povezati. Nadogradnja je skupa i zahteva velike izmene. Ako rack nije popunjen dosta portova je neiskorišćeno.

    End of Row – switch se smešta na kraj ili u sredinu reda. Kablovi su duži, povezivanje je složenije, skalabilnost i nadogradnja su lakšiv

CLOUD PITANJA 2

1. **Šta je upravljanje resursima? Na koje kriterijume za evaluaciju sistema direktno utice?**

   Upravljanje resursima je osnovna funkcija koju obavlja bilo koji sistem koji je napravio čovek. Direktno utiče na tri osnovna kriterijuma za evaluaciju nekog sistema: **performanse**, **funkcionalnost** i **cena**. Neefikasno upravljanje resursima ima direktne negativne posledice na performanse i eksploatacionu cenu sistema, a najčešće indirektno pogađa i funkcionalnost sistema (jer izvršavanje nekih funkcija može postati preskupo ili trajati predugo)

2. **Da bi se postiglo efikasno upravljanje resursima u cloudu kakva optimizacija je neophodna?**

   Upravljanje resursima u oblaku zahteva složene politike upravljanja i odlučivanja koje se zasnivaju na “višeciljnoj“ optimizaciji (multi-objective optimization).

3. **Jedan od primarnih ciljeva upravljanja resursima u oblaku je obezbeđivanje elastičnosti. Koji je ključni izazov sa kojim se upravljanje resursima suočava u ovakvim okruženjima?**

   Ponuđač usluge mora da reši problem obezbeđivanja obećane elastičnosti u situaciji kada se suočava sa velikim, **fluktuirajućim opterećenjem**. U nekim situacijama, kada je povećanje opterećenja predvidljivo alokacija resursa se može planirati unapred.Ako se desi **nagli nepredviđeni skok opterećenja**, autoscaling algoritam može da se upotrebi ali samo ako: A) u tom momentu postoje resursi koji su slobodni ili se mogu osloboditi i dodeliti za izvršavanje usluge i B) postoji dobar monitoring podsistem.

4. **Koja su dva osnovna pristupa (načina) upravljanju resursima?**

   U nekim situacijama, kada je povećanje opterećenja predvidljivo alokacija resursa se može planirati unapred.Ako se desi nagli nepredviđeni skok opterećenja, autoscaling algoritam može da se upotrebi ali samo ako: A) u tom momentu postoje **resursi koji su slobodni** ili se mogu osloboditi i dodeliti za izvršavanje usluge i B) postoji dobar **monitoring podsistem**.

5. **Zašto je centralizovani pristup upravljanju resursima tema polemika od nastanka cloud okruženja?**

   Skoro od samog nastanka sistema u oblaku polemiše se da centralizovana kontrola teško da može da se izbori sa ovim brzim i nepredvidljivim promenama i da obezbedi kontinualnu isporuku servisa sa odgovarajućim stepenom kvaliteta.

6. **Koje su osnovne grupe politika upravljanja resursima u oblaku? Šta je cilj svake od njih?**

Politika definiše donošenje ključnih odluka, a mehanizam predstavlja način kako se te ključne odluke sprovode u delo (impelmentiraju).

a. **Kontrola prihvatanja** (admission control) - sprečava sistem da prihvati rad na zadacima (workload) koji narušavaju postavljene politike sistema na visokom nivou. (Npr. ne prihvatati novi posao ukoliko to sprečava da se tekući ili već pripremljeni zadaci završe)
b. **Alokacija kapaciteta** (Capacity allocation) - alociranje resursa za pojedinačnu aktivaciju servisa. Kada se stanje svakog sistema često menja, ovo podrazumeva pretraživanje preko velikog broja čvorova da se nađe odgovarajući.
c. **Balansiranje opterećenja** (Load balancing) - distribuiranje opterećenja ravnomerno između raspoloživih instanci servera
d. **Optimizacija utroška energije** (Energy optimisation) - minimizovanje energije utrošene na izvršavanje zadataka
e. **QoS garancije** - sposobnost da se ispune vremenski ili drugi zahtevi specificirani u ugovoru - Service Level Agreement (SLA)

7. Šta je problem raspoređivanja (scheduling) u računarskom sistemu?

To je odlučivanje kako dodeliti resurse sistema (vreme na procesoru, memoriju, prostor za skladištenje, I/O i mrežni propusni opseg) različitim zadacima i korisnicima

8. Šta je "raspoređivač" (scheduler) i koja je njegova funkcija?

Scheduler - program koji implementira određeni algoritam raspoređivanja resursa. Raspoređivanje(scheduling) je kritična komponenta upravljanja resursima u oblaku. Odgovorno je za efikasno deljenje/mulitpleksiranje resursa na različitim nivoima.

9. Koje su ključne odluke koje raspoređivač donosi u pogledu resursa?

a. Količini resursa koje je neophodno dodeliti nekome
b. Trajanju alokacije

10. Šta su ključne željene karakteristike algoritma za raspoređivanje resursa?

a. efikasan
b. pravedan
c. sprecava izgladnjivanje (starvation-free)

11. Koji tipovi vremenskog ograničenja za izvršavanje zadataka se najčešće identifikuju? Za koje aplikacije je primerena određena vrsta vremenskog ograničenja?

Postoje tvrda i meka ograničenja. Tvrda ograničenja se primenjuju **kod aplikacija u realnom vremenu** jer je bitna brza i precizna reakcija. Ukoliko se ne poštuju postoje kazne. Meka ograničenja se primenjuju **u multimedijalnim aplikacijama** gde preciznost nije toliko bitna, već je bitno korisničko iskustvo. Ograničenja su fleksibilna i ne postoje kazne.

12. Kada se radi raspoređivanje na VM, koje dodatne faktore treba uzeti u obzir?

Kako bi se obezbedile optimalne performanse i izolacija resursa između zadataka treba uzeti u obzir da su ti zadaci raznoliki i svaki treba da dobije potrebne resurse, da je broj zadataka koji se može obaviti na serveru ograničen, da su vremena izvršavanja zadataka različita i da treba da postoji balans i izolacija resursa između zadataka kako bi se izbegli konflikti.

13. **Šta je Proportional Share raspodela resursa?**
    Svakom zadatku se dodeljuje broj (udeo/shares) tako da raspoređivač vrši raspodelu resursa u skladu sa tim brojem (proporcionalno)
14. **Šta je Fair Share raspodela resursa?**
    Vremenom se vrši prilagođavanje raspodele resursa kako bi se obezbedila pravičnost tj. dobijanje resursa na osnovu merenja stvarnog korišćenja umesto po broju *shares.* Ova razlika između stvarno korišćenih resursa i zatraženih/prvobitno dobijenih resursa se zove greška alokacije
15. **Šta je Work Conserving, a šta Non Work Conserving režim rada raspoređivača?**
    Work Conserving režim rada: svaki zadatak koji ima bilo koji udeo će se izvršavati na CPU sve dok se ne završi, iako je možda već iskoristio vreme predviđeno za njega
    Non Work Conserving režim rada: udeo zadatka predstavlja ograničenje za vreme pristupa CPU i ne sme da se prekorači
16. **Šta je Preemptive, a šta Non Preemptive metod funkcionisanja raspoređivača?**
    Preemptive raspoređivači imaju mogućnost da čim se pojavi klijent sa većim prioritetom, CPU se prebacuje na njega bez obzira da li je trenutni klijent završio ili ne.
    Non Preemptive raspoređivači dozvoljavaju trenutnom klijentu da završi korišćenje CPU-a ili dobrovoljno oslobodi CPU i tek onda se CPU prebacuje na novog klijenta.
17. **Šta je Workload management (WM)? Koja su dva osnovna pristupa WM?**
    Workload management je sposobnost sistema da **precizno dodeljuje resurse** aplikacijama u skladu sa njihovim zahtevima za performansama. Postoje statički i dinamički pristup. U statičkom pristupu potreba za resursima se procenjuje samo na početku, pri konfigurisanju, a u dinamičkom resursi se prilagođavaju trenutnim zahtevima aplikacija i tako se raspoređuju.
18. **Šta je BVT algoritam? Borrowed Virtual Time**
    BVT algoritam dodeljuje CPU onim nitima čija VM ima najmanje stanje brojača virtuelnog vremena. Kada nit sa velikim brojačem zatreba resurse, može pozajmiti vreme procesora koje će mu biti dodeljeno u budućnosti i tako dobiti na prioritetu. Ovaj algoritam se koristi u Preemptivnom i Work Conserving režimu - CPU se prebacuje na nove, ali će svi biti završeni kad tad. CPU dodeljuje vreme nitima takod da svaka dobije fer deo resursa u skladu sa prioritetom i zahtevima. Jednostavam je za implementaciju. Ne podržava NWC režim rada što znači da ne može uticati na izolaciju između niti (jedna mora da čeka neku drugu)
19. **Šta je SEDF algoritam? Simple Earliest Deadline First**
    Svaki zadatak specificiraju tri vrednosti (si, pi, xi).
    - Pi - vremenski interval u kojem se zadatak ponavlja.
    - Si - vreme izvršavanja zadatka u okviru jednog perioda
    - Xi flag - određuje da li je omogućen WC mode tj. da li se dobija ekstra vreme

    Algoritam daje prioritet zadacima na osnovu njihovih rokova (deadline) i očekivanog preostalog vremena izvršavanja. Duži i sa bližim rokom imaju prioritet.
    Algoritam se koristi u Preemptivnom režimu rada (i WC i NWC). Raspodela je pravična ako su dužine perioda zadataka slične, ako nisu onda će zadaci sa manjim periodom češće imati

CPU. Mana mu je što ne može da se prilagodi i uspostavi balans u multiprocesorskim okruženjima.

20. Šta je Credit Scheduler algoritam?

Obezbeđuje da nijedan fizički CPU ne bude u idle stanju dok ima posla za izvršavanje na drugim CPU-ovima (njihovi virtualni CPU-ovi) i vodi pravičnu raspodelu između resursa CPU-a i vCPU-ova. Koristi se u virtualizovanim okruženjima. Raspoređivač se prebacuje između virtualnih CPU-ova kako bi bolje iskoristio raspoložive CPU resurse. Svaka virtualna mašina ima dodeljenu težinu(prioritet) i limit. Limit predstavlja odsečak vremena koje VM može dobiti na procesoru (NWC). Ako je limit nula onda je reč o WC režimu rada.

Algoritam radi u Non preemptive režimu (i WC i NWC). Za razliku o SEDF-a, rešava problem globalnog opterećenja na multiprocesorskim sistemima

21. Koji su ključni ekonomski faktori koji čine aplikacije u oblaku prihvatljivim za korisnike?

- Veoma male početne investicije u infrastrukturu
- Korisnik može da ubrza izvršavanje upotrebom paralelizacije. Ukoliko aplikacija može podeliti podatke ili računarski problem na n delova, može se dobiti ubrzanje blisko n.
- Programeri takođe mogu da razvijaju aplikaciju fokusirajući se na funkcionalne zahteve, a ne na ograničenja sistema na kome će se izvršavati. Elastičnost u oblaku omogućava da takve aplikacije obezbede dodatne resurse kada im ustrebaju (just-in-time-infrastructure), i to bez dodatnog angažovanja programera.

22. Koji su ključni ekonomski faktori koji čine aplikacije u oblaku prihvatljivim za ponuđače?

- tipično im omogućava znatno bolje korišćenje resursa
- ponuđači imaju sposobnost da privuku veliki broj korisnika nudeći dobra rešenja za bezbednost aplikacija, skalabilnost, pouzdanost, QoS, i ispunjavanje SLA
- Računarstvo u oblaku je istovremeno usmereno I ka velikim poslovnim rešenjima (enterprise), što ih jasno izdvaja u odnosu na prethodna grid rešenja koja su primarno bila projektovana za naučne ili inženjerske proračune I aplikacije.

23. Koja je ključna osobina aplikacija koje su izrazito pogodne za iskorištavanje prednosti okruženja u oblaku?

Kada aplikacija može razdeliti posao u segmente proizvoljne veličine koji se mogu paralelno procesirati na serverskim instancama koje su trenutno dostupne. Upravo ovakve aplikacije (arbitrarily divisible workloads) su vrlo pogodne za računarstvo u oblaku.

24. Koji su ključni izazovi za aplikacije u oblaku?

- Moraju da se izbore sa debalansom između računarskih resursa, brzine I/O i ograničene propusnosti komunikacionih kanala. Sama veličina sistema u oblaku čini ove izazove potencijalno mnogo većim.
- Deljenje resursa može u nekim momentima biti i dodatni izazov i imati negativan efekat. Izolacija performansi je skoro nedostižna u realnim sistemima, pogotovo kada su VM koje dele hardverske resurse pod vrlo visokim opterećenjem. Bezbednosna izolacija je takođe veliki izazov za višekorisničke (multitenant) sisteme.

- Pouzdanost – otkazi čvorova se mogu očekivati kada god se radi o kompleksnom sistemu sa velikim brojem čvorova. Izbor optimalne instance od onih koje su ponuđene je takođe vrlo bitan faktor za uspešan rad aplikacije. Troškovi korišćenja pojedinih instanci.
- Skladištenje podataka - Organizacija skladištenja podataka, lokacija čuvanja i propusni opseg prilikom pristupa skladištu podataka
- Problem logovanja (gde, koliko, kako ograničiti količinu podataka koji se loguju).

25. Koji su tipične grupe aplikacija u oblaku?

- Processing pipelines
- Batch processing
- Web aplikacije
- (mozda ne mora: u poslednje vreme sve češće su i aplikacije za:
    - Biznis analizu
    - Paralelizona batch obrada – map reduce
    - Apliakacije koje prikupljaju i obrađuju podatke koji pristižu sa raznorodnih senzorskih uređeja)

26. Ključna osobina distribuiranih, pa i aplikacija u oblaku je koordinacija aktivnosti. Koji načini postoje za koordinisanje aktivnosti?

Radni tokovi (workflows) predstavljaju model za opis koordinisanog izvršavanja velikog broja nezavisnih zadataka, u predviđenom redosledu.

Vrste workflow-a:

- Statički – lako se mogu modelvati nekim od WFDL
- Dinamički – podrazumevaju mogućnost izmene workflowa tokom izvršavanja

Po pitanju načina koordinacije tokom izvršavanja:

- Strong coordination models – postoji kooordinacioni/kontrolni proces
- Weak coordination models – koordinacija se obavlja peer-to-peer komunikacijom bez posebnom koordinacionog procesa

27. Šta je ideja koordinacije na principu konačnog automata?

Ukoliko imamo složenu aplikaciju, u kojoj imamo servere spcijalizovane za obavljanje određenih zadataka, neophodno je obezbediti koordinisani rad servisa na ovim instancama kako bi aplikacija funkcionisala. Jedno od rešenja je da se servis sagleda kao konačni automat (finite state machine), koji obavlja određene operacije kada od klijenta dobije komandu, i izvršavanje operacije ga dovodi u sledeće definisano stanje.

28. Šta je ZooKeeper?

ZooKeeper je distribuirani koordinacioni servis koji zasniva na modelu konačnog automata. ZooKeeper se downloaduje i instalira ne više servera, nakon čega klijenti mogu ostvariti konekciju i pristupiti koordinacionom servisu. Servis je dostupan sve dok je većina servera u grupi dostupna.

29. Koji su osnovni principi funkcionisanja ZooKeeper-a i koje garancije on pruža?

Osnovni princpi:

- Sistem je organizovan kao deljeni, hijerarhijski prostor imena, slično fajl sistemu
- Naziv je sekvenca elemenata putenje (razdvojenih sa /)
- Svaki naziv u ovom prostu imena je identifikovan jedinstvenom putanjom.
- Znode-ovi ZooKeeper-a mogu sadržavati podatke koji su namenjeni za čuvanje stanja (verizja podataka, timestampovi promene u ACL)
- Klijent može postaviti watch na neki znode, i tako primiti notifikacije kada se stanje znode-a promeni.

ZooKeeper garantuje:

1. Atomičnost – transakcija ili uspeva ili ne
2. Sekvencijalnu konzistentnost update-a. Updatei se primenjuju striknto u redosledu u kom su nastali.
3. Klijenti vide jednu sliku sistema. Dobiće isti odgovor bez obzira na koji server iz grupe su ostvarili konekciju.
4. Trajnost update-a – jednom snimljen trajno se čuva dok klijent ne izazove izmenu podataka.
5. Pouzdanost – sistem garantuje korektno funkcionisanje sve dok je većina servera iz grupe u funkciji.

30. Šta je MapReduce programski model?

MapReduce je programski model za grupnu obradu. Omogućava ubrzavanje obrade velikih skupova podataka. Omogućava nam da izrazimo jednostavne proračune (obrade) koje smo pokušavali da izvršimo, ali skriva nezgodne detalje paralelizacije, tolerancije kvarova, distribucije podataka i raspoređivanja opterećenja. Nastaje iz potrebe za mnogim izračunavanjima u odnosu na velike/ogromne skupove podataka. Većina ovih izračunavanja je relativno jednostavna. Da bismo ubrzali izračunavanje i skratili vreme obrade, možemo da distribuiramo podatke preko većeg broja mašina i paralelno ih obrađujemo. Idealno bi bilo da paralelno obrađujemo podatke, ali ne i da se bavimo kompleksnošću paralelizacije i distribucije podataka.

31. Koje su ključne komponente u MapReduce programskom modelu?

Ključne komponente u MapReduce su zadatak mapiranja (MapTask: jedan par u listu parova međurezultata) i zadatak redukovanja (ReduceTask: svi međurezultati sa istim ključem u listu izlaznih vrednosti).

32. Šta je uloga mastera u MapReduce? Šta je uloga Workera?

Master čvor prosleđuje lokacije regiona u kome se čuvaju međurezultati od mapera ka reducerima (dodeljivanje, monitoring zadataka, upravljanje procesom MapReduce-a). Ova informacija se inkrementalno (kako koji mapper završi posao) dostavlja onim worker čvorovima koji imaju in-progress reduce zadatke. Worker čvorovi imaju ulogu da izvršavaju dodeljen posao od master čvora.

33. Kako MapReduce rešava problem otkaza čvorova?

Postoje dva tipa otkaza:

a. Otkazi worker čvorova: identifikuju se preko “otkucaja srca” - signala koje master čvor šalje. Ukoliko ne stigne odgovor, smatra se da je worker čvor otkazao.

In-progress i completed map zadaci i in-progress reduce se preraspoređuju, a worker čvorovi koji izvršavaju reduce taskove koji su afektirani otkazom map čvora se obaveštavaju o preraspoređivanju.

b. Otkazi master čvora: vrlo su retki, a oporavak je moguć iz checkpointa. Ideja je da se prekine trenutna obrada i počne iznova.

34. Zašto je lokalnost podataka bitna za performanse MapReduce programskog modela?

Mrežni komunikacija - propusni opseg je relativno "redak" resurs, a I povećava kašnjenje. Cilj je štedeti mrežni propusni opseg. Korišćenje GFS tipično zahteva tri kopije na tri različite mašine. Zbog toga se zadaci mapiranja raspoređuju tako da budu na istim čvorovima na kojima su i podaci lokalno smešteni (ukoliko je to moguće) ili na čvoru koji je mrežno blizu samim podacima.

35. Koji mehanizam MapReduce programski model koristi da bi poboljšao performanse kada u mreži postoje čvorovi sa problemima u obradi (stragglers)

Spori radni čvorovi (stragglers) izazivaju kašnjenje cele obrade (problem sa diskovima, drugi zadaci na mašini, neodgovarajuća konfiguracija mašine). Kada je MapReduce obrada blizu kraja master raspoređuje i spekulativne rezervne (backup) obrade preostalih in-progress zadataka. Zadatak se potom označava završenim kada ili primarni ili rezervni čvor prijave završetak, koji god to uradi pre.

36. Koje dodatne izazove donosi upotreba MapReduce programskog modela na velikim heterogenim klasterima?

Osnovni način rešavanja problema sa strugglerima kreiranje spekulativne rezervne kopije nije dovoljan i u velikim heterogenim okružejnima. Glavni problem je šarolikost performansi pojedinih čvorova, što nepovoljno utiče na pretpostavke s Kojima raspoređivač radi pri dodeli zadataka. Da bi se smanjilo vreme odziva, uvodi se noviji LATE raspoređivač.

37. Od kojih pretpostavki o načinu funkcionisanja čvorova se polazi u homogenim okruženjima?

Polazi se od sledećih pretpostavki:

a. Čvorovi mogu obavljati posao približno istom brzinom
b. Zadaci napreduju konstantnom brzinom tokom svog izvršavanja
c. Nema "troškova" pokretanja spekulativnog zadatka
d. Progres pojedinog zadatka je približno jedan njegovom udelu u ukupnom poslu
e. Zadaci imaju tendenciju da se završavaju u "talasima", pa onaj zadatak koji ima nizak indicator progresa je verovatno usporen
f. Različiti zadaci iste kategorije sadrže okvirno sličnu količinu posla

38. Koje pretpostavke o načinu rada čvorova ne važe u heterogenim virtuelizovanim okruženjima?

U heterogenima ne važe ključne pretpostavke o sličnoj brzini čvorova i konstantnoj brzini izvršavanja zadataka - neki čvorovi su sporiji od ostalih (starija konfiguracija); virtualizovani klasteri pate od tzv. interferencije na deljenom resursu. I pretpostavke o nepostojanju troškova spekulativnih izvršavanja, proporcionalnom udelu i završetku u talasima u heterogenim okruženjima nisu realne - previše spekulativnih zadataka oduzima resurse drugim zadacima; faza kopiranja kod redukovanja je najsporija, a ona se računa kao 1/3 posla faze redukovanja; zadaci iz različitih "generacija" mogu se izvršavati konkurentno, što dovodi do toga da se noviji (brži) zadaci pri raspoređivanju evaluiraju zajedno sa starijim – srednji indicator progresa se uveliko menja.

39. **Za koje tipove okruženja je progress score dovoljno dobra mera za odlučivanje o spekulativnim zadacima?**
    Za homogena okruženja.
40. **Koja mera se koristi za raspoređivanje spekulativnih zadataka u heterogenim virtualizovanim okruženjima?**
    Umesto indikatora napretka - progress score values, izračunavati stopu progresa - progress rates, i napraviti backup zadatke za one koji su dovoljno ispod srednje vrednost.
41. **Koja je osnovna ideja LATE raspoređivača zadataka?**
    Osnovna ideja je: napraviti backup za zadatke kod kojih je procenjeno vreme završetka najduže.
42. **Zašto je na velikim klasterima, na kojima se pokreću različite vrste aplikacija nephodno imati dodatni servis koji upravlja dodelom resursa aplikacijama? Zašto oslanjanje na raspoređivač određene vrste aplikacija ne bi bilo dobro rešenje za ostale?**

    Veliki klasteri često podržavaju mnoštvo različitih aplikacija, svaka sa svojim jedinstvenim zahtevima za resurse i karakteristikama performansi. Svaka aplikacija ima specifične potrebe i očekivanja kada je reč o dodeli i upotrebi resursa, a ove potrebe se često međusobno razlikuju.

    Na primer, aplikacije za obradu podataka u realnom vremenu mogu zahtevati brz pristup memoriji i procesoru, dok aplikacije za skladištenje podataka možda više vrednuju diskovni prostor ili mrežnu propusnost. Raspoređivač određene vrste aplikacija može biti optimizovan za te specifične zahteve, ali ne bi bio efikasan za raspoređivanje resursa za aplikacije sa drugačijim potrebama.

    Stoga je, na velikim klasterima, važno imati dodatni servis koji upravlja dodelom resursa, umesto da se oslanja na jedan raspoređivač. Ovaj servis za upravljanje resursima može pružiti prilagođeno i optimizovano raspoređivanje za svaku aplikaciju, uzimajući u obzir njene specifične potrebe. Pored toga, može omogućiti bolje upravljanje ukupnim resursima klastera, poboljšavajući efikasnost i minimizirajući otpad.

43. **Šta je statička podela klastera? Dobre strane? Loše strane?**
    a. Statička podela klastera se odnosi na proces gde se resursi klastera podele unapred i dodeljuju specifičnim aplikacijama ili servisima. Ova podela se obično ne menja dinamički na osnovu promenljivih zahteva ili uslova.

    b. Dobre strane statičke podele klastera obvuhvataju:
        i. **Predvidljivost** - Aplikacije ili servisi znaju tačno koliko resursa imaju na raspolaganju i mogu to koristiti u svom planiranju i operacijama.
        ii. **Smanjena kompleksnost** - Upravljanje resursima može biti jednostavnije, pošto nema potrebe za složenim algoritmima za raspoređivanje ili odlučivanje u realnom vremenu.
        iii. **Izolacija** - Može smanjiti međusobno ometanje različitih aplikacija ili servisa, jer svaka ima svoje namenske resurse.

    c. Nedostaci statičke podele klastera obuhvataju:

i. **Neefikasnost resursa** - Statička podela može rezultovati lošim iskorišćenjem resursa, jer jedan tip aplikacije možda neće koristiti sve dodeljene resurse sve vreme, dok druge aplikacije možda imaju veće potrebe.
ii. **Nepodudaranje sa realnim potrebama** - Tvrd princip particionisanja možda se ne poklapa sa promenljivim potrebama aplikacija u realnom vremenu. Statička podela ne omogućava dinamičko prilagođavanje na promene u radnom opterećenju ili zahtevima za resursima.
iii. **Nepodudarnost granularnosti** - Granularnost statičkih particija - to jest, veličina i broj particija - možda se ne poklapa sa potrebama trenutno aktivnih aplikacija. Na primer, ako aplikacija zahteva više resursa nego što je dostupno u jednoj particiji, ili ako je broj aktivnih aplikacija veći od broja particija, to može dovesti do problema sa performansama ili iskorišćenjem resursa.

44. Šta je Mesos? Koji je princip rada Mesos-a?

a. Ideja Mesos-a je da omogući izvršavanje različitih okruženja (tipova aplikacija) na jednom klasteru. Mesos je “tanak” deljeni sloj koji omogućava deljenje resursa na “finom nivou granularnosti” između različitih okruženja - okruženjima pruža jedinstveni interfejs preko koga one pristupaju resursima u klasteru
b. Izazovi i ciljevi Mesos-a su:
    i. **Visoka iskorištenost resursa**
    ii. **Podrška za različita okruženja** - svako okruženje može imati različit princip raspoređivanja zadataka
    iii. **Skalabilnost** - Sistem raspoređivanja zadataka mora se skalirati da radi sa klasterima od hiljada nodova koji izvršavaju stotine poslova koji se dele na milione pojedinačnih zadataka
    iv. **Pouzdanost** - Pošto će sve aplikacije biti zavisne od Mesosa on sam mora biti otporan na otkaze i visoko dostupan
c. Elementi dizajna sistema Mesos-a:
    i. **Deljenje resursa na finom nivou granularnosti** - Alokacije se radi na nivou pojedinačnog zadatka unutar nekog posla i time se poboljšava iskorištenje resursa, smanjuje latencija i povećava stepen lokalnosti podataka
    ii. **Nuđenje (ponuda) resursa** - Ponudi raspoložive resurse izvršnim okruženjima aplikacija čime se dobija jednostavan, skalabilan mehanizam raspoređivanja zadataka koji je upravljan od strane pojedinačnih aplikacije:
        1. **Mane** - Kao i bilo koje decentralizovano rešenje, ne nudi optimalne karakteristike
        2. **Prednosti** - Jednostavan i proširiv za buduća okruženja za izvršavanje aplikacija

45. Koje su osnovne komponente Mesos-a?

a. **Mesos master** - Upravlja deljenjem resursa između različitih okruženja na finom nivou granularnosti. “Mesos master utvrđuje koliko resursa da ponudi svakom

pojedinačnom okruženju, a mehanizmi raspoređivanja pojedinačnih okruženja određuju koji od ponuđenih resursa se koristi (i za šta)”

b. **Mesos podređeni čvor (slave) na svakom čvoru**
c. **Frameworks** - Aplikativno okruženje koje izvršava zadatke pojedinačne aplikacije na svakom podređenom nivou
d. **Framework schedulers** - Aplikativno specifični system za raspoređivanje zadataka

46. Koja je uloga Mesos mastera, a koja Mesos slave komponente?

a. **Mesos master** - Upravlja deljenjem resursa između različitih okruženja na finom nivou granularnosti. Evo nekoliko ključnih uloga:
    i. **Prihvatanje zahteva** - Prihvata zahteve za izvršavanje aplikacija od korisnika ili framework-ova
    ii. **Raspodela resursa** - Na osnovu dostupnih resursa u klasteru i definisanih politika raspodele, odlučuje koje aplikacije će dobiti koje resurse
    iii. **Planiranje** - Planira raspoređivanje aplikacija na čvorove klastera uzimajući u obzir njihove zahteve za resursima i prioritete
    iv. **Praćenje i nadgledanje** - Nadgleda stanje čvorova u klasteru i prati izvršavanje aplikacija. Ukoliko se pojavi neka greška ili čvor postane nedostupan, Mesos master preduzima odgovarajuće mere kako bi očuvao stabilnost sistema.
    v. **Skaliranje** - Može dinamički skalirati broj Mesos slave čvorova u klasteru, u skladu sa zahtevima i dostupnim resursima.
b. **Mesos slave** - Čvor u Mesos klasteru koji pruža računarske resurse i izvršava aplikacije. Njegova glavna uloga je prijavljivanje dostupnih resursa Mesos masteru i izvršavanje aplikacija koje mu dodeli. Evo nekoliko ključnih uloga:
    i. **Prijavljivanje Resursa** - Prijavljuje dostupne resurse Mesos masteru, i na osnovu tih informacija Mesos master može donositi odluke o raspoređivanju aplikacija.
    ii. **Izvršavanje aplikacija** - Prima instrukcije od Mesos masters o tome koje aplikacije treba pokrenuti i na kojim resursima. On preuzima aplikacije i njihove zadatke za izvršavanje, i upravlja njihovim pokretanjem i praćenjem.
    iii. **Izveštavanje o stanju** - Redovno izveštava Mesos master o svom stanju, kao što su zauzeti resursi, dostupnost i druge metrike. Ovi izveštaju pomažu Mesos masteru da donosi informisane odluke o raspodeli resursa.

47. Da li aplikativni scheduler-i i dalje obavljaju svoju ulogu u Mesos-u?

Aplikativni scheduler-i se koriste za raspoređivanje i izvršavanje zadataka na resurse koje Mesos master pruža. Mesos master upravlja resursima klastera i šalje ponude o raspoloživim resursima registrovanim schedulerima. Ovi scheduleri odlučuju da li će prihvatiti ili odbiti ponude. Ako prihvate ponudu, oni će zatim odlučiiti kako će upotrebiti dodeljene resurse za raspoređivanje i izvršavanje zadataka.

Dakle, **da, aplikativni scheduleri i dalje obavljaju svoju ulogu** u Mesos-u, a ta uloga je

kritična za fleksibilnost i efikasnot koju Mesos pruža.

48. Zašto su za Mesos bitne liste filtera?

Filter je pravilo koje koristi aplikativni scheduler za odlučivanje o tome koje ponude za resurse treba odbiti. Na primer, scheduler može imati filter koji kaže da odbije sve ponude koje ne uključuju najmanje 4 GB RAM-a. To bi moglo biti korisno ako znamo da naša aplikacija neće raditi efikasno sa manje od 4 GB RAM-a.

Ove filter liste omogućavaju Mesosu da optimizuje proces raspoređivanja tako što će smanjiti broj ponuda koje scheduleri odbijaju. Na taj način, resursi mogu biti brže i efikasnije raspoređeni na zadatke koji ih zaista trebaju.

Takođe, Mesos koristi ove filtere da uskladi potrebe aplikacija sa dostupnim resursima u klasteru. Na primer, ako jedan scheduler ima filter koji zahteva velike količine memorije, a drugi scheduler zahteva velike količine CPU-a, Mesos može bolje balansirati upotrebu resursa između ovih dvaju scheduler na osnovu njihovih filtera.

49. Šta je Omega? Da li kod Omega okruženja postoji komponenta koja centralno donosi odluku o dodeli resursa?

a. Omega je kao i Mesos sistem za upravljanje klasterima i raspoređivanje resursa. Neke od mana Mesosa:
   i. **Pesimističko zaključivanje** - Mesos koristi strategiju pesimističkog zaključivanja pri dodeli resursa. Ovo znači da Mesos master pruža ponude resursa samo jednom okruženju u datom trenutku, i to okruženje zadržava resurse dok donosi odluku o tome da li će prihvatiti ponudu. Ovo može da dovede do sporijeg tempa dodele resursa, jer svako okruženje mora da čeka na svoj red da dobije ponudu resursa od Mesos mastera
   ii. **Delimičan uvid u stanje klastera** - U Mesosu, scheduleri nemaju pristup informacijama o celokupnom stanju klastera. Umesto toga, oni dobijaju ponude resursa od Mesos mastera, koje mogu da prihvate ili odbiju. Ovo može da ograniči fleksibilnost scheduler, jer oni ne mogu da donose odluke na osnovu celokupnog stanja klastera
   iii. **Nedostatak autonomije u upravljanju resursima** - U Mesosu, scheduleri ne mogu samostalno da oslobode resurse ili da zauzmu sve resurse. Ove akcije su pod kontrolom Mesos mastera. Ovo može da dovede do manje efikasne upotrebe resursa, jer scheduleri ne mogu da prilagode svoje zahteve za resursima u realnom vremenu na osnovu svojih trenutnih potreba.
b. Naprotiv, u Omegi, scheduleri imaju potpuni uvid u stanje klastera, što im omogućava da nezavisno donose odluke o raspoređivanju zadataka i upravljanju resursima. Omega koristi optimističku konkurenciju, što znači da scheduleri mogu doneti odluke na osnovu zastarelih podataka, ali sistem proverava da li su te odluke validne pre nego što se izvrše. Ovo može da dovede do efikasnijeg upravljanja resursima i bržeg tempa raspoređivanja zadataka. Zaključujemo da kod Omege **nema centralne komponente** koja donosi odluke o dodeli resursa. Umesto toga, svako okruženje koristi svoj scheduler, koji donosi odluke o

raspoređivanju zadataka na osnovu svoje lokalne kopije stanja klastera.

50. Na koji način Omega obezbeđuje informaciju o zauzetim resursima raspoređivačima pojedinih aplikacija?

U Omega okruženju, za upravljanje klasterima, centralna komponenta u procesu dodele resursa je **Cell State.** Cell State je otporna master kopija tabele alokacije resutsa klastera. To znači da ona pruža pouzdan, dosledan prikaz trenutne raspodele resursa unutar klastera.

Ne postoji centralizovani scheduler u Omegi. Umesto toga, svako okruženje koristi svoj scheduler, koji ima svoju privatnu, lokalnu kopiju stanja klastera. Ova kopija se redovno ažurira sa informacijama iz Cell State-a, što omogućava scheduleru da donosi odluke o raspoređivanju zadataka na osnovu najnovijih dostupnih informacija.

Kada scheduler odluči da rasporedi zadatak, on pokušava da ažurira Cell State sa novim rasporedom resursa. Međutim, pre nego što se promene primene, Omega proverava da li su te promene i dalje validne u odnosu na trenutno stanje klastera u Cell State-u. Ako nisu, raspoređivač će morati da ponovo razmotri svoju odluku.

Ovaj model omogućava visok stepen paralelizma i efikasnosti u raspoređivanju resursa, jer svaki raspoređivač može nezavisno da donosi odluke na osnovu svoje lokalne kopije stanja klastera. Istovremeno, korišćenje Cell State-a kao centralne reference za stanje klastera osigurava doslednost i sprečava konflikte u dodeli resursa.

51. Šta je GFS ? Koji je osnovni princip rada?

GFS je Google File Sistem, distribuirani sistem datoteka razvijen krajem 90-ih. Koristi hiljade sistema za skladištenje, izgrađenih od **komercijalno dostupnih komponenti**. Obezbjeđuje **veliko skladište podataka** velikoj korisničkoj zajednici sa raznovrsnim potrebama.
Motivacija za korištenjem GFS-a je sistem u kom postoji **tolerancija na otkazivanje** komponenti i greške, ali postoji potreba za **ogromnim fajlovima** (reda veličine GB ili TB), te je najčešća operacija sa takvim fajlovima ***append*** (pisanje u postojeću datoteku) sa opuštenim modelom konzistentnosti.

52. Koje tipične obrasce pristupa podacima u fajlovima pretpostavlja GFS ?

Sistem je izgrađen od jeftinih komercijalnih komponenti koje često otkazuju. Skladišti se umeren broj ogromnih datoteka. Poslovi obrade (*workload*) se uglavnom sastoje iz dvije vrste čitanja (**velika streaming učivanja** i **mala nasumična čitanja**) i mnogo **velikih sekvencijalnih upisa**, gdje se od sistema očekuje da primeni dobro definisanu semantiku za kontrolu istovremenog dodavanja u istu datoteku od strane više klijenata. Važniji je **veliki propusni opseg**, nego malo kašnjenje.
GFS API pruža interfejs koji podržava kreiranje, brisanje, otvaranje, zatvaranje, čitanje i pisanje, snimak (kopija datoteke ili stabla direktorijuma po niskoj ceni) i dodavanje zapisa (više klijenata uporedo dodaje podatke u istu datoteku, atomičnim operacijama).

53. Koja je uloga master-a, a koja chunkserver-a kod GFS ?

Postoji jedan **master** (samo jedan radi jednostavnijeg dizajna - ima globalno znanje i donosi preciznije odluke o raspoređivanju i replikaciji). Master u tabeli održava sve metapodatke sistema datoteka: kontrolu pristupa, mapiranje iz datoteka u blokove, lokacije blokova, itd. Konzistentnost sistema se postiže tako što se kritične operacije usmjeravaju kroz master. Kada se samo pristupa podacima, minimizuje se učešće mastera.

Chunkserveri su serveri na čijim lokalnim diskovima se nalaze blokovi podataka (*chunks*) fiksne veličine (do 64 MB), kojima se pristupa preko *chunk handler-a*. Ako su blokovi veliki, povećava se vjerovatnoća da se više operacija prosleđuje istom chunkserveru, te se time smanjuje broj dodatnih zahteva za lociranje bloka u sistemu, a smanjuju se i tabele sa metapodacima na master čvoru. Blokovi se čuvaju u **tri replike** koje se nalaze na različitim serverima, na rekovima. Pošto saobraćaj ka blokovima troši ukupan propusni opseg mreže ka rekovima, novi blokovi se smeštaju na serverima koji imaju nizak nivo iskorišćenosti diskova i na više rekova. Ukoliko dođe do disbalansa broja kopija blokova, master je zadužen za rerepliciranje blokova i rebalansiranje replika kako bi se bolje iskoristili resursi.

Kada se izvršava upis ili dodavanje zapisa, to se izvršava na svakoj replici bloka. Kako bi se održao pravilan redosled operacija (mutacija), master najpre dodeljuje **najam** (*lease*) jednoj od replika (**primarna replika**), zatim ta replika određuje redosled mutacija u bloku, a ostale replike su tada prinuđene da ispoštuju taj redosled. Ukoliko master utvrdi gubitak primarne replike, pravi novi najam. Globalni redosled mutacija se utvrđuje na osnovu broja najma dobijenog od strane mastera, ili unutar najma, na osnovu serijskog broja koji dodeljuje primarna replika. Najmovi traju oko 60 sekundi. Treba izbeći keširanje na klijentskoj strani, jer se usložnjava konzistentnost između replika.

Master omogućava **konkurentnost mutacija** u istom direktorijumu, tako što prvo obezbedi zaključavanje. Kako bi se izbeglo međusobno zaključavanje, važno je da se zaključavanja dobijaju u konzistentnom poretku.

54. Šta je BigTable ?

BigTable je distribuirano skladište **za struktuirane podatke**, koje je dizajnirano da se skalira na velike skupove podataka (reda veličine PB na hiljadama mašina). Može da opsluži razne tipove obrada: batch obrade kojima treba **velika propusna moć** ili čak aplikacije koje su **osetljive na kašnjenja**. Klijent može da utiče na lokalnost podataka i na to da li se podaci dobavljaju iz memorije ili sa diska.

55. Po kom principu se BigTable deli na tablete ?

Unutar BigTable, podaci se održavaju u leksikografskom poretku po ključu. Opseg redova u tabeli može se dinamički particionisati, a svaki pojedinačni opseg je jedan tablet - jedinica distribucije podataka. Bliski redovi se opslužuju na istom serveru. Dobra lokalnost podataka se postiže dobrim izborom ključa.

56. Za koje potrebe BigTable koristi GFS, SSTable i Chubby ?

BigTable koristi GFS za **skladištenje logova i podataka**. BigTable klasteri se izvršavaju na mašinama iz deljenog bazena resursa, a za operacije raspoređivanja zadataka se oslanja na sistem za upravljanje klasterom. Unutar GFS, postoji jedan master čvor i više servera za tablete.
Uloga master čvora je da vrši raspoređivanje pojedinačnih tableta na tablet-server, te da dodaje, nadzire i vrši balansiranje opterećenja nad tablet-serverima. Takođe, master čuva listu **živih tablet-servera** i trenutnih alokacija tableta po tablet-serverima.
Tablet-serveri upravljaju skupom tabela, vrše operacije čitanja i pisanja nad tabletima, te dele prevelike tablete.
Klijent ima ugrađenu biblioteku pomoću koje direktno komunicira sa pojedinim tablet serverima kada odrađuje čitanje ili pisanje nad tabletima. Ne zavisi od mastera za utvrđivanje lokacije tableta.

BigTable koristi SSTable za **čuvanje BigTable podataka**, jer je SSTable perzistentna, sortirana mapa, sekvence 64KB blokova. Koristi se indeks bloka (skladišti se u memoriji) za njegovo lociranje. *Lookup* operacije su sa samo jedim *disk seek-om***.** BigTable klaster čuva više tabela, gde se svaka sastoji od skupa tableta i sastoji podatke o opsegu redova. Tabela se deli na više tableta veličine 100-200MB.

BigTable koristi Chubby perzistentni servis za **zaključavanje**. To omogućava da postoji najviše **jedan aktivni master** u svakom trenutku. Takođe, čuva informaciju o lokaciji sa koje treba učitati BigTable i čuva šemu podataka. Chubby koristi Paxos za konzistenciju.

Kada se master startuje, postavlja se ***master lock*** na Chubby fajl. Traže se svi živi tablet-serveri, i prikuplja se lista tableta sa svakog servera, kako bi se utvrdio njihov raspored. Zatim se utvrdi ukupan skup tableta, a nedodeljeni tableti se dodaju u listu za raspoređivanje.

57. Šta je osnovna ideja Serverless arhitekture ?

Serverless arhitektura je stil arhitekrue u kom je osnovna ideja ***Backend as a Service*** (BaaS), ili sopstveni kod koji se izvršava u izvršnim okruženjima na ***Function as a Service*** (FaaS) platformi. Uz odgovarajući *frontend*, omogućava se odstupanje od klasične instalacije na servere i potrebe da se aplikacija stalno održava na serverima. Troškovi i vreme konfiguracije se mogu znatno smanjiti, a dobija se na **jednostavnosti**, međutim, dolazi do **trajnog vezanja** za određenu platformu. U serverless arhitekturi **nema centralnog modula** koji upravlja svim operacijama.

58. Koja dva osnovna pristupa Serveless možemo uočiti ? BaaS ? FasS ?

BaaS pruža **već gotove servise**, što je isprva bila osnovna ideja serverless arhitekture. Aplikacije koje koriste ovakvu arhitekturu imaju tzv. *rich client* koji kao *backend* koristi neke od ponuđenih servisa na oblaku (BaaS).
Primjeri: Firebase za pristup cloud hosted bazama, Oauth servisi za autentikaciju, …

FaaS pruža mogućnost da se koristi **specifična programska logika**, koja se izvršava u ***stateless*** **kontejnerima**, što je glavna trenutna ideja serverless arhitekture. Kontejnere

obezbeđuje ponuđač usluge, što eliminiše potrebu da se razvojni tim bavi infrastrukturnim zadacima.

59. Ključne karakteristike FaaS pristupa ?

FaaS pristup obezbeđuje izvršavanje *backend* koda bez opterećenja održavanja sopstvenog serverskog sistema i dugotrajnih (*long lived*) serverskih aplikacija - funkcije se aktiviraju po potrebi i deaktiviraju kada su neaktivne. Okidanje funkcije se vrši na osnovu događaja koje ponuđač usluge definiše, ili na HTTP zahtev (preko *API Gateway-a*, koji rutira saobraćaj, te podatke iz zahteva prepakuje u očekivani oblik rutirane funkcije, aktivirajući događaj na koji funkcija reaguje).
FaaS funkcije nemaju stanje i ne smeju se osloniti na čuvanje nekih podataka između dve aktivacije funkcija ili na deljenje podataka između više pokrenutih instanci funkcija (ukoliko nešto od podataka treba trajno da se čuva, ta operacija mora biti eksternalizovana na neki dodatni servis).
Uvek postoji kašnjenje dok FaaS inicijalizuje funkciju. Hladni start se koristi ako već ne postoji instanca funkcije, a topli start koristi već inicijalizovanu funkciju koja je ostala u memoriji posle završetka rada. Takođe, postoji vremensko ograničenje izvršavanja funkcije, pa zadaci koji zahtevaju funkcije koje ostaju dugo u aktivnom stanju nisu pogodne za ovakvu implementaciju.

FaaS ne zahteva kodiranje u skladu sa nekim posebnim razvojnim okvirom, funkcije su standardne ali moraju biti pisane u podržanim programskim jezicima. Prilikom promene okruženja, jedina izmena u kodu bi bila ulazna *event handler* funkcija. Instalacija je specifična - vrši se *upload-om* koda funkcije.

Horizontalno skaliranje je u potpunosti automatizovano i elastično, a definiše ga ponuđač usluge.

60. Prednosti i mane FaaS ?

Prednosti:
- smanjeni operativni troškovi i troškovi razvoja
- dobro reagovanje na nekonzistentne obrasce opterećenja (povremeni saobraćaj i nagli skokovi opterećenja)
- smanjena kompleksnost isporuke
- *greener computing* - poboljšanje energetske efikasnosti računarskih sistema, kao što su serveri, računarske mreže i uređaji

Mane:
- čvrsto vezivanje za platformu određenog isporučioca
- *multitenant* problemi - više korisnika deli istu instancu softverskog sistema
- pitanje bezbednosti - povećan broj komponenti za koje je potrebno obezbediti dobru kontrolu prava pristupa
- nepostojanje čuvanja stanja na serverskoj strani