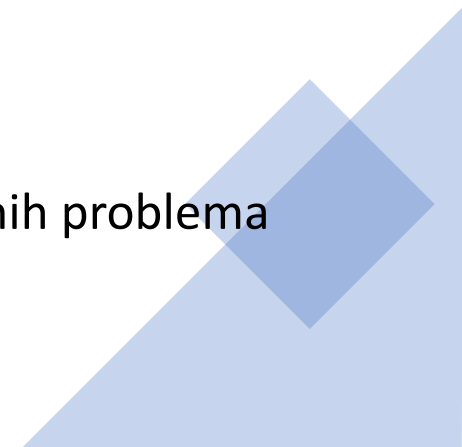


# Aplikacije i paradigme

Računarstvo u oblaku  
(Cloud Computing)



# Sadržaj

- Uvod
  - Aplikacije u oblaku
  - Izazovi sa kojima se susreće aplikacije u oblaku
  - Tipične aplikacije
    - processing pipelines
    - batch processing
    - web aplikacije
    - novi modeli aplikacija
  - Šta utiče na arhitekturu cloud aplikacije?
  - Koordinisanje aktivnosti - workflows
  - Koordinacija na bazi konačnog automata
    - ZooKeeper
  - Map Reduce programski model
  - Cloud aplikacije za rešavanje naučnih problema
- 

# Uvod

- Računarstvo u oblaku je vrlo interesantno za korisnike iz više ekonomskih razloga:
  - Veoma male početne investicije u infrastrukturu
  - U isto vreme korisnik sa zahtevnim aplikacijama može da iskoristi raspoložive resurse 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. U klasičnom modelu isporuke, uspešne aplikacije često budu žrtve sopstvenog uspeha jer vrlo brzo dosegnu limite sistema na koje su instalirane. Elastičnost u oblaku omogućava da takve aplikacije obezbede dodatne resurse kada im ustrebaju (*just-in-time-infrastructure*), i to bez dodatnog angažovanja programera.  
*Ovo naravno ne znači da treba da pravimo loš i neefikasan kod samo zato što će u oblaku biti dovoljno resursa.*

# Uvod

- Računarstvo u oblaku isplativo i za ponuđače ovakvih usluga jer im tipično omogućava znatno bolje korišćenje resursa
- Uspeh računarstva u oblaku počiva i na sposobnosti ponuđača 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.

# Aplikacije u oblaku

- Jedna od velikih prednosti računarstva u oblaku je mogućnost da iskoristi koliko god servera je potrebno da bi se optimalno odgovorilo na potrebe aplikacije (uzimajući u obzir i troškove i vremenska ograničenja koja aplikacija treba da zadovolji)
- Ovo je moguće samo ako se posao aplikacije može razdeliti 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.
- S druge strane, aplikacije čija obrada ne može podeliti ili aplikacije koje zahtevaju intenzivnu komunikaciju između konkurentnih instanci najverovatnije neće imati dobre performanse kada su u oblaku.

# Izazovi sa kojima se susreće aplikacije u oblaku

- Kao i sve aplikacije i 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. Iako infrastruktura oblaka teži da automatski balansira opterećenja, ipak je neophodno voditi računa da se podaci smeste relativno blizu procesnih jedinica na kojima će se obrađivati.
- Jedna od glavnih prednosti računarstva u oblaku – 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.

# Izazovi sa kojima se susreće aplikacije u oblaku

- 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 takođe igra bitnu ulogu u performansama bilo koje aplikacije koja obrađuje veliku količinu podataka. Organizacija skladištenja podataka, lokacija čuvanja i propusni opseg prilikom pristupa skladištu podataka su sve vrlo bitni faktori koje treba analizirati tokom razvoja aplikacije kako bi se postigle željene performanse.
- Takođe neophodno je razmotriti problem logovanja (gde, koliko, kako ograničiti količinu podataka koji se loguju). Pri tome treba voditi računa da prikupljeni podaci treba da budu dovoljni i za naknadne analize problema.

# Tipične aplikacije

- Postojeće aplikacije u oblaku se ugrubo mogu svrstati u tri kategorije
  - Processing pipelines
  - Batch processing
  - Web aplikacije



# Tipične aplikacije – processing pipelines

- Tipično obrađuju velike količine podataka, ponekad i procesno zahtevne operacije nad tim podacima
  - *Indeksiranje* – indeksiranje velike količine podataka koja se prikuplja sa net-a.
  - *Istraživanje podataka (Data mining)* – pretraživanje velikih skupova podataka sa ciljem pronaaženja podataka od interesa
  - *Procesiranje slika (Image processing)* – Flickr, Google. Obrada slika, kompresija ili enkriptovanje.
  - *Konverzija video formata*
  - *Obrada dokumenata* – konverzija formata dokumenta, obrada sadržaja dokumenata...

# Tipične aplikacije – batch processing

- Obrada veće količine prikupljenih podataka, često sa zadatim rokom
  - Generisanje dnevnih, mesečnih, godišnjih izveštaja
  - Procesiranje, agregacija i sumiranje dnevnih transakcija
  - Upravljanje inventarom velikih korporacija
  - Procesiranje računa, platnih listi...
  - Upravljanje razvojem softvera (nightly builds, repo updates, automatski testovi...)
  - ...

# Tipične aplikacije – web aplikacije

- Cloud je pogodno rešenje za veb aplikacije koje imaju periodičnu ili kratkotrajnu posećenost (konferencije, sajmovi...)
- Ili za veb aplikacije koje imaju vrlo visoke vrhove opterećenja, kada je neophodno opslužiti za nekoliko redova veličina veći broj zahteva nego u regularnim uslovima.

# Tipične aplikacije – novi modeli aplikacija

- Kako je skladištenje podataka u cloudu jeftino, a za aplikacije sa velikom količinom podataka pogodno je držati podatke blisko mestu obrade, u poslednje vreme sve češće su i aplikacije za
  - Biznis analizu
  - Paralelizovana batch obrada – map reduce
  - Aplikacije koje prikupljaju i obrađuju podatke koji pristižu sa raznorodnih senzorskih uređaja
  - ...

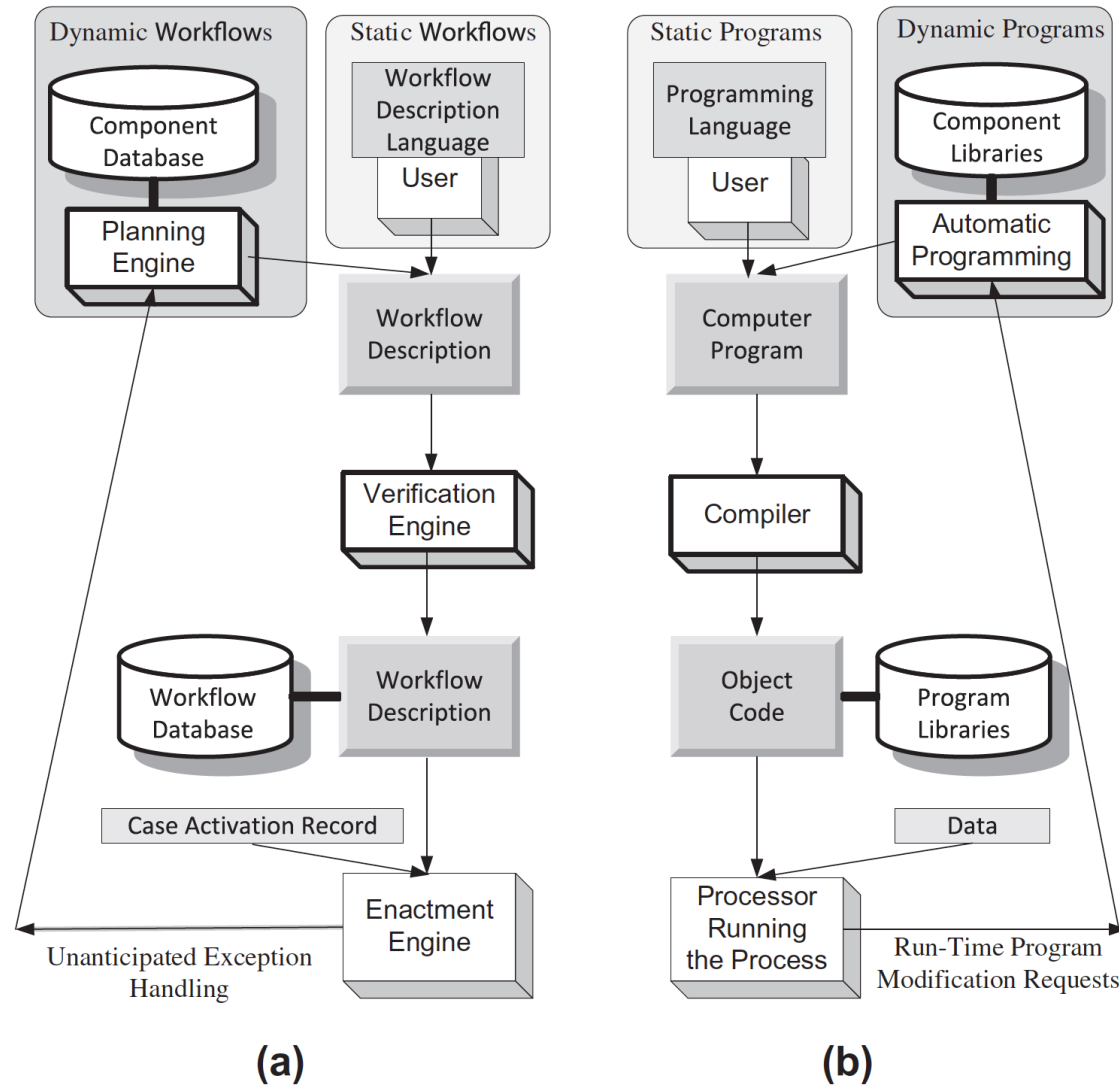
# Šta utiče na arhitekturu cloud aplikacije?

- Osnovna paradigma je client/server
  - Stateless system je poželjno stanje (jednostavniji, robusniji, lakše se skalira)
- Potreba da procesi i niti mrežne aplikacije razmenjuju podatke u nekom strukturiranom obliku koji je razumljiv svim stranama
  - *Neutrality* (neutralnost) – sposobnost aplikativnog protokola da se oslanja na različite transportne protokole i da uopšteno „trči“ nad različitim stekovima protokola.
  - *Extensibility* (proširivost) – mogućnost proširenja dodatnim funkcionalnostima – npr. bezbednosnim
  - *Independence* (nezavisnost) - mogućnost usvajanja različitih programskih rešenja.
- Komunikacija obezbeđuje razmenu podataka između procesa – RPC, ORB (CORBA), SOAP+WSDL, REST

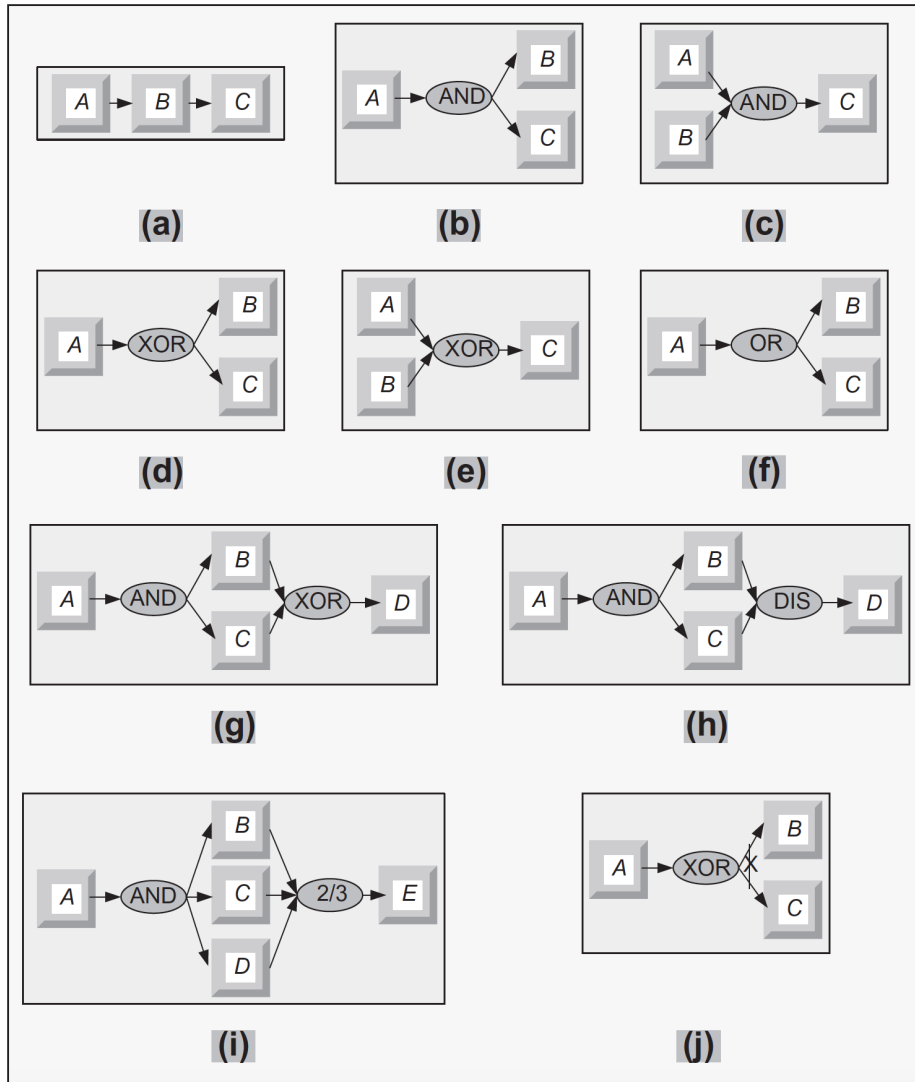
# Koordinisanje aktivnosti - workflows

- Veliki broj aplikacija za svoj uspešan završetak zahteva koordinisano izvršavanje velikog broja nezavisnih zadataka, u predviđenom redosledu.
- Radni tokovi (*workflows*) predstavljaju model za opis takvih složenih orkestracija zadataka.
- Modeli radnih tokova ili procesa su način da se opišu ključni koraci (aktivnosti) koje je neophodno obaviti kako bi aplikacija napredovala ka svom uspešnom završetku.
- Postoje različiti jezici za opisivanje modela radnih tokova, a postoje i izvršna okruženja koja tako definisane modele prevode u izvršive aplikacije
- I na cloud platformama moguće je koristiti workflow-e koji organizuju izvršavanje zadataka kroz aktivaciju više pojedinačnih servisa (Amazon Simple Workflow Service, Amazon Step Functions...)

# Koordinisanje aktivnosti - workflows



# Koordinisanje aktivnosti - workflow patterns



- (a) Sequence.
- (b) AND split.
- (c) Synchronization.
- (d) XOR split.
- (e) XOR merge.
- (f) OR split.
- (g) Multiple merge.
- (h) Discriminator
- (i) N out of M join
- (j) Deferred choice



# Koordinisanje aktivnosti - vrste workflow-a

- Statički – lako se mogu modelovati 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

# Sličnosti i razlike između klasičnog transakcionog načina rada i cloud workflowa

- Transakcioni model težište stavlja na „ugovorni“ aspekt transakcije gde ona garantuje određene ishode – dok je kod cloud workflowa izvršavanje po principu „best effort“ ageti i servisi daju „sve od sebe“ da se dostigne željeno ciljno stanje, ali ne postoji garancija
- Kritični aspekt transakcionog modela u bazama podataka je konzistentnost stanja – cloud je vrlo otvoren sistem pa je sam koncept „stanja“ teže definisati
- Transakcije u bazi traju kratko – zadaci u cloud workflowu mogu trajati dugo
- Transakcije se sastoje od dobro definisanih koraka koji se ne menjaju tokom izvršavanja, opis procesa cloud workflowa može se menjati tokom izvršavanja
- Individualni zadaci u workflowu ne moraju da ispoljavaju tradicionalne osobine baza – npr. durability – u bilo kom momentu, pre nego što je dostignuto ciljno stanje workflow može da odradi povratak na neko prethodno stanje i odatle nastavi izvršavanje. Neki od zadataka mogu biti opozivi a neki ne.
- Proces alokacije resursa je jedno od mesta za implementaciju workflow-a

# Koordinacija na bazi konačnog automata

- Elastičnost na cloud platformama zahteva mogućnost preraspoređivanja procesiranja i pristupa podacima na više sistema. Koordinacija između tih sistema onda postaje kritična funkcija za takvo distribuirano okruženje.
- Model koordinacije zavisi od sepcifičnih zadataka kao što su koordinacija skladišta podatka, orkestracija više aktivnosti, blokiranje aktivnosti dok se ne pojavi određeni događaj, postizanje konsenzusa o narednoj aktivnosti ili oporavak nakon otkaza...
- Ukoliko se koordinišu procesi koji se izvršavaju na grupi servera, vrlo često je neophodno za kritične zadatke obezbediti replikaciju serverskih instanci. Pa ukoliko primarna instanca otkáže njena zamena može odmah preuzeti izvršavanje. Ovo je moguće jedino ako je zamena u *hot standby* režimu. U ovom režimu zamenska instanca je uvek u istom stanju kao i primarni.
  - Recimo proxy bi morao biti redundantan, ako je on ulazna točka u sistem i tako single point of failure

# Koordinacija na bazi konačnog automata

- Ukoliko imamo složenu aplikaciju, u kojoj imamo servere specijalizovane za obavljanje određenih zadataka, neophodno je obezbediti koordinisani rad servisa na ovim instancama kako bi aplikacija funkcionisala
- Koordinacija svih ovih servera može se zadati putem konfiguracionih fajlova.
  - Ali ovakvo rešenje je statično i u slučaju neke promene zahteva redistribuciju te konfiguracije svim serverima
  - Osim toga u slučaju otkaza ovakvo statičko rešenje ne omogućava da se izvršavanje nastavi od tačke do koje je stiglo pre greške, već se konfiguracija učitavao iznova

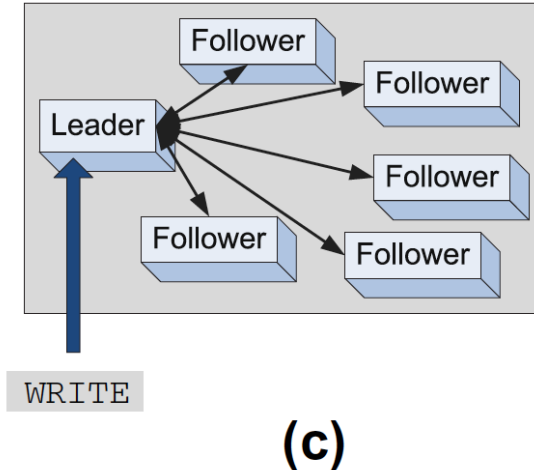
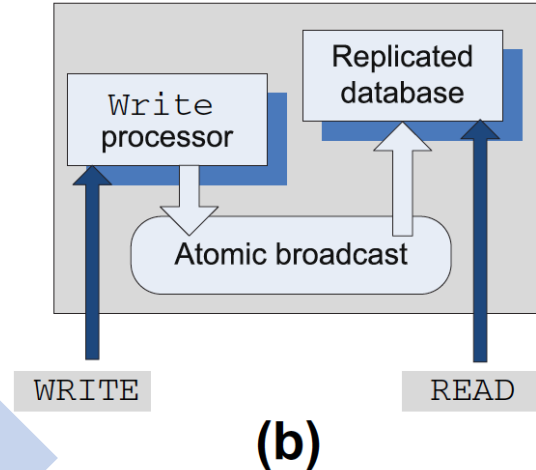
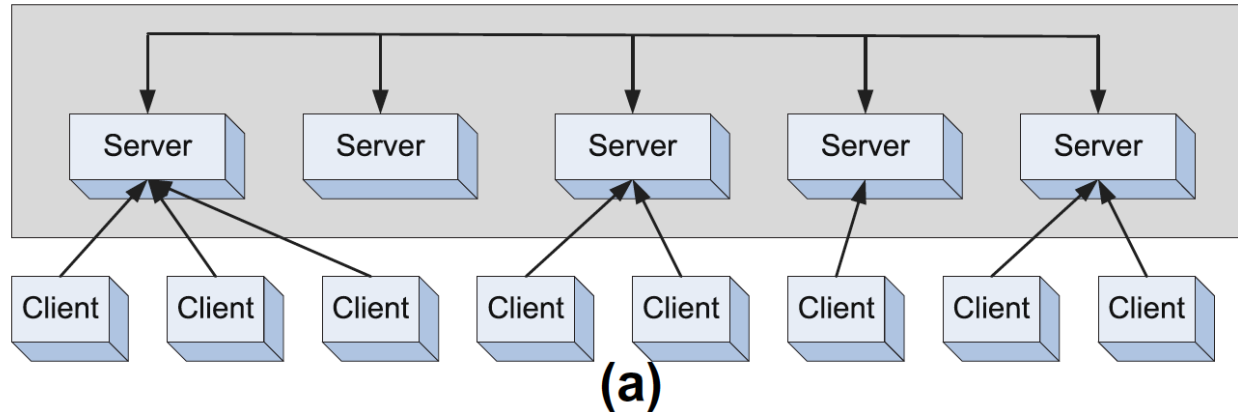
# Koordinacija na bazi konačnog automata - ZooKeeper

- Jedno od rešenja za npr. problem proxyja 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.
  - Na primeru proxyja, ako želimo da budu sinhronizovani to se postiže tako što svi izvršavaju istu sekvencu komandi, istim redosledom, što ih onda posledično dovode u isto krajnje stanje
  - Ovo se može garantovati ako svi serveri implementiraju verziju Paxos algoritma za postizanje konsenzusa
- ZooKeeper je distribuirani koordinacioni servis koji zasniva na ovom modelu.

# Koordinacija na bazi konačnog automata - ZooKeeper

- ZooKeeper se downloaduje i instalira na više servera, nakon čega klijenti mogu ostvariti konekciju i pristupiti koordinacionom servisu.
- Servis je dostupan sve dok je većina servera u grupi dostupna.
  - Serveri u grupi komuniciraju i biraju vodećeg.
  - Baza se replicira na svaku instancu i replike se održavaju konzistentnim. Servis tako obezbeđuje jedinstvenu “sliku” sistema.
  - Klijent se može povezati na bilo koji server u grupi (pack).

# Koordinacija na bazi konačnog automata - ZooKeeper



ZooKeeper koordinacioni servis.

(a) The service provides a single system image. Clients can connect to any server in the pack.

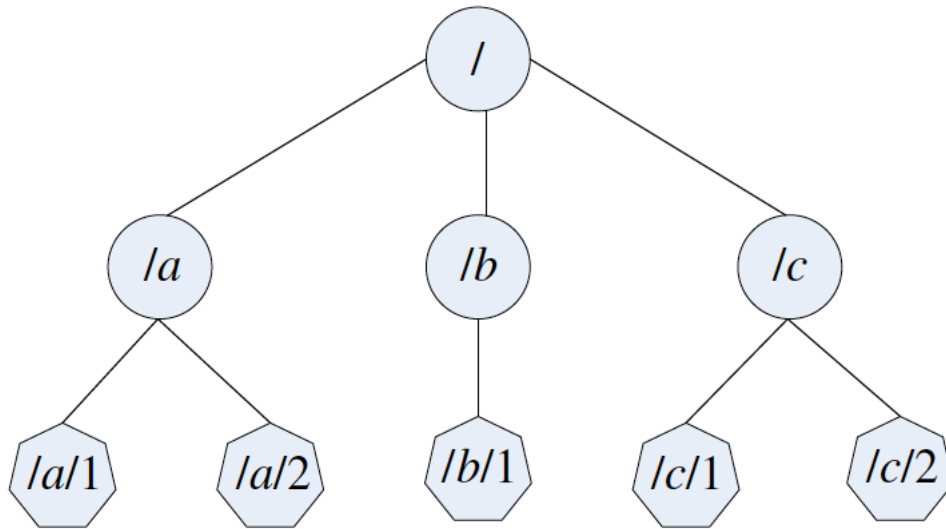
(b) Functional model of the ZooKeeper service. The replicated database is accessed directly by `read` commands; `write` commands involve more intricate processing based on atomic broadcast.

(c) Processing a `write` command:

(1) A server receiving the command from a client forwards the command to the leader;

(2) the leader uses atomic broadcast to reach consensus among all followers.

# Koordinacija na bazi konačnog automata - ZooKeeper



- 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 (verzija podataka, timestampovi promene u ACL)
- Klijent može postaviti *watch* na neki *znode*, i tako primiti notifikacije kada se stanje *znode-a* promeni.



# Koordinacija na bazi konačnog automata - ZooKeeper

ZooKeeper garantuje:

1. Atomičnost – transakcija ili uspeva ili ne
2. Sekvencijalnu konzistentnost update-a. Updatei se primenjuju striktno 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.

# Koordinacija na bazi konačnog automata - ZooKeeper

- Za brži odziv *read* operacije se izvršavaju nad lokalnom kopijom baze
- Kada leader dobije zahtev za *write* on prvo utvrdi stanje sistema na koji se write primenjuje i zatim transformiše stanje u transakciju koja enkapsulira ovu promenu
- Sloj za komunikaciju (messaging) je zadužen za sprovođenje izbora novog vodećeg ako trenutno vodeći čvor otkaže. Protokol komunikacije koristi pakete (bajt sekvence koje se šalju preko FIFO kanala), predloge (proposals) i poruke.
- Atomični sistem za razmenu poruka obezbeđuje da serveri u grupi ostaju sinhronizovani.

# Koordinacija na bazi konačnog automata - ZooKeeper

- API ZooKeeper-a:
- create – add a node at a given location on the tree.
- delete – delete a node.
- get data – read data from a node.
- set data – write data to a node.
- get children – retrieve a list of the children of the node.
- synch – wait for the data to propagate.

# Map Reduce programski model

- Glavna prednost cloud computing-a je elastičnost – mogućnost da se iskoristi onoliko servera koliko je neophodno da aplikacija obavi svoje funkcionalnosti uz zadovoljavanje vremenskih ograničenja.
- Kod transakcionih sistema tipično klijentski frontend distribuira transakcije na više backend sistema i pokušava da izbalansira njihovo opterećenje. Ako se opterećenje povećava, novi serveri se dodaju u bazen.
- Za aplikacije koje obrađuju velike količine podataka podela posla nije uvek jednostavna. Razlikuju se dve grube grupe poslova
- Modularno deljivi – podela ukupnog posla definiše se unapred
- Proizvoljno deljivi – posao se može proizvoljno podeliti u blokove koji su međusobno slične veličine

# Map Reduce programski model

- MapReduce se zasniva na jednostavnoj ideji paralelnog procesiranja aplikacija koje obrđuju velike količine podataka koje omogućavaju proizvoljnu podelu posla.
  - Podaci se podele u blokove
  - Svaki blok se dodeli jednoj instanci procesa koje se izvršavaju paralelno.
  - Kada ove instance završe svoj posao, dalja obrda se prepušta drugoj fazi u kojoj se spajaju rezultati procesiranja dobiveni sa pojedinih instanci

# Map Reduce programski model

- Primer : funkcije za prebrojavanje broja pojava neke reči u dokumentu

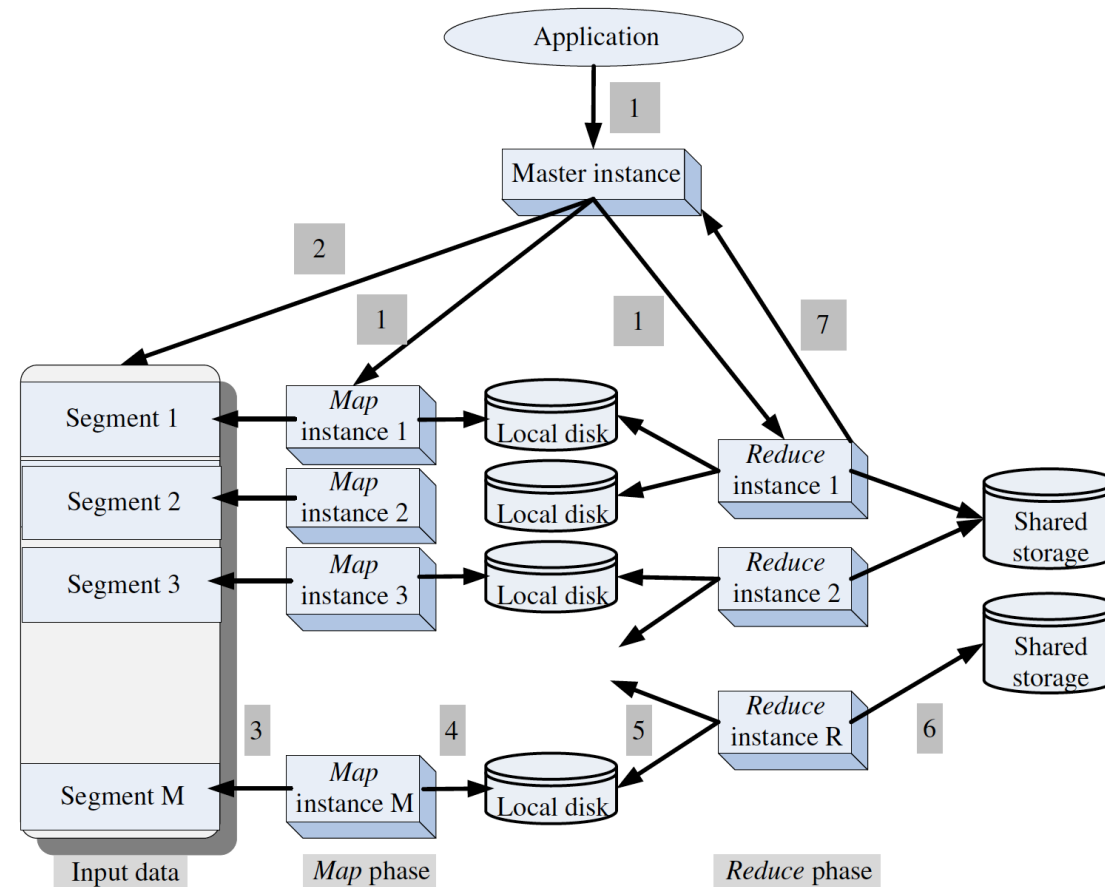
```
map(String key, String value):  
    //key: document name; value: document contents  
    for each word w in value:  
        EmitIntermediate (w, "1");
```

```
reduce (String key, Iterator values):  
    // key: a word; values: a list of counts  
    int result = 0;  
    for each v in values:  
        result += ParseInt (v);  
    Emit (AsString (result));
```

# Map Reduce programski model

1. Tokom izvršavanja podaci se podele na  $M$  delova i identifikuje  $N$  serera (sistema) na kojima će se izvršavati obrada na kojim se pokreću kopije programa. Jedan od sistem apostaje master, a ostali su *workeri*. Master raspoređuje svakom slobodnom sistemu bilo Map bilo Reduce zadatak. Master donosi  $O(M+R)$  odluk ao raspoređivanju i čuva  $O(M \times R)$  stanja woorkera (vektori stanja pojedinih sistema). Ovo utiče na izbor  $M$  i  $N$
2. Worker koji radi Map, čita odgovarjući segment podataka i produkuje međurezultat koji se smešta u bafer i zatim snima i svi rezultati se dele na  $R$  particija
3. Lokacija bafera (referenca) se prosleđuje masteru koji ih prosleđuje workerima koji izvršava Reduce
4. Kada se završe sve obrade, master obaveštava klijenta.

# Map Reduce programski model





# Cloud aplikacije za rešavanje naučnih problema

- Nauka i proučavanje raznih pojava u prirodi i društvu je stotinama godina bila empirijska
- Poslednjih decenija razvoj matematičkih i računarskih modela omogućava da se mnoge pojave analiziraju simulacijama
- cloud aplikacije omogućavaju upotrebu velike količine resursa što daje mogućnost analize velike količine podataka i pokretanje kompleksnih simulacija

# Cloud aplikacije za rešavanje naučnih problema

- Opšti problemi u skoro svim oblastima nauke:
  - Prkupljanje eksperimentalnih podataka
  - Upravljanje vrlo velikim skupovima podataka
  - Kreiranje i izvršavanje kompleksnih modela
  - Integrisanje podataka i literature
  - Dokumentovanje eksperimenata
  - Čuvanje podataka i deljenje podataka (znanja)
- Sve ovo zahteva snažne računarske sisteme, cloud pruža prilagođene aplikacije koje omogućavaju formiranje snažnih računarskih platformi prilagođenih specifičnim problemima
- Hi performance okruženja na cloudu

# Zaključak

- Aplikacije u oblaku mogu dobro iskoristiti elastičnost i raspoloživost resursa
- Neophodno je pri razvoju aplikacija obratiti pažnju na to da omogućavaju paralelizaciju kako bi postigle bolje performance
- Pitanja?