Dizajn Twitter-like sistema

Šta je Twitter?

- Twitter je socijalna mreža na kojoj korisnici mogu da objavljuju kratke poruke do 140 karaktera zvane tvitovi
- Registrovani korisnici mogu da postavljaju i čitaju tvitove, dok oni koji nisu registrovani mogu samo da ih čitaju
- Korisnici pristupaju Twitteru kroz web ili mobilnu aplikaciju

Funkcionalni zahtevi

- Korisnik može da postavlja nove tvitove
- Korisnik može da "zaprati" druge korisnike
- Korisnik može da označi tvit kao omiljeni (favourites)
- Servis treba da kreira i prikaže korisniku timeline koji se sastoji od top tvitova svih drugih korisnika koje prati
- Tvitovi mogu da sadrže slike i video zapise

Nefunkcionalni zahtevi

- Servis mora biti visoko dostupan (high availability)
- Prihvatljivo kašnjenje (latency) sistema za generisanje timeline-a tvitova je 200ms
- Konzistentnost može da trpi (u cilju visoke dostupnosti) što znači da ukoliko korisnik ne vidi novi tvit neko vreme smatra se da je to u redu

Pretpostavke i ograničenja

- Pretpostavićemo da imamo ukupno 500 miliona korisnika, sa 100 miliona dnevno aktivnih korisnika (DAK)
- Pretpostavićemo da imamo 50 miliona novih tvitova dnevno i u proseku da svaki korisnik prati 100 drugih
- Pretpostavićemo da u proseku svaki korisnik označi za omiljene 5 tvitova na dan što nam daje 100m korisnika * 5 omiljenih tvitova = 500 miliona omiljenih tvitova dnevno
- Pretpostavićemo da korisnik u proseku poseti svoj timeline dva puta dnevno i poseti 5 drugih profila. Ako na svakom profilu vidi 20 tvitova ukupan broj pregleda tvitova će biti:
- 100m DAK * ((2 + 5) * 20 tvitova) => 14 milijardi tvitova po danu

Procene za skladištenje

- Pretpostavićemo da svaki tvit ima maksimalno 140 karaktera i da su nam potrebna dva bajta da sačuvamo karakter bez kompresije (većina UTF-16 karaktera zauzima 2 bajta)
- Pretpostavićemo da je potrebno oko 30 bajta da se sačuvaju metapodaci za svaki tvit (npr. tweetID, userID, creationDate, itd)
- Ukupno za skladištenje treba:

50m tvitova * (2 bajta * 140 + 30 bajta) ≃ 15GB/dan

- Koliko nam treba prostora za skladištenje za 5 godina?
- Koliko nam još treba za korisnikove osnovne podatke, omiljene tvitove, informacije o praćenjima, itd?
- Pretpostavićemo da u proseku svaki peti tvit ima sliku a svaki deseti video. Ako u proseku slika zauzima 200KB a video 2MB to je:

(50m tvitova/5 slika * 200KB) + (50m tvitova/10 videa * 2MB) ≃ 12TB/dan

Procene za propusni opseg (bandwidth)

- Ako po danu imamo 12TB tvitova to je dolazni saobraćaj od ≃ 140MB/s
- Pretpostavka je da ima 14 milijardi tvit pregleda po danu
- Ako tvit ima sliku moramo je prikazati, a ako ima video pretpostavićemo da će korisnik svaki treći video pogledati
- Odlazni saobraćaj sistema je:

(14 milijardi * 280 bajta) / 86400s(jedan dan) teksta ≃ 45MB/s

(14 milijardi/5 * 200KB) / 86400s slika ≃ 6.5GB/s

(14 milijardi/10/3 * 2MB) / 86400s videa ≃ 11GB/s

- Ukupno: ≃ 18GB/s

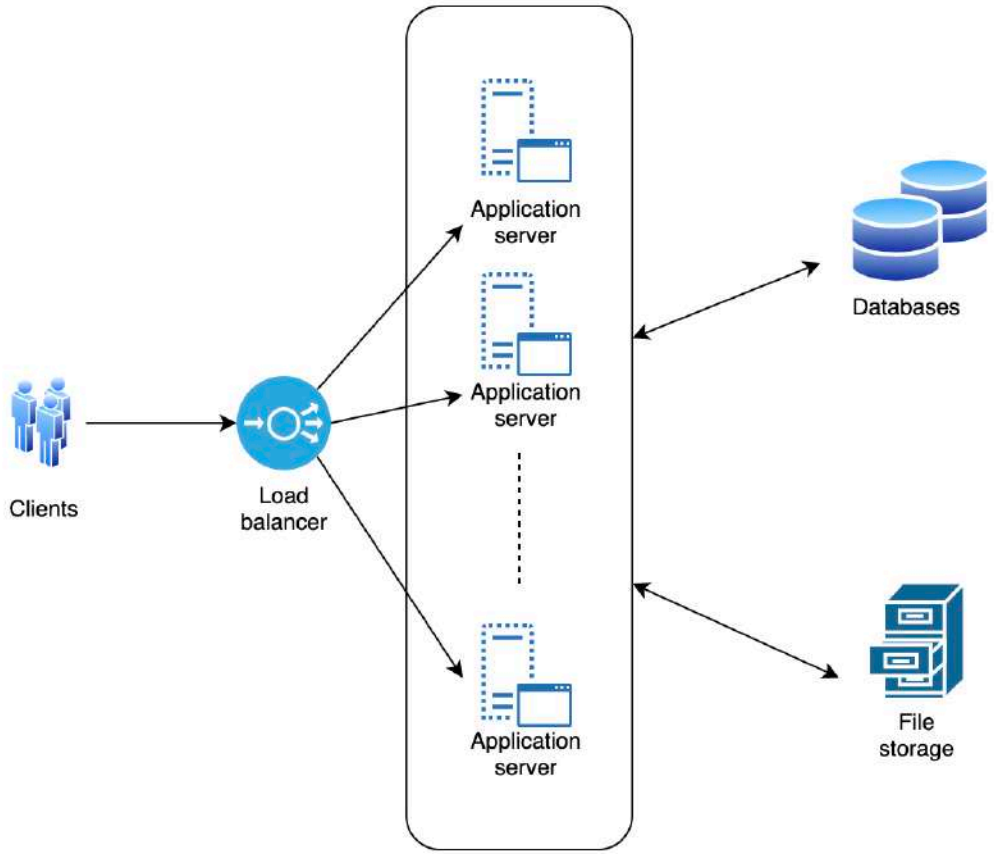
Arhitektura sistema 1/3

- Sistem je pretežno opterećen čitanjem do čega se dolazi na osnovu proračuna:

50 miliona novih tvitova / 86400s ≃ 580 tvitova po sekundi

14 milijardi pregleda tvitova / 86400s ≃ 162k tvitova po sekundi

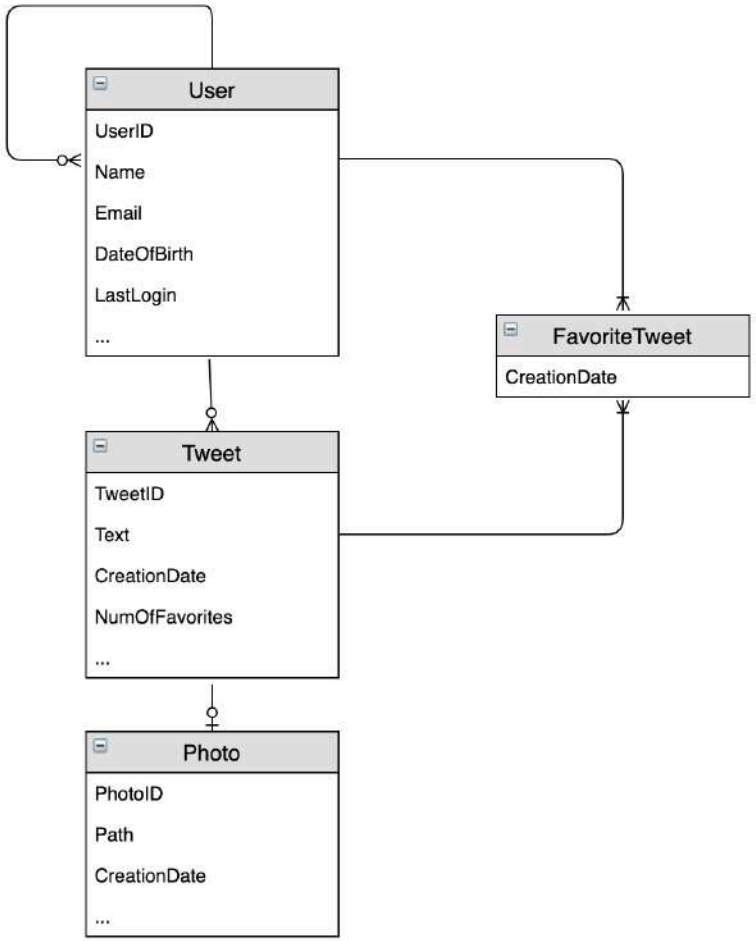
- Zbog velikog broja zahteva potrebno je više aplikativnih servera ispred kojih će se postaviti load balanseri koji će distribuirati saobraćaj
- Potrebna je baza podataka koja će ćuvati nove tvitove i koja može da podrži veliki broj čitanja
- Takođe, potrebna je baza za čuvanje slika i videa



Arhitektura sistema 2/3

Arhitektura sistema 3/3

- Iako je pretpostavka da će sistem dnevno pisati u bazu 50 miliona tvitova i čitati 14 milijardi tvitova, saobraćaj neće biti ujednačen tokom celog dana
- Kako ćemo odrediti kada će biti najveće opterećenje sistema (u podne, ponoć, samo za praznike, itd)?
- Možemo očekivati da će tada biti nekoliko hiljada zahteva za pisanjem i do milion zahteva za čitanjem



Šema baze podataka

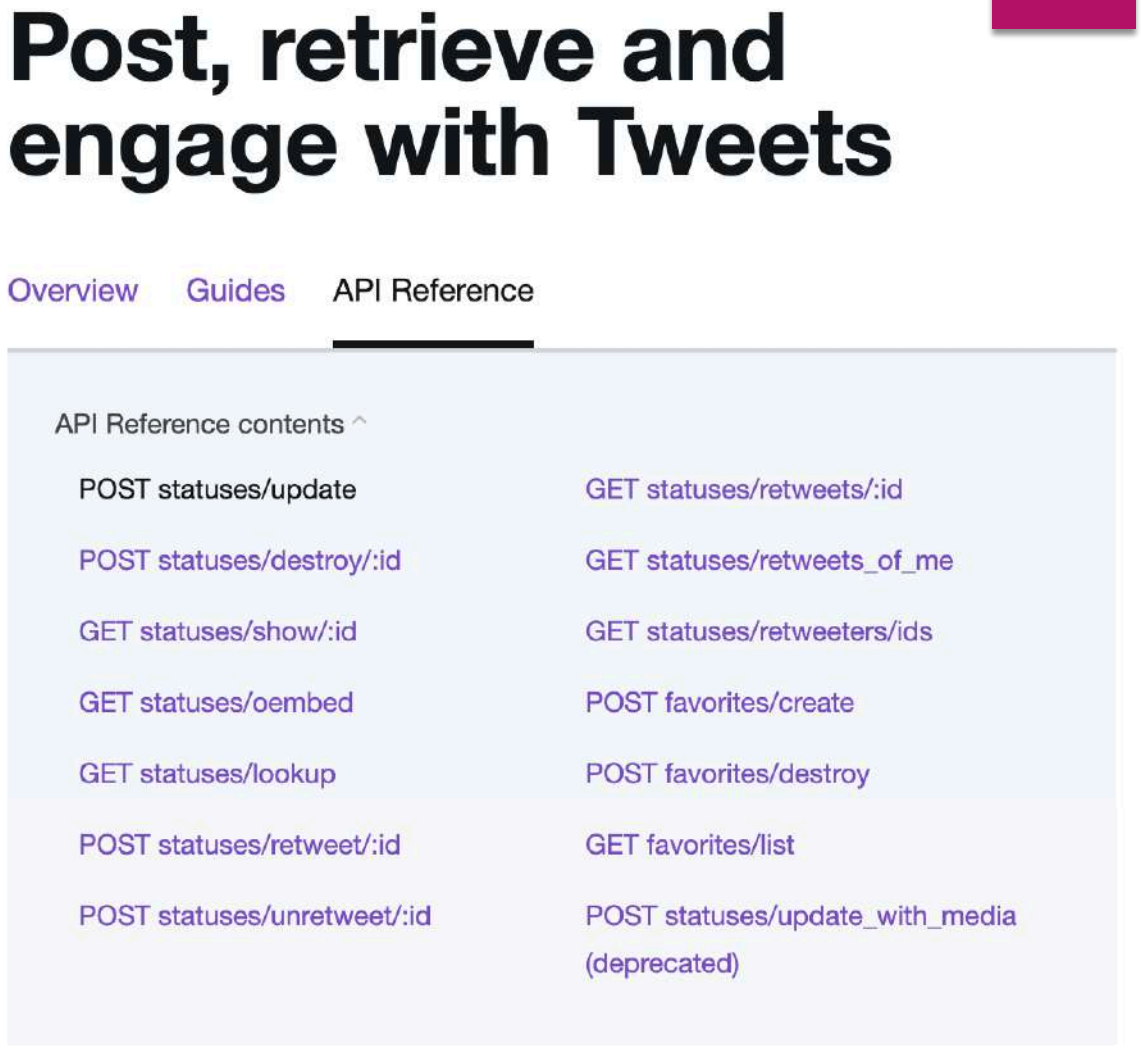
DIZAJN SE MOŽE PROŠIRIVATI

Alternative za relacione baze

- Dizajn nije ograničen na korišćenje samo relacionih baza
- NoSQL baze se mogu iskoristiti za različite stvari poput praćenja ukupnog broja zahteva po danu, broja pretraga po danu, itd. (https://readwrite.com/2011/01/02/how-twitter-uses-nosql/)
- Za čuvanje slika i videa može se koristiti npr. Amazon S3 object storage

Dizajn REST API

- Na primerima postojećeg rešenja može se osmisliti adekvatan API (https://developer.twitter.com/en/docs/basics/getting-started)
- Možemo iskoristiti REST arhitektonski stil za dizajniranje funkcionalnosti servisa



Particionisanje podataka (data sharding)

- Pošto se radi sa velikim brojem tvitova na dnevnom nivou, potrebno je distribuirati podatke na više mašina kako bi se čitanje/pisanje odvijalo efikasnije

Particionisanje na osnovu UserID

- Možemo čuvati sve podatke istog korisnika na jednom serveru
- UserID se može proslediti hash funkciji koja će odrediti koji server baze podataka će čuvati korisnikove tvitove, omiljene tvitove, itd.
- Kada hoćemo da očitamo podatke za korisnika, hash funkcija će reći odakle je potrebno čitati podatke
- Problemi:
- Šta ako je korisnik "hot" tj. za njega ima dosta upita? To će uticati na performanse servera
- Šta ako neki korisnici tokom vremena generišu mnogo više tvitova ili imaju više pratilaca od drugih? Postizanje uniformne raspodele podataka po serverima postaje poteškoća

Particionisanje na osnovu TweetID

- Hash funkcija će mapirati TweetID na proizvoljni server gde će se čuvati tvit
- Da bi se našli tvitovi, moraju se pretražiti svi serveri i svaki server će vratiti kolekciju tvitova
- Centralizovani server će agregirati rezultate i vratiti ih korisniku
- Problemi:
- Pristup rešava problem “hot” korisnika, ali je potrebno pretražiti sve particije da bi se našli svi traženi tvitovi korisnika tako da ovaj pristup može negativno uticati na odziv
- Keširanje ”hot” tvitova?

Particionisanje na osnovu vremena kreiranja tvita

- Čuvanje tvitova po vremenu kreiranja ima prednost bržeg dobavljanja najsvežijih tvitova
- Ako se specificira na kojem se (u tom trenutku) serveru čuvaju poslednji tvitovi, pretraga se ograničava na poslednjih nekoliko dodatih servera
- Problemi:
- Saobraćaj nije adekvatno distribuiran
- Prilikom pisanja, novi tvitovi će ići na jedan server dok će ostali serveri biti neupotrebljeni
- Prilikom čitanja, server koji čuva najsvežije podatke će imati mnogo veće opterećenje od ostalih

Kombinacija TweetID i vreme kreiranja

- Kombinacijom dobijamo benefite koje oba pristupa daju
- Brzo možemo naći najsvežije tvitove
- TweetID mora biti jedinstven na nivou sistema i sadržati timestamp
- Za potrebe kreiranja TweetID možemo koristiti Unix vreme (https://en.wikipedia.org/wiki/Unix_time)
- Prvi deo TweetID će biti vreme u sekundama a drugi autoinkrement sekvenca npr:
  - 1570458603 000001
  - 1570458603 000002
  - 1570458603 000003
- Koliko bita je potrebno da se čuvaju tvitovi u narednih 50-100 godina?

Keširanje

- Možemo iskoristiti mehanizme za keširanje “hot” tvitova i korisnika
- U te svrhe možemo koristiti gotova rešenja poput Memcached ili Redis da čuvamo cele objekte
- Pre upita ka bazi proveravaće se da li se traženi tvitovi nalaze u kešu

Kako možemo unaprediti strategiju keširanja?

- Ako se vodimo pravilom 80-20 koje kaže da 20% tvitova generiše 80% ukupnog broja zahteva za čitanjem tvitova možemo na dnevnom nivou keširati samo tih 20% najpopularnijih tvitova
- Šta ako bismo keširali samo najnovije podatke?
- Ako npr. 80% korisnika vidi tvitove od poslednja 3 dana možemo keširati sve tvitove iz tog perioda

Strategija zamene podataka u kešu

- Kada je keš pun, želimo da zamenimo stare tvitove novijim
- Možemo koristiti Least Recently Used (LRU) strategiju
- Ovom strategijom prvo odbacujemo tvitove koji najduže nisu korišćeni
- Već smo izračunali da za skladištenje tvitova bez slika i videa treba:

50m tvitova * (2 bajta * 140 + 30 bajta) ≃ 15GB/dan

- Znači da za poslednja tri dana treba oko 50GB memorije
- Podatke možemo keširati na samo jednom serveru ali takođe možemo podatke replicirati na više servera kako bismo distribuirali saobraćaj bolje i izbegli opterećenje samo jednog keš servera
- Slična situacija bi bila i sa slikama i videima

Dizajn Tweets Feed-a

- Možemo ga posmatrati kao potpuno odvojen sistem koji zasebno treba dizajnirati
- Problem možemo podeliti na dva dela:
  1. Generisanje prikaza tvitova korisnika koje ulogovani korisnik prati
  2. Objavljivanje tvitova i dostavljanje pratiocima (push)

Generisanje feed-a

- Feed se generiše od tvitova korisnika koje ulogovani korisnik prati
- Postupak generisanja bi se mogao definisati na sledeći način;
  1. Korisnik šalje zahtev za generisanje svežeg feed-a
  2. Naći sve korisnike koje ulogovani korisnik prati
  3. Dobaviti *N* najnovijih/najpopularnijih tvitova tih korisnika
     a. Rangirati tvitove na neki način
     b. N može biti maksimalan broj tvitova koji ulogovanom korisniku mogu da se prikažu na ekranu (ako je podešeno npr. da se prikazuje max 20 u trenutku)
     c. N može biti npr. 5 puta max broj tvitova koji se mogu u jednom trenutku prikazati ulogovanom korisniku
     d. Tih *N* tvitova keširati i prikazati top *X* (koliko se može prikazati na feed-u)
  4. Kada ulogovani korisnik dođe do kraja feed-a, da bi mu se prikazalo novih *X*, čitati ih iz keša ili se ponovo obratiti serveru

Objava i push tvita

- Proces objave i *pushovanja* tvitova pratiocima zove se *fanout*
- Moguće varijante fanout-a:
  1. Fanout on load ili Pull – pratioci sami šalju zahtev za najsvežijim tvitovima (ili se automatski pull radi periodično)
     a. Novi tvitovi se neće prikazati dok se ne izda zahtev
     b. Teško je odrediti period na koji raditi pull jer zahtev ne mora vratiti nove tvitove ako ih nema
  2. Fanout on write ili Push - kada se objavi tvit odmah se pushuje svim pratiocima uz notifikaciju
     a. Ako neko ima mnogo pratilaca server treba da pošalje tvit velikom broju korisnika
  3. Kombinacija prva dva - za korisnike sa mnogo pratilaca radimo pull, za ostale push

Replikacija i otpornost na greške

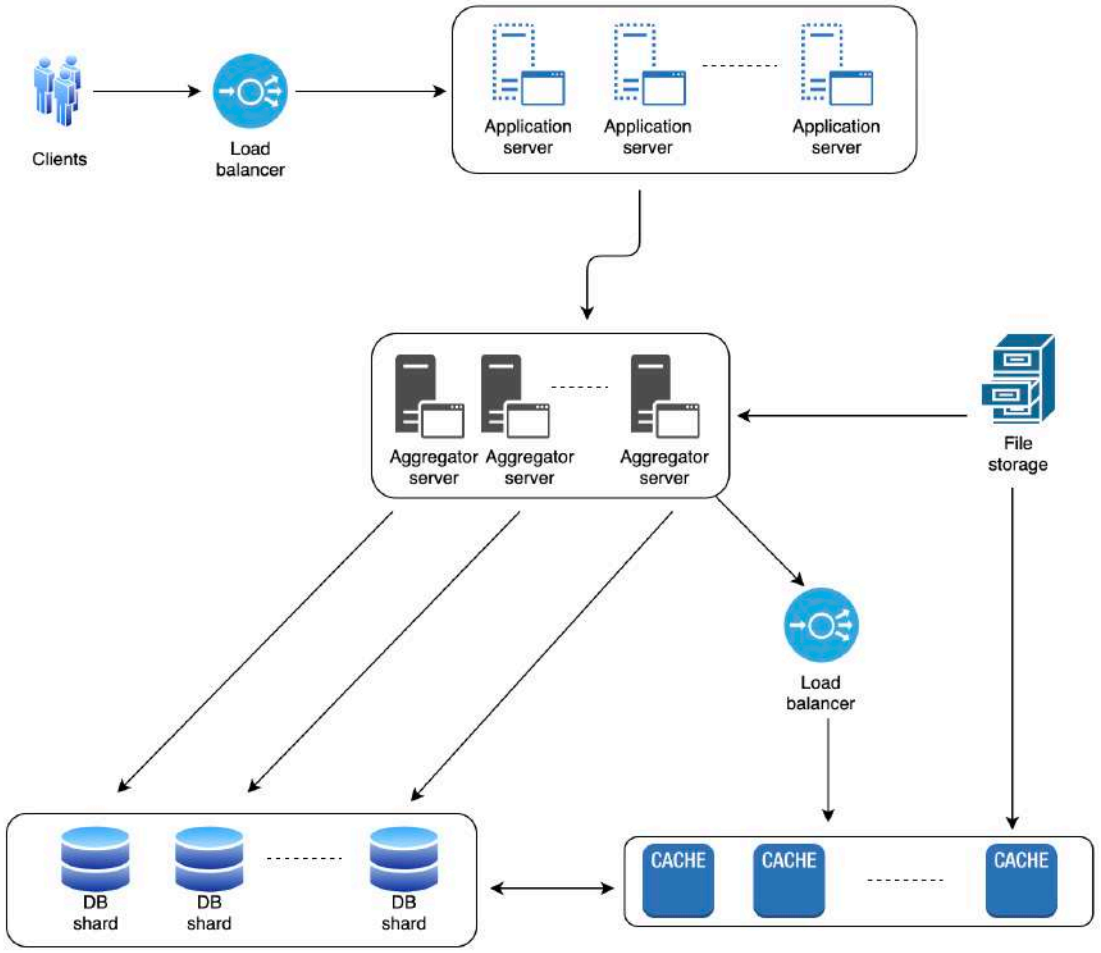
- Pošto je sistem pretežno okrenut ka čitanju podataka, možemo imati više sekundarnih servera koji će služiti samo za čitanje podataka
- Svo pisanje će ići na primarni server baze podataka i repliciraće se na sekundarne
- Kada primarni server nije dostupan, možemo izabrati neki od sekundarnih servera za novi primarni (odraditi failover)

Load balancing

- Load balansere možemo dodati na bar dva mesta u sistemu:
  - Između klijenata i aplikativnih servera
  - Između centralnih servera koji vrše agregaciju prikupljenih podataka i keš servera
- Različite strategije za raspodelu zahteva se mogu koristiti
- Round Robin je jedan od jednostavnijih koji zahtev usmerava na prvi sledeći server u rotaciji
- Ako neki od servera nije dostupan izbaciće ga iz razmatranja pri usmeravanju zahteva
- Problem:
- Ako je server preopterećen ili spor, load balanser neće prestati da šalje zahteve na njega pa je neka naprednija strategija raspodele zahteva bolja opcija

Monitoring

- Mogućnost da se sistem aktivno posmatra otvara nove opcije za unapređenje
- Ako sakupljamo odgovarajuće podatke, možemo dobiti uvid u to kako sistem radi
- Možemo meriti:
  - broj novih tvitova po danu
  - broj novih tvitova po sekundi
  - kada je najveći saobracaj
  - …
- Na osnovu rezultata možemo videte da li nam je potrebna druga strategija za LB, keširanje, replikaciju, itd.



Finalna arhitektura sistema