Skalabilnost

Pretpostavke:

- ukupan broj korisnika aplikacije je 200 miliona,
- broj zakazanih novih pregleda i operacija na mesečnom nivou je milion,
- sistem moram biti skalabilan i visoko dostupan.

1. Strategija za particionisanje podataka

Predlažemo sledeću strategiju za particionisanje podataka:

- Horizontalno particionisanje tabela pregled i operacija prema koloni id_klinike, a zatim prema koloni id_termina, odnosno prema datumu, jer pretpostavljamo da će postojati veliki broj pregleda zbog velikog broja registrovanih korisnika koji zakazuju preglede.
- Horizontalno particionisanje tabela pacijenti, lekari i medicinske_sestre prema koloni id_adrese, odnosno prema državi, jer prema pretpostavci postoji 200 miliona korisnika, te bi bilo pogodno da se oni podele po državama.
- Horizontalno particionisanje tabele zdravstveni_kartoni prema koloni krvna_grupa, jer ako pretpostavimo da su 90% korisnika pacijenti, postojaće 180 miliona zdravstvenih kartona.
- Horizontalno particionisanje tabele lekovi prema koloni sifra_leka, jer pretpostavljamo da u sistemu može da bude registrovan jako veliki broj lekova.
- Horizontalno particionisanje tabele ocene prema koloni id_pacijenta, jer pretpostavljamo da će bar jedna trećina pacijenata da ocijeni lekara ili kliniku koju poseti, čime dobijamo veliki broj ocena.

2. Strategija za replikaciju baze i obezbeđivanje otpornosti na greške

Predlažemo korištenje jednog primarnog servera baze, koja bi imala svoj backup koji bi služio za oporavak sistema u slučaju otkaza. Takođe, možemo uvesti i server koji bi čuvao podatke o pacijentima jer oni nisu vezani za određenu kliniku, kao i servere sa po jednim backup serverom za svaku kliniku, po potrebi. Pošto naša aplikacija koristi sesiju za proveru prava pristupa resursima, moramo voditi računa i o smeštanju podataka u sesiji, te predlažemo da se svaka sesija čuva na po dva servera, jedan primarni i jedan backup server.

3. Nadgledanje operacija korisnika

Pošto je najčešća operacija korisnika zakazivanje pregleda (na više načina, od strane različitih tipova korisnika), predlažemo praćenje broja novih zahteva za zakazivanje pregleda od strane pacijenta, kreiranja i zakazivanja predefinisanih pregleda, i zahteva za zakazivanje pregleda i operacija od strane lekara. Možemo pratiti broj zahteva po danu, broj zahteva na

određenoj teritoriji, broj zahteva u određenom periodu dana itd. što bi nam pomoglo da utvrdimo da li je potrebno menjati strategiju keširanja, particionisanja ili replikacije baze.

4. Strategija keširanja podataka

Svrha keširanja podataka jeste brža razmena podataka između baze podataka i servera, stoga predlažemo da informacije koje se često koriste postavimo u keš memoriju. Pošto pretpostavljamo da će biti million zahteva za kreiranje pregleda, mislimo da tip posete i sale postavimo u keš jer će svakako najviše oni biti korišćeni i time bi smanjili potencijalni saobraćaj ka bazi podataka. Takođe, na osnovu istog problema, svaki pregled treba unositi dijagnoze i lekove tako da i njih bismo mogli staviti u keš. Pretpostavimo da pacijent koji dolazio u skorašnje vreme (npr. u roku od mesec dana), verovatno imati ponovni pregled za kontrolu ili dodatni pregled koji mu lekar odredi. Na osnovu ovoga, možemo keširati skorašnje preglede i pacijente. Kada pacijent zakazuje pregled potrebno je da izabere kliniku u kojoj će odraditi pregled. Možemo informacije o klinici keširati (npr. opis, adresu i prosčnu ocenu).

5. Okvirna procena hardverskih resursa u narednih 5 godina

Pretpostavimo da je za svakog pacijenta u bazi potrebno 240B. Pošto imamo 200 miliona korisnika (smatramo da su 98% korisnika pacijenti) za njih bi bilo potrebno oko 48GB memorije za skladištenje. Pretpostavimo da za svaki pregled je potrebno 230B. Za 1 milion pregleda mesečno dolazimo do cifre od 60 miliona pregleda u 5 godina. Za skladištenje ove memorije nam je potrebno oko 14GB memorije. Ostali podaci ne zauzimaju više od 5GB memorije. Time dobijamo cifru od 67GB za jednu bazu podataka. Svako repliciranje baze dovodi do dupliranja za ovom količinom memorije.

Međutim, kada govorimo o razmeni podataka između aplikacije i klijenta ovi podaci bivaju pretvoreni u teksutalni sadržaj koji se prenosi krož mrežu. Stoga dajemo drugu procenu za veličinu pregleda i zahteva korisnika. Pretpostavimo da svaki zahtev ima oko 400 karaktera što iznosi po UTF-8 standaradu 800B. Recimo da korisnik svakog dana pošalje oko 100 zahteva dolazimo do toga da svaki korisnik dnevno koristi 80KB podataka. Ukoliko dnevno imamo prosečno 40 miliona korisnika (pretpostavimo da u zimskim periodima imamo više aktivnih korisnika) dolazimo do toga da dnevno bude preneseno 3200GB memorije tj. otprlike 38 MB/s. Treba pretpostaviti da u nekom delu dana poraste broj zahteva koji pristignu (npr. smatramo da je broj korisnika noću manji nego danju), smatramo da je dovoljno imati protok od 100MB/s, da bi zadovoljili zahtev za visoku dostupnost sistema.

6. Predlog strategije za load balnsera

Smatramo da je potrebno imati load balanser na mestu između mreže i aplikacionog sloja kao i između server i keš skladišta koji će služiti da preraspodele pristigle zahteve podjednako i time smanje opterećenje između navedenih slojeva arhitekture. Predlažemo tehniku “Weighted Round Robin” na osnovu koje radi load balanser. Load balancer će za svaki pristigli zahtev poslati serveru koji je sledeći i koji trenutno ima najmanje zahteva (za svaki

server load balancer je potrebno da pamti broj zahteva koji trenutno obrađuje server). Server koji ima manji broj zahteva biće odabran u cilju smanjenja saobraćaja na servere koji obrađuju trenutno više zahteva ili zahteve koji zahtevaju više procesorskog vremena.

7. Dizajn arhitekture sistema