SKALABILNOST

Postoje dva načina kako se može postići skalabilnost. Prvi način je najintuitivniji i najlakši, ali nije dobro rešenje. On podrazumeva skaliranje u smislu kupovine većeg servera (sa većim brojem CPU-ova, kao i memorije). To može raditi određeno vreme, ali ako opterećenje nastavi eksponencijalno da raste (što je i najčešće slučaj) došli bi do tačke kada bi zahtevi za resursima premašili hardverska ograničenja današnjice ili bi održavanje tih servera dovelo do velikih finansijskih gubitaka.

Pošto ovo nije adekvatno rešenje, drugi način predstavlja skaliranje podelom opterećenja sistema. Jedan uobičajen scenario za skaliranje je da imamo grupu servera na čelu sa *load balancer*-om. Kako naš sistem raste (broj korisnika se povećava), možemo jednostavno dodati novi server u grupu, ovaj način je značajno jeftniji na duže staze i postoje više strategija na osnovu kojih se može postići. Međutim, ovo rešenje zahteva određene kompromise što se tiče arhitekture sistema.

1. Predlog strategije za particionisanje podataka

Osnovna ideja iza particionisanja podataka jeste da podelimo jednu veliku bazu podataka u nekoliko manjih. Ova strategija je takođe poznata kao *shared nothing approach.* Postoje dva pristupa particionsanju: vertikalno i horizontalno.

Pod pretpostavkom da je ukupan broj korisnika aplikacije 200 miliona i da je broj zakazanih novih pregleda i operacija na mesečnom nivou oko 1 milion, moramo organizovati podatke tako da sistem bude skalabilan i visoko dostupan. To možemo postići tako što nad određenim tabelama odradimo vertikalno i/ili horizontalno particionisanje u zavisnosti od operacija nad tim tabelama.

Sa tabelama gde nad nekim kolonama češće radimo *read/write* operacije nego nad ostalim možemo koristiti strategiju vertikalnog particionisanja na dve tabele. U vertikalnom particionisanju bi te dve tabele spajali primarnim ključem, gde će šema prve tabele sadržati kolone kojima više pristupamo, a druga tabela ostatak prvobitne šeme. Time optimizujemo rad nad ovakvim tabelama. Međutim, vertikalno particionisanje po definiciji je ograničeno brojem funkcionalnih particija koje možemo identifikovati i tada je neophodno primeniti i horizontalno particionisanje. U slučajevima kada imamo veliku količinu podataka u tabeli onda radimo horizontalno particionisanje, što podrazumeva kreiranje više tabela sa identičnom šemom i podelu podataka po određenoj koloni.

Neki od predloga na primeru naše aplikacije:

Vertikalno particionisanje:

Tabelu ZKarton bismo particionisale na ovaj način, tako što bi kolone id, datum_rođenja, krvna_grupa i pol predstavljale jednu šemu, a id, težina, visina i dioptrija drugu šemu jer bi nad njima imale učestalije izmene.

Horizontalno particionisanje:

Tabelu Pregled bismo particionisale na ovaj način, po koloni klinika_id, u zavisnosti od rasta količine podataka bi naknadno horizontalno particionisale po opsegu vrednosti kolone datum_vreme.

2. Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Postoje dva razloga za replikaciju podataka: povećanje pouzdanosti sistema (ukoliko se jedna od replika ošteti mogu se koristiti podaci iz drugih replika) i poboljšanje performansi (podaci se smeštaju bliže procesu koji ih koristi). Međutim, replikacija stvara problem konzistentnosti podataka:

- Promene u replikama se moraju propagirati, a propagiranjem smanjujemo performanse sistema.
- Da bismo ostvarili konzistentnost neophodno je da promene propagiramo tako da privremene nekonzistentnosti nisu uočljive.
- Korišćenjem više procesa koji rade sa kopijama podataka dobijamo skalabilan sistem i bolje performanse.

Vrste replikacija: Transakciona, *Merge* i *Snapshot* replikacija.

Transakciona replikacija se obično koristi u server-to-server scenariju. Nastale promene u šemi i podacima odmah se šalju pretplatnicima istim redosledom kojim nastaju, time postižemo ispravnost transakcija u svakoj publikaciji. Kod Merge replikacije promene u šemi i podacima se prate okidačima, dodaju se sve promene nakon poslednje sinhronizacije. Snapshot replikacija obezbeđuje početne podatke za prethodne dve replikacije.

Na našem primeru predlažemo korišćenje Transakcione replikacije. Imali bismo glavni server (*publisher*), na kom bi se nalazili svi podaci kliničkog centra, koji bi u realnom vremenu distribuirao sve promene bazama podataka (*subscriber*). Kako ne bi došlo do konflikata zabranili bismo promene na *subscriber*-ima i tako bi *subscriber*-i predstavljali read-only baze podataka, tj. *backup*-ove. Prema tome transakciona replikacija nudi *backup* za česte promene baze podataka, na dnevnom nivou. Uvođenjem replikacije baze obezbeđujemo otpornost na greške, tj. ukoliko dođe do delimičnog otkaza sistem će moći da nastavi sa radom zahvaljujući repliciranim podacima.

3. Predlog strategije za keširanje podataka

Keširanje je tehnika koja se koristi da bi se ubrzala pretraga i dobavljanje podataka. Umesto čitanja podataka iz baze podataka ili drugog udaljenog sistema, podaci se čitaju direktno iz keša na računaru koji zahteva te podatke. U zavisnosti od podataka i učestalosti pristupa podacima biramo način keširanja.

Predlog za keširanje naših podataka je *Cache-Aside* strategija, jedna od široko primenjenih strategija. Aplikacija prvo proveri keš, ako pronađe podatke pročita ih i vrati ih klijentu, u suprotnom pristupa bazi podataka i podatke koje dobavi iz baze skladišti u keš. Ovom strategijom podržavamo otpornost na otkaze keš sistema. Ukoliko keš sistem padne, sistem i dalje može da radi, tako što direkno komunicira sa bazom.

Neki od primera upotrebe keša:

- Za pacijenta: u kešu čuvati zakazane preglede i operacije i istoriju pregleda i operacije, zato što će njima pristupati svaki put kada se uloguje u aplikaciju. Čuvati listu klinika jer će njima pristupati kad želi da zakaže pregled.
- Za lekara: u kešu čuvati radno vreme, zakazane preglede i operacije.
- Admin klinike: u kešu čuvati sale jer njima pristupa svaki put kad treba da odobri neki pregled/operaciju i pronađe salu za njega.

4. Okvirna procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina

Procenu smo odradile pod pretpostavkom da je trenutni broj korisnika aplikacije 200 miliona i da će broj rasti godišnje za 50 miliona. S obzirom da naša aplikacija ne manipuliše multimedijalnim sadržajem dnevno bi svaki korisnih uneo manje od 50kB. Na osnovu toga možemo okvirno proceniti hardverske resurse potrebne u narednih 5 godina po formuli:

broj_godina * broj_korisnika * 365 * 50kB

broj_godina = 5

broj_korisnika = prosečan broj korisnika u periodu od budućih 5 godina

5. Predlog strategije za postavljanje *load balancera*

Postoji dosta strategija za održavanje ravnoteže zahteva klijenata među serverima. U zavisnosti od odabrane strategije *load balancer*-a zahtevi će biti prosleđeni na odgovarajući server. Kada je opterećenje malo koristiće se neka od jednostavnijih strategija, a pri visokom opterećenju potrebne su kompleksnije metode koje osiguravaju jednaku distribuciju zahteva među serverima.

Po pretpostavci da imamo 200 miliona korisnika, imaćemo veliki broj zahteva i samim tim nam je potrebna kompleksnija metoda za opsluživanje.

Naš predlog je korišćenje *Weighted Round Robin* strategije. Ona predstavlja unapređenu verziju *Round Robin* algoritma, koji radi po principu kružnog raspoređivanja, s tim što

Weighted Round Robin dodeljuje svakom serveru numeričku vrednost u zavisnosti od toga koliko zahteva u proseku može da obradi u jedinici vremena. Na osnovu tih vrednosti radi se raspoređivanje tako da serveri sa većom težinom dobijaju više zahteva na izvršavanje. Ovu strategiju bismo koristile za raspoređivanje klijentskih zahteva ka aplikativnim serverima.

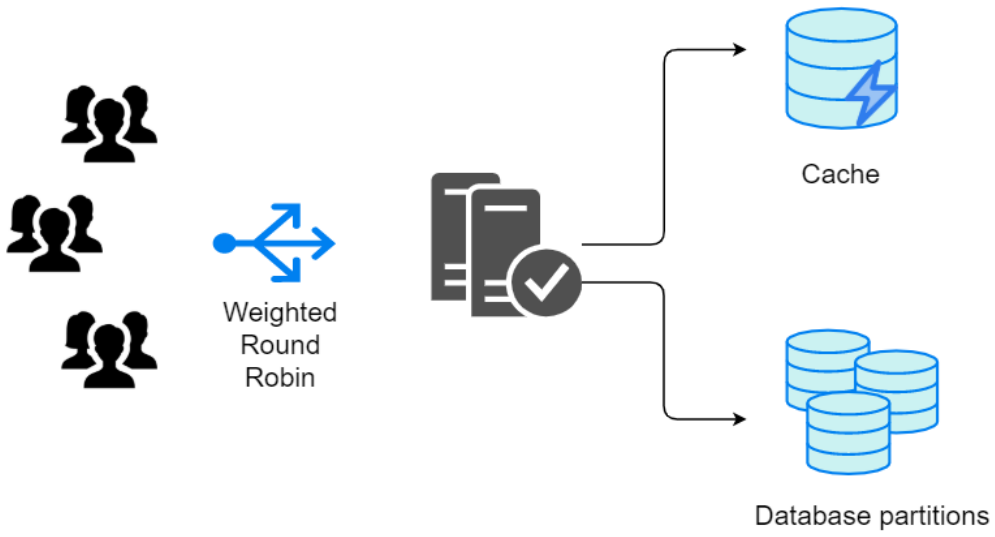
6. Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

U cilju poboljšanja performansi sistema predlažemo nadgledanje učestalosti sledećih operacija:

1. Slanje zahteva za pregledom/operacijom
2. Slanje zahteva za registracijom
3. Prijava na sistem

Takođe bismo pratile i analizirale vremenske intervale kada aplikacija ima veliko opterećenje, i time obezbedile da u tim vremenskim intervalima imamo obezbeđen kontinualan i brz protok podataka. Na osnovu prikupljenih rezultata opservacije bismo mogle eventualno promeniti prethodno odabrane strategije.

7. Kompletan crtež dizajna predložene arhitekture (aplikativni serveri, serveri baza, serveri za keširanje, itd.)



Slika 1 Dizajn predložene arhitekture