Skalabilnost – Tim 15

1. Dizajn šeme baze podataka(koceptualni,logički ili fizički)

2. Predložiti strategije za particionisanje podataka

Particionisanje podataka bi moglo da se obavi na dva načina:

-Kada su u pitanju sami entiteti (avanture, brodovi, vikendice...), tabele koje ih sadrže mogle bi da se podele na dve tabele, tabela koja sadrži podatke koji se koriste pri skoro svakom čitanju, a u drugu tabelu dodatne informacije koje su potrebe samo kada je u pitanju pregled profila samog entiteta.

-Kada su u pitanju vremenski zavisni događaji poput rezervacija, brzih rezervacija, otkazivanja itd. Pogodan način za particionisanje je horizontalni. Mogli bismo ih particionisati u tabele koje bi predstavljale određen vremenski period (mesec, kvartal...) što bi pretragu slobodnih termina, izveštaja itd. Učinio efikasniim.

3. Predlog strategije za replikaciju baze i odredjivanje otpornosti na greske

U razvoju naše aplikacije korišćen je PostgreSQL, koji nudi različite načine za arhiviranje i replikaciju primarne baze podataka za bekap, visoku dostupnost i *load balancing* scenarije. U *Hot Standby* modu, sistem zapošljava dva ili više servera, od kojih primarni server podržava aktivnu bazu podataka. Proces kopiranja podataka sa aktivnog na drugi server se naziva *PostgreSQL Replication*. Aktivni server je u tom slučaju *Master* server, a server koji prima kopirane podatke je *Replica* server.

4. Predlog strategije za kesiranje podataka

Horizontalno particionisanje može da odgovorina povećan broj zahteva do određene mere. U onom trenutku kada dođe do toga da ni horizontalno skaliranje ne može da pomogne sistemu da opsluži sve zahteve na poželjan način.

Strategije za prevazilaženje velikog broja zahteva I dostizanja vrhunca kada je u pitanju horizontalno particionisanje jeste paginacija podataka, keširanje podataka koji se često čitaju I isti su za sve korisnike I dalje particionisanje baza podataka na manje celine kojima se lakše upravlja.

Hibernate po default-u ima podržan *First-level cache,* samim tim i naša aplikacije. Za *Second-level cache* potreban je spoljni provajder. Naš izbor bi bio EHCache koji predstavlja jedan od najboljih i najpopularnijih provajdera za *Second-level cache.*

Ideja za našu aplikaciju da se retko izmenjivi podaci keširanju i strategija za njih bi bila *Transactional* . Ti podaci bi u našoj aplikaciji bile informacije o samim entitetima kao i o korisnicima. Ostali podaci poput rezervacija, akcija i otkazivanja su visoko izmenjive kategorije i njih nema smisla keširati pošto se u slučaju visokog eksploatisanja aplikacije menjaju skoro pa trenutno.

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

- Skladištenje korisnika: podaci o jednom korisniku zauzimaju 454 bajta, podaci za 100 miliona korisnika će iznositi približno 42.3 GB.
- Skladištenje entiteta: podaci o jednom entitetu u proseku zauzimaju oko 688 bajta, podaci za 100 miliona korisnika će iznositi približno 64.1 GB
- Skladištenje rezervacije: podaci o jednoj rezervaciji u proseku zauzimaju oko 278 bajta, podaci za 100 miliona korisnika će iznositi približno 26 GB

6. Predlog strategije za postavljanje load balansera

Load balanser za koji bismo se mi odlučili je *Weighted least connection* . On distributira zahtev klijenta na server sa najmanjim brojem aktivnih konekcija u trenutku kada ja zahtev dobijen. Administrator dodaje težine na servere u zavisnosti od njihovih karakteristika. *LoadMaster* uzima u obzir kriterijum zadan od strane administratora i broj aktivnih konekcija.

7. Predlog koje operacije korisnika treba nadgledati u cilju poboljsanja Sistema

Trebalo bi imati događaje koji bi pratili radnje korisnika koje su korisne kako za poboljšanje same infrastructure tako I za poboljšanje biznis logike.

Za određena pitanja infrastrukture bilo bi bitno nadgledati operacije korisnika u određenim vremenskim periodima poput ukupnog broja pregleda u određenom periodu dana, nedelje, meseca, na praznike… Time bismo znali u kojim periodima možemo očekivati veći saobraćaj na serverskoj strai I alocirati više resursa da bi opslužili sve zahteve, eventualno menjali neke od gore pomenutih strategija.

Za samu biznis logiku bilo bi bitno nadgledati najpopularnije entitete, pretrage koje se najčešće unose u cilju eventualnog daljeg particionisanja postojećih tabela, kao i preporučivanja entiteta u odnosu na pokazane afinitete, kao I kategoriju u koju korisnik spada. Za samo poboljšanje pojedinih procesa kod korisnika i eventualne izmene, bilo bi važno dobiti najčešće greške koje je korisnik pravio i dobijao kao povratnu informaciju.

Za prikupljanje opštih informacija o našoj aplikaciji (propusnost, zauzeće memorije, iskorišćenje procesora...) možemo koristiti neku od besplatnih *open-source* aplikaciju kao npr. *Prometheus.*

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