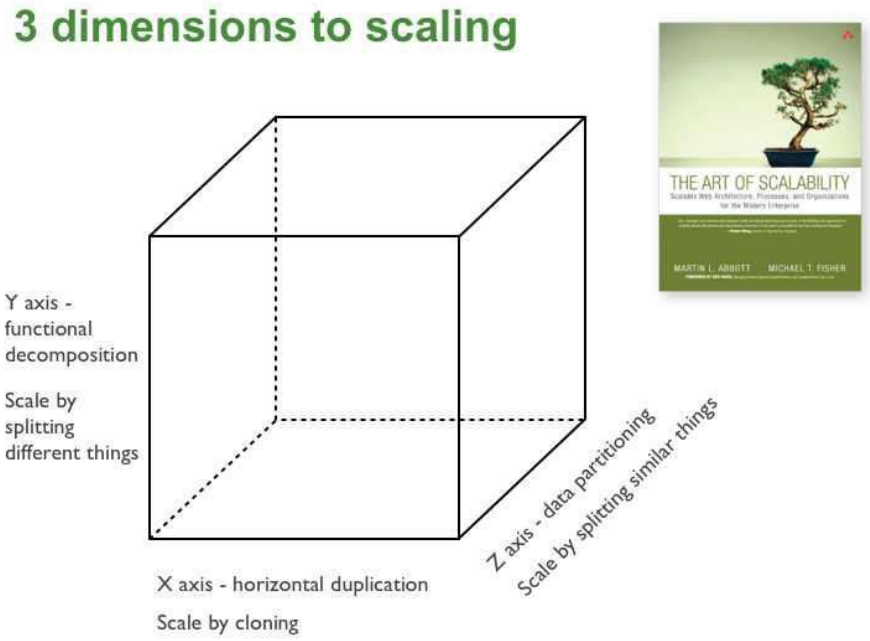
SIIT Tim 7

Proof of Concept

- CDN - odličan za keširanje statičkog sadržaja. Njih ima na više mesta u svetu (CDN points), čime se podaci distribuiraju na razna mesta u svetu, što ubrzava učitavanje. Prvo učitavanje je sporije, jer tad CDN point mora da dobavi podatke od servera, ali nakon toga su podaci keširani. Ovo smanjuje load na serveru. CDN-ovi mogu da se koriste kao kratkoročna rešenja. Oni se implementiraju za nekoliko sati. Sa druge strane, kupovina dodatnih servera radi rukovanja saobraćajem za kratak period nema mnogo smisla, jer kupovina servera povećava vreme potrebno za pripremu, tj. lead time. Osim toga, nije jednostavno proceniti koliki je tačan broj servera potreban.
- Keširanje - sa veb servera ne mora direktno da se komunicira sa serverom baze podataka. Pre nego što zahtev bude poslat do takvog servera, on može da se pošalje nekom caching sloju. Caching slojevi su implementirani u Memcached/Redis/Cassandra (prvo dvoje su in-memory data stores, a Cassandra je disk database). Oni služe da: keširaju podatke u RAM-u, čime smanjuju broj puta koliko je neophodno da se čita eksterni izvor podataka, i na taj način ubrzavaju rad baze podataka. Caching slojevi su ponekad na glavnom serveru, a mogu da budu i na odvojenom. Komplikovani upiti idu u caching slojeve. Ono što se dobjia je da se baza podataka pogodi samo jednom u nekoliko sati, kada je potrebno ažurirati podatke.
- Max broj konekcija za Apache je 1000, što serveri brzo dostignu. Ako se dostigne ovaj broj, razlog može da bude to što je konekcija sa bazom podataka otvorena predugo, a upiti ka bazi podataka su spori. Rešenje je optimizacija upita, tako da upiti ne budu otvoreni predugo.
- Koristiti kvalitetan hardver je dobra ideja – što više RAM-a, to je veća šansa da će baza podataka u celini da se čuva u memoriji
- Dva razloga za skaliranje baze podataka:
    1. Previše saobraćaja (load-a) – rešenje je Master/slave replikacija baze podataka = dva servera baze podataka, pri čemu jedan kopira podatke od drugog. Postoji lag između njih, što i jeste problem ovog pristupa, jer narušava C u ACID. Master je „izvor istine“, tj. ono u šta se upisuje. Slave je kopija. Iz slave-a može da se čita, ali se u njega nikada ne upisuje. Ovo je dobro zato što često nije potrebna apsolutna konzistentnost. Ako se čita iz slave-a, to doprinosi boljem raspoređivanju saobraćaja. Master se bavi samo upisima. Time što postoji barem jedna kopija master-a, lako je da se on dalje replicira, tj. da se stvore novi slave-ovi. Ako ne postoji niti jedan slave, jedina opcija za stvaranje slave-a je da se pauzira master baza podataka, ceo sajt skine sa interneta, iskopira master baza podataka, pa da se onda sve vrati online, što je previse downtime-a (može da traje desetinama sati). Problemi: 1. ne ubrzava write 2. replication lag
    2. Previše podataka – rešenje je Database sharding (= skaliranje po z-osi, tj. Z-axis scaling) = podela baze podataka na delove, gde se svaki deo naziva „shard“. Drugim rečima, Database sharding je rastavljanje jedne baze

podataka na više manjih koje su međusobno potpuno nezavisne i mogu da se nalaze na različitim serverima baza podataka. Logika iza Database sharding-a je da linearnim porastom broja transakcija u jedinici vremena vreme odziva baze podataka raste eksponencijalno. Osim toga, velika baua podataka zahteva skuplji hardver, dok shard-ovi mogu da budu distribuirani na jeftinijem hardveru. Database sharding može da se radi geografski, npr. poseban server za kupce sa Bliskog Istoka, poseban za kupce iz Amerike. Database sharding može da se radi i na mnogo jednostavnijem osnovu, npr. ostatak pri deljenju id-a korisnika brojem 10 određuje na koji od 10 servera će taj korisnik da bude poslat. Problemi: 1. upiti postaju složeniji - npr. range query (npr. vrati sve radnike koji imaju između 3 i 5 godina iskustva) će morati da prođe kroz sve baze podataka, naročito ako su podaci raspoređeni po njima pomoću nekog heš algoritma 2. join-ovi postaju teško izvodljivi, jer tabele više nisu na istom mestu

- Load balancing (= skaliranje po x-osi, tj. X-axis scaling) – raspoređuje zahteve na različite servere. Kako load balancer bira server: 1. random – ako se brojevi generišu na dovoljno slučajan način, svaki od n servera bi trebalo da dobije oko 1/n load-a 2. round-robin 3. load-based – kod ovog načina, load balancer ima uvid u to koji server obrađuje koliki load, pa na osnovu toga donosi odluku
- Prelazak na mikroservisnu arhitekturu (= skaliranje po y-osi, tj. Y-axis scaling) - prednosti: 1. ubrzava razvoj – s obzirom na to da su mikroservisi međusobno nezavisni, mogu da se razvijaju i testiraju zasebno, po sopstvenom rasporedu 2. olakšava skaliranje – ako neki mikroservis mora da rukuje velikom količinom saobraćaja, moguće je skalirati samo njega. Sa druge strane, u monolitnom razvoju bi morala da se skalira cela aplikacija. Mane: 1. usložnjava aplikaciju (mreža zavisnosti između delova aplikacije 2. sa organizacionog aspekta može da bude teže pratiti razvoj n aplikacija, nego pratiti razvoj jedne. 3. keširanje na svakom do n servisa može da bude složeno



- Connection pooling – iako Tomcat automatski vrši connection pooling, taj proces je moguće optimizovati u skladu sa tim kada očekujemo manje, a kada više saobraćaja. Kada očekujemo manje saobraćaja, poželjno bi bilo smanjiti broj niti u pool-u, i obrnuto. Veličina pool-a može da se povećava dok to hardver (broj procesorskih jezgara) dozvoljava.