nedeljainformatike v8.0

Kako preživeti million korisnika?

Mladen Puzić

Matematička gimnazija

15. 05. 2023.

Agenda

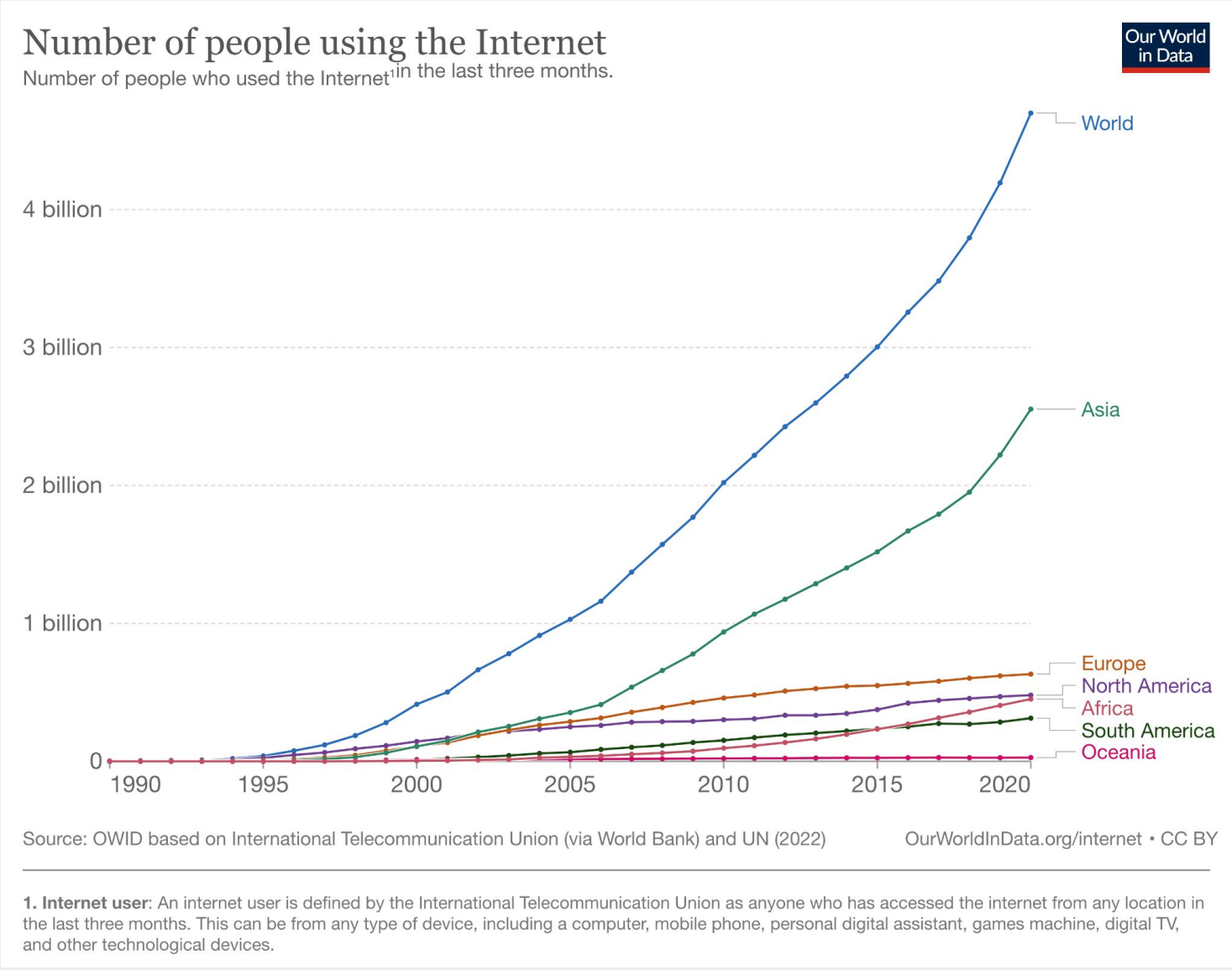
1. Motivacija
2. Mikroservisi
3. Distribuirani sistemi
4. Service Fabric

Motivacija

- Broj ljudi koji koriste internet naglo raste iz godine u godinu, pretežno u Aziji
- Sa porastom broja korisnika nekog servisa, nastaju nove komplikacije, konkretno, potrebni su veći resursi, koliko računarski (procesori, memorija, ...), toliko i ljudski (programeri, administratori, ...), pa i finansijski

- Kompanijama je u interesu da zarade novac, što se delom postiže smanjivanjem troškova pružanja usluga (*cost of goods sold - COGS)*
- Kako bi se COGS smanjio (kako bismo „preživeli''), potrebno je da koristimo određene tehnike kako bismo:
  - efikasnije koristili računarske resurse koje imamo
  - pojednostavili održavanje i rad na servisu
  - sačuvali što više $$$

Broj korisnika interneta kroz godine



Dve stvari za razmatranje

- Problem *Kako preživeti milion korisnika* posmatraćemo iz dva ugla:
  - Kako preživeti **milion** korisnika? - mikroservisi
  - Kako **preživeti** milion korisnika? - distribuirani sistemi

- Nakon toga, pogledaćemo Majkrosoftov open-source proizvod *Azure Service Fabric* i kako on spaja ova dva koncepta

Agenda

1. Motivacija
2. Mikroservisi
3. Distribuirani sistemi
4. Service Fabric

Šta su mikroservisi?

- Arhitektura mikroservisa je tip arhitekture pri kojoj se glavni servis podeli na više manjih *(mikro)* servisa
- Ovi mikroservisi imaju veoma razvijene API-je *(Application Programming Interface)* kako bi što lakše međusobno komunicirali
- Ovakvi servisi se mogu zamisliti kao fabrika, u kojoj svako ima neku svoju (potencijalno različitu) ulogu, ali svi rade zajedno zarad zajedničkog cilja
- Svaki mikroservis bi trebalo:
  - da bude funkcionalno samostalan - pad jednog mikroservisa ne bi trebalo da utiče na pad drugog (ali može uticati na funkcionalnost glavnog servisa)
  - da bude samostalan po dizajnu - promena koda jednog mikroservisa bi trebalo da minimalno zahteva promenu koda drugog
  - da poseduje mali tim programera
  - da bude samostalno *deployable*

Prednosti mikroservisa

- Funkcionalna samostalnost pomaže da greška u jednom mikroservisu ne narušava funkcionalnost drugih delova servisa
- Samostalnost po dizajnu pomaže da promena koda (npr. ispravljanje greške ili novi *feature*) ne zahteva promenu drugih mikroservisa, što omogućava da mikroservise poseduju mali, samostalni timovi
- Samostalni *deployment* pomaže da se delovi koda lakše testiraju odvojeno što olakšava proces razvijanja, takođe je moguće odvojeno skalirati broj instanci mikroservisa, ukoliko je različita potražnja
- Nezavisnost i manja veličina mikroservisa pomaže kod bržeg *onboardinga* programera koji na njima rade

Sve u svemu, mikroservisi pomažu pri razvoju jednostavnijim testiranjem i raspodelom posla, dok u izvršavanju pomažu lakšim skaliranjem pri povećanim brojem korisnika.

Prednosti mikroservisa

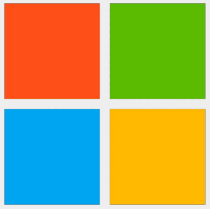
	Monolithic design	Microservice-based design
Application complexity	Complex	Modular
Fault-tolerance	Complex	Modular
Agile development	No	Yes
Communication between components	NA	RPCs
Easily scalable	No	Yes
Easy app lifecycle management	No	Yes
Cloud ready	No	Yes

Table 1: Monolithic Vs. Microservice Applications.

Gde se koristi?

Uber

amazon



Microsoft

NETFLIX

SOUNDCLOUD

I još mnogo…

Agenda

1. Motivacija
2. Mikroservisi
3. Distribuirani sistemi
4. Service Fabric

Šta je distribuirani sistem?

- Grubo rečeno, distribuirani sistem je skup računara koji su povezani tako da funkcionišu zajedno, kao da su jedan računar
- Ovi računari ne moraju biti fizički na istoj lokaciji, dokle god su povezani, uglavnom internetom (štaviše, ovi računari su često na različitim stranama sveta)
- Lokacija računara se uglavnom deli na geografske zone (države, ili delove država), koji mogu biti povezani ekstremno brzim konekcijama (u Azure se zovu *Availability Zones)*

Prednosti distribuiranih sistema

Prednosti su slične kao kod servisa:

- lako skaliranje resursa dodavanjem još računara u sistem (horizontalno skaliranje) ili pojačavanjem računara u sistemu (vertikalno skaliranje)
- pri kvaru jednog računara ili komunikacije između računara, moguće je nastaviti sa radom
- Ukoliko se koriste računari iz različitih geografskih zona, sistem je otporan i na događaje poput poplava, požara, zemljotresa, ratova
- Takođe je moguće koristiti najbliži računar korisniku za komunikaciju, kako bi komunikacija bila što efikasnija
- Moguće je dizajnirati sistem tako da bude pogodan za paralelno programiranje

Komplikacije sa distribuiranim sistemima

- Sinhronizacija mašina - ne postoji globalni časovnik
- Paralelni pristup resursima - potrebno je obezbediti da više računara mogu pristupati resursima u isto vreme (bazama podataka, ...)
- Bezbednost - pošto se komunikacija često održava preko mreže, potrebno je obezbediti njenu sigurnost
- Cena - cena distribuiranih sistema na početku je viša, zbog potrebne dodatne infrastrukture za povezivanje, ali se smanjuje što je sistem veći

Ilustracija

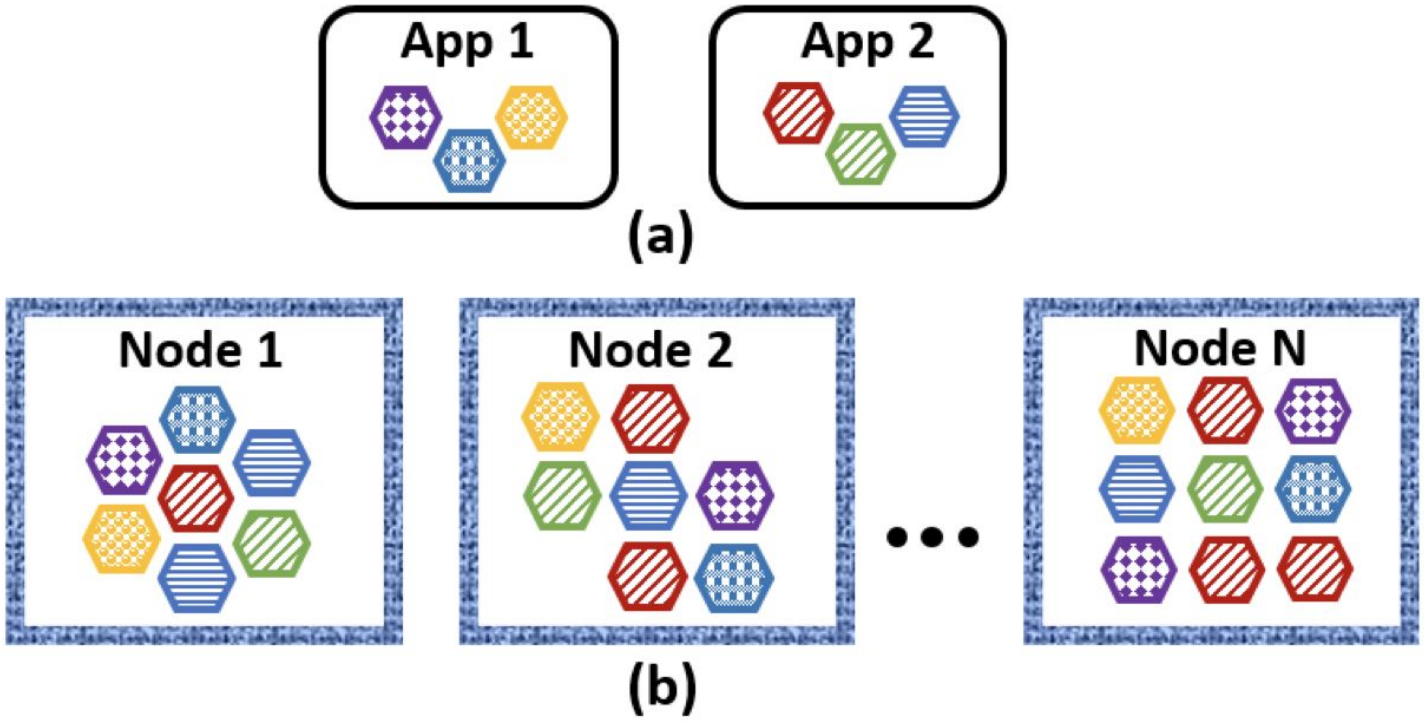
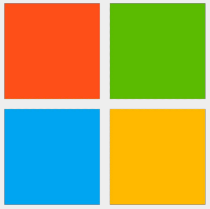


Figure 1: A Microservice-based Application. a) *Each colored/tiled hexagon type represents a microservice, and* b) *Its instances can be deployed flexibly across VMs.*

Gde se koristi?

Uber

amazon



Microsoft

NETFLIX

SOUNDCLOUD

I još mnogo…

Agenda

1. Motivacija
2. Mikroservisi
3. Distribuirani sistemi
4. Service Fabric

Šta je *Service Fabric?*

- Azure Service Fabric je Microsoftov open-source PaaS (*Platform as a Service*) koji omogućava pokretanje mikroservisa na distribuiranim sistemima
- Servisi se lako pakuju, deployuju, održavaju, skaliraju, testiraju...

Prednosti *Service Fabrica*

- Kao računare u SF možemo koristiti bilo šta - naše fizičke mašine, mašine u nekom data centru, virtuelne mašine, bilo Azure-ove, bilo tuđe, bilo sa Windowsom, bilo sa Linuxom
- Podržava mikroservise sa pamćenjem *(stateful service)* - mikroservise sa unutrašnjom memorijom, koja se održava čak i u slučaju nasilnog zaustavljanja rada
- Mikroservisi se mogu pisati u bilo kom jeziku

Struktura *Service Fabrica*

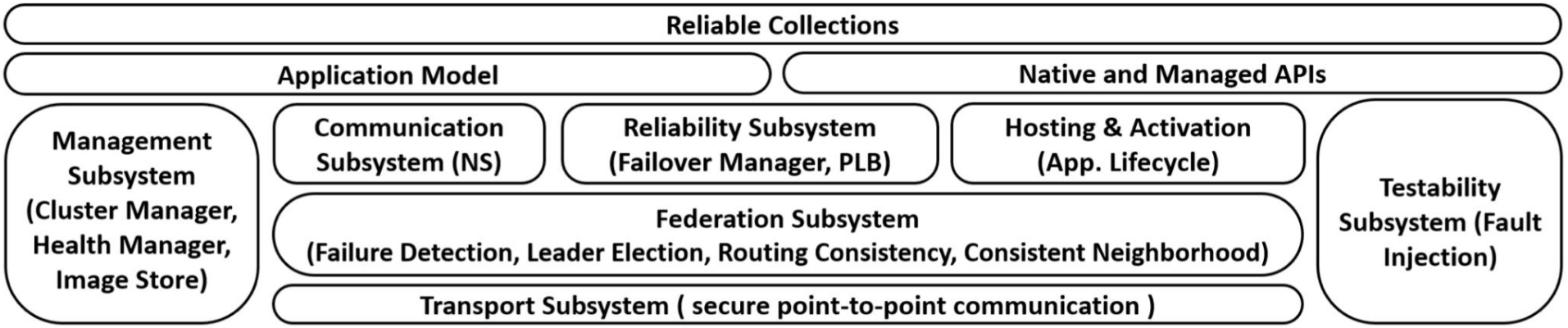


Figure 3: Major Subsystems of Service Fabric. *NS = Naming Service, PLB = Placement and Load Balancer.*

Struktura *Service Fabrica*

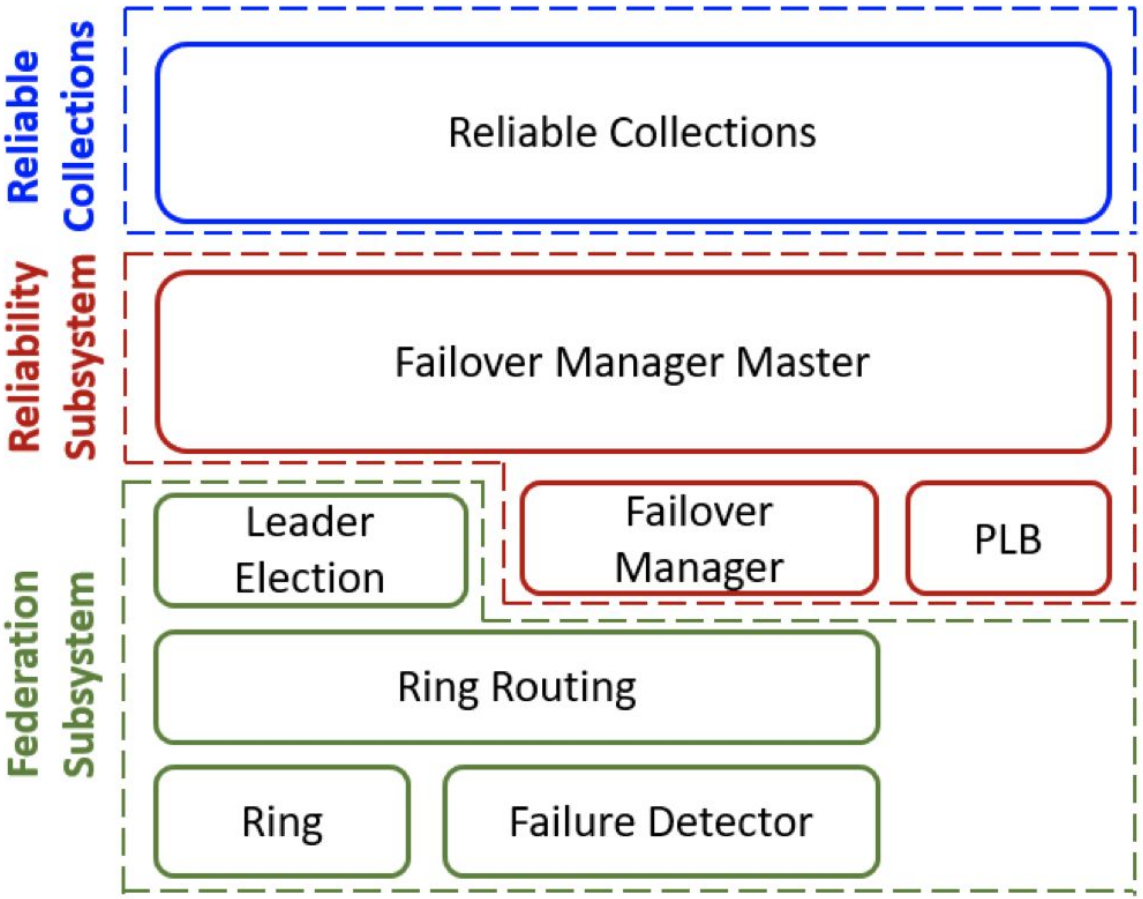


Figure 4: Federation and Reliability Subsystems: Deep-Dive.

Federation subsystem

- Čvorovi (računari, *node*) se održavaju u SF-prstenu (*SF-ring*), što možete zamisliti kao kružni niz sa 2^m elemenata (recimo *m = 128)*
- Svakom računaru dodeljena je pozicija u SF-prstenu
- Za svaki računar se održava skup suseda, odnosno skup nekoliko najbližih čvorova sa obe strane u SF-prstenu
- Jako je bitno da tačno održavamo ovu listu suseda (što nije toliko jednostavno, jer radimo sa distribuiranim sistemom) i da ona bude simetrična

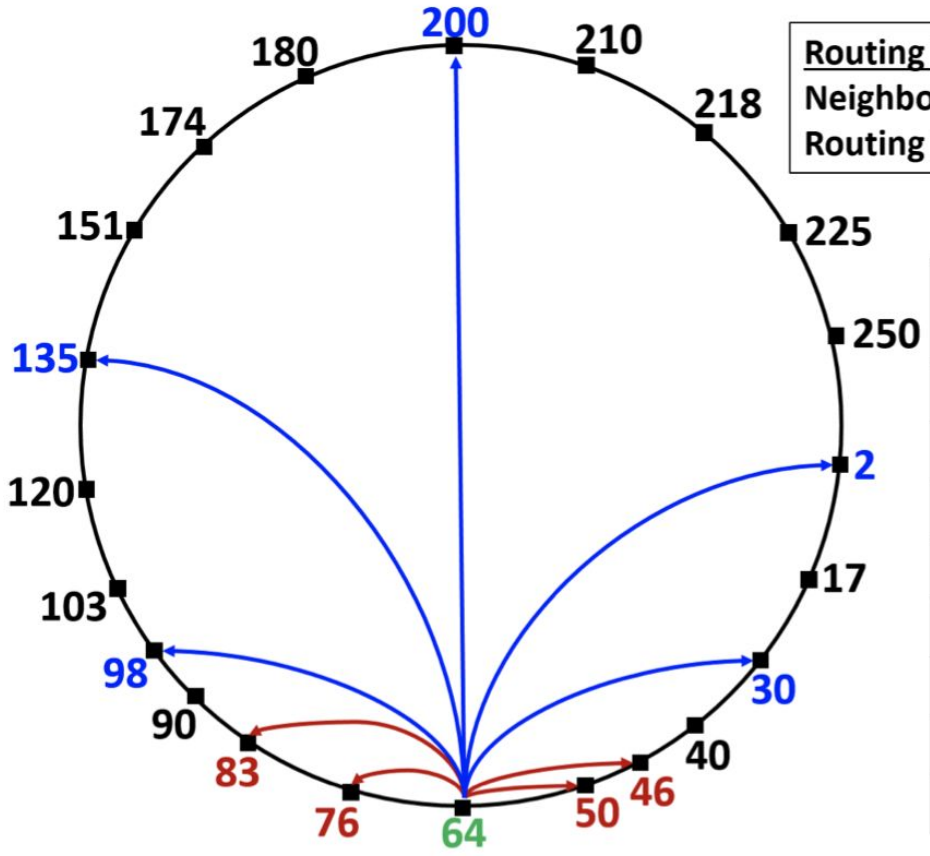
Obnova zakupa

- Odvajamo odluku o tome da li je čvor živ od detekcije da li je čvor živ
- Čvorovi susedi čvora *x* pokušavaju da li vide da je *x* živ i sve informacije šalju trećem entitetu (*arbitratoru*) koji odlučuje da li je *x* zapravo živ
- Ovo je kako bi se umanjio uticaj problema distribuiranih sistema - naime moguće je da neki čvor ne vidi suseda zbog greške u komunikaciji, a da je sused sasvim funkcionalan
- Svakih *T* sekundi (često 30s), svaki čvor šalje zahtev svojim susedima za obnavljanje zakupa (*lease request*) svog mesta na SF-prstenu
- Kada njegov sused odobri zahtev, on obećava da ga neće označavati kao mrtvog neko vreme
- Ukoliko se zbog greške u komunikaciji zahtev preskoči, on se šalje ponovo još nekoliko puta
- Ukoliko čvor ne dobije potvrdu od svih suseda, on razmatra da sam sebe označi kao mrtvog
- Obratiti pažnju da se ovo radi asinhrono, pošto ne postoji globalni časovnik

Detaljnije o SF-prstenu

- SF-prsten je u stvari distribuirana heš tabela (DHT), koja svakoj vrednosti od 0 do 2^m dodeljuje neki čvor, konkretno čvor čiji indeks u SF-prstenu je najbliži traženom indeksu
- Čvorovi neretko moraju da komuniciraju sa čvorovima koji nisu neophodno njihovi susedi, pa je potrebno imati metodu da nađu te čvorove
- Za svaki čvor čuvamo i partnere za usmeravanje (*routing partner*) - ako je indeks čvora neko *idx*, to su čvorovi koji su najbliži pozicijama idx-2^0, idx-2^1,idx-2^2,..., idx-2^{m-1} i idx+2^0, idx+2^1,idx+2^2,..., idx+2^{m-1}
- Poruka koju čvor dobije se šalje partneru za usmeravanje koji je najbliži destinaciji, pa se proces nastavlja rekurzivno
- Za čvor koji je najbliži nekoj tački *k*, kažemo da je vlasnik, vođa (*leader)* za tu tačku

Detaljnije o SF-prstenu (ilustracija)



Routing Table at node 64:
Neighborhood = {83,76,50,46}
Routing Partner = {200,2,30,98,135,200}

Routing Partner Calculation

Clock.	Active Node	Anti-Clock.	Active Node
$64+2^0$	64	$64-2^0$	64
$64+2^1$	64	$64-2^1$	64
$64+2^2$	64	$64-2^2$	64
$64+2^3$	76	$64-2^3$	50
$64+2^4$	83	$64-2^4$	46
$64+2^5$	98	$64-2^5$	30
$64+2^6$	135	$64-2^6$	2
$64+2^7$	200	$64-2^7$	200

Figure 5: Routing Table *of node 64. The ring has* $2^{m=8}$ *points. Numbered dots represent active nodes.*

Reliability subsystem

- Održava dostupnost podataka, obezbeđuje raspodelu poslova, replikaciju
- Sastoji se od tri najvažnija dela - Naming, Placement and Load Balancer (PLB) i Failover Manager
- Servisi se prave u više kopija (replika) - primarna replika je ona čiji se rezultati koriste, ali postoji više sekundarnih replika koje su na drugim čvorovima i koriste se u slučaju da primarna replika postane nedostupna (na primer ako njen čvor više nije živ ili ga nije moguće kontaktirati)
- U slučaju stateful servisa, svi podaci se održavaju između primarne i sekundarnih replika
- **Naming** - svaki servis dobija jedinstven URI (Uniform Resource Identifier) po kojem se može doći do njega
- **PLB** - zadužen je za odabir čvorova na koji će da idu nove replike, koristi informacije o dostupnosti resursa na čvorovima (cpu, memory, replica count...)
- Ovo je veoma komplikovan optimizacioni problem koji liči na probleme poput knapsacka, bin packinga...

Failover Manager

- **Failover Manager** - održava globalni pregled stanja, aplikacija, servisa, replika i na kojim čvorovima se nalaze, koordinira sa PLB i prikuplja podatke o korišćenju resursa sa čvorova, kao i odluke o uklanjanju čvorova od arbitra i obaveštenja o priključenju od novih čvorova
- FM se bavi sledećim ključnim radnjama:
  - Postavljanje replike - u slučaju pada replike ili pravljenja nove replike. Konsultuje se sa PLB
  - Pomeranje replike - u slučaju narušavanja balansa, PLB napravi plan rebalansiranja koji FM izvrši
  - Rekonfiguracija - ukoliko primarna replika padne, FM bira sekundarnu repliku da postane primarna. Ukoliko se primarna replika vrati, postaje sekundarna replika

- Ukoliko ceo FM padne, potrebno ga je brzo vratiti i za to koristimo **FMM (Failover Manager Master)**, koji se nalazi na čvoru koji je vođa indeksu 0 u SF-prstenu
- FMM se ponaša isto kao FM, samo što upravlja FM-ovima umesto servisima
- Ukoliko FMM padne, mora da se gradi od nule

Reliable collections

- API na osnovu kojeg se grade stateful servisi, sadrži Reliable Dictionary i Reliable Queue
- Podaci se čuvaju lokalno na mašini, tako da su read operacije jako brze, ali su takođe veoma dostupni preko procesa pripreme sekundarnih replika
- Omogućavaju preživljavanje podataka čak i u slučaju pada celih čvorova, a takođe su i veoma efikasni

Sledeći koraci

- Glavna osnova za ovu prezentaciju: link
- Veoma razvijena dokumentacija: link
- Repozitorijum za SF: link
- Možete se igrati i napraviti cluster čak i sa samo jednim računarom

Hvala na pažnji!

Pitanja?