UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
U NOVOM SADU

Matija Maksimović E2 33/2024

CRDT i njegova primena u distribuiranim sistemima

SEMINARSKI RAD

- Master akademske studije -

Novi Sad, 2024.

UNIVERZITET U NOVOM SADU o **FAKULTET TEHNIČKIH NAUKA**
21000 NOVI SAD, Trg Dositeja Obradovića 6

Predmet: Arhitekture sistema velikih skupova podataka
Predmetni profesor: prof. dr Vladimir Dimitrieski

SADRŽAJ

1.Uvod

U savremenim distribuiranim sistemima, balansiranje između konzistentnosti, dostupnosti i tolerancije na mrežne particije predstavlja ključni izazov. Tranzicija sa centralizovanih baza podataka ka distribuiranim donela je potrebu za novim modelima replikacije podataka, koji nude otpornost na greške i skalabilnost bez žrtvovanja performansi. Jedan od fundamentalnih problema distribuiranih sistema je usklađivanje promena između različitih čvorova koji rade nezavisno i mogu imati privremene mrežne prekide.

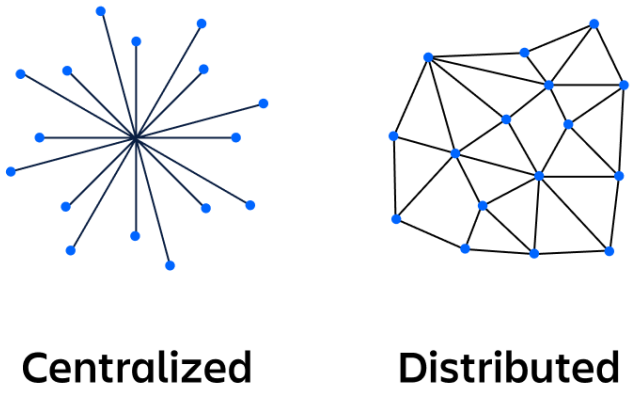
Beskonfliktni kopirajući tipovi podataka (**engl.** Conflict-free Replicated Data Type, CRDT) predstavljaju jedno od ključnih rešenja u distribuiranim sistemima. Ovaj tip čuva doslednost podataka bez potrebe za centralizovanim mehanizmima zaključavanja i koordinacije. Kroz unapred definisana pravila spajanja podataka, *CRDT* strukture garantuju konvergenciju stanja, čak i u prisustvu mrežnih particija. Ovaj model je postao naročito značajan u sistemima kao što su kolaborativne aplikacije (Google Docs, Figma), distribuiranim bazama podataka i federativnom učenju, gde se modeli treniraju na različitim uređajima bez centralizovanog deljenja podataka. U radu se analizira sam koncept *CRDT*-a, njegove osnovne vrste i svojstva, kao i praktična upotreba.

Ovaj tip podataka samo je mali deo ogromne količine znanja potrebne za kreiranje aplikacija sa intenzivnom obradom podataka. Istoimena knjiga, vrlo popularna među inženjerima, govori nam da aplikacije moraju biti skalabilne, održive i pouzdane. Međutim, gomila naučenih sleng reči (**engl.** buzzword) neće učiniti programera dovoljno veštim da se ispetlja iz problema koje distribuirane aplikacije kreiraju. Sa većim apstrahovanjem programskih jezika i raznih radnih okvira, logično je bilo očekivati da će problemi postati manji, ali desilo se upravo suprotno i ova olakšanja dovela su do složenijih i teže razumljivih sistema. Distribuirani sistemi nisu samo pitanje teorijske konzistentnosti ili mrežnih protokola, oni su nepredvidivi, skloni greškama i zahtevaju duboko razumevanje. Ipak, ova oblast računarstva zasigurno ima svetlu budućnost, koliko god izgledala izazovno na prvi pogled. Sa pravim alatima i tehnikama, moguće je graditi rešenja i prevazići projektne izazove, što je u suštini i poenta inženjerstva.

2.Distribuirani sistemi

Distribuirani sistem je skup nezavisnih računara koji korisnicima izgledaju kao jedinstven, koherentan sistem. Ovi računari, čvorovi u mreži, međusobno komuniciraju, usklađuju svoje aktivnosti i dele podatke i zadatke kako bi ostvarili zajednički cilj.

U centralizovanom sistemu, svi podaci i računarski resursi su smešteni i kontrolisani na jednom centralnom serveru. Ovaj model je jednostavan za održavanje ali može postati problematičan kada veliki broj korisnika istovremeno pristupa sistemu ili ako centralni server otkaže. Distribuirani sistem, s druge strane, raspoređuje podatke i resurse na više servera ili lokacija. Ova arhitektura pruža mnogo veću skalabilnost i pouzdanost, jer sistem može nastaviti sa radom čak i pri otkazu njegovih delova. Međutim, distribuirani sistemi su manje bezbedni i složeniji za upravljanje zbog velikog broja čvorova. [1]

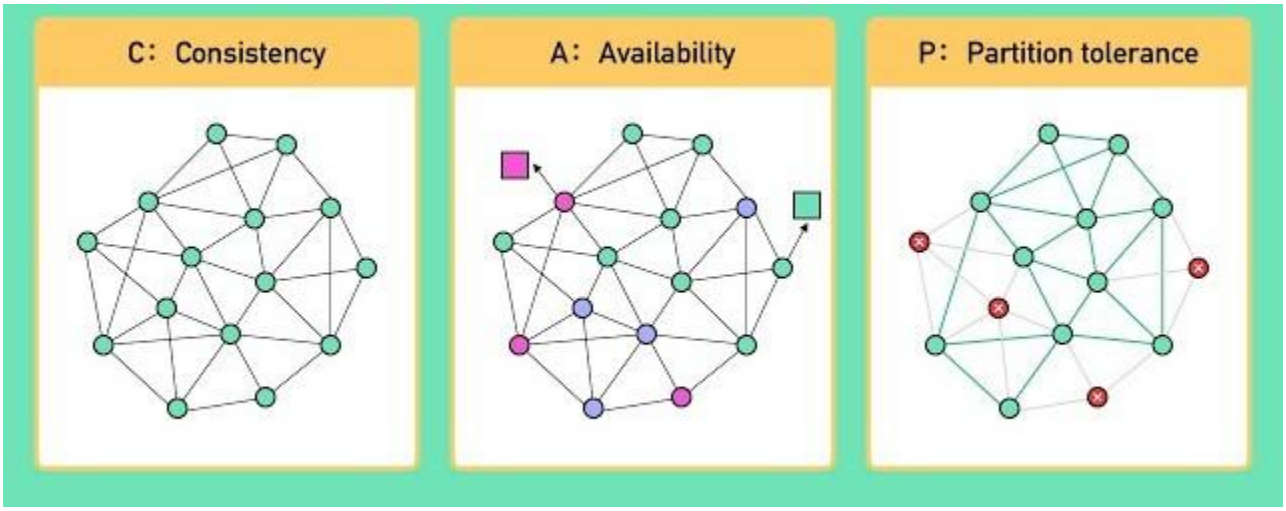


Slika 1: Centralizovan i distribuiran sistem (preuzeto sa https://www.atlassian.com/microservices/microservices-architecture/distributed-architecture)

Postoji nekoliko uobičajenih arhitektura distribuiranih sistema:

- Arhitektura klijent-server - Serveri pružaju resurse i usluge, dok ih klijenti zahtevaju putem mreže.

- Arhitektura čvor na čvor (**engl.** Peer-to-peer, P2P) – Svaki čvor direktno deli resurse sa svojim susedima, tj. deluje i kao klijent i kao server.
- Slojevita arhitektura – Aplikacija se razdvaja na slojeve, na primer, sloj korisničkog interfejsa, sloj poslovne logike i sloj baze podataka.
- Mikroservisna arhitektura – Aplikacija je podeljena na male nezavisne servise, koji komuniciraju putem mreže.
- Arhitektura vođena događajima – Komponente reaguju na događaje umesto direktnih zahteva.



Slika 2: *CAP* teorema (preuzeto sa **https://i.ytimg.com/vi/BHqjEjzAicA/sddefault.jpg**)

Fundamentalno ograničenje distribuiranih sistema u teoriji postavlja *CAP* teorema (**engl**. Consistency, Availability, Partition tolerance):

- U distribuiranom sistemu nije moguće istovremeno garantovati sva tri svojstva:
  - Konzistentnost (**engl.** Consistency, C) – Svi čvorovi vide isti podatak u isto vreme.
  - Dostupnost (**engl.** Availability, A) – Svaki zahtev dobija odgovor, čak i ako neki čvorovi otkažu.
  - Otpornost na mrežnu particiju (**engl.** Partition tolerance, P) – Sistem nastavlja rad čak i ako postoji prekid komunikacije između čvorova.

- Sistem može imati samo 2 od 3 svojstva:
    - *CA* (Konzistentnost + Dostupnost) - Moguće samo u centralizovanim sistemima, ali ne u distribuiranim zbog particija.
    - *CP* (Konzistentnost + Otpornost na particiju) - Sistem može žrtvovati dostupnost.
    - *AP* (Dostupnost + Otpornost na particiju) - Sistem može postati privremeno nekonzistentan.

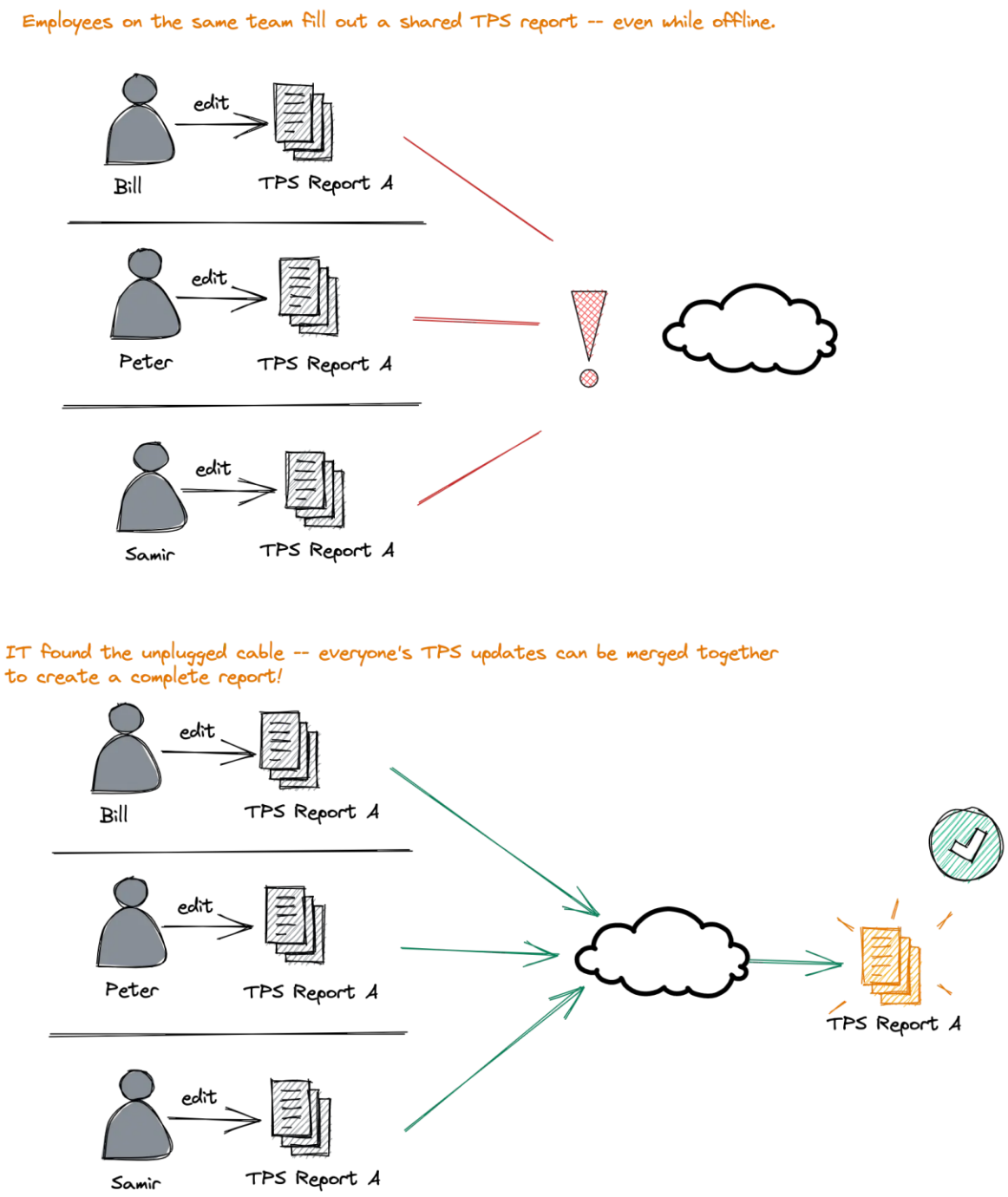
3.Osnove *CRDT*-a

CRDT je struktura podataka dizajnirana za distribuirane sisteme kako bi omogućila eventualnu konzistentnost bez potrebe za centralizovanim sinhronizacijama. Ove strukture osiguravaju da svi čvorovi sistema konvergiraju ka istom stanju, čak i ako se izmene podataka dešavaju nezavisno na različitim replikama.

Starije tehnike konzistentnosti distriburianih sistema zahtevaju strogo serijalizovanu obradu zahteva, što može postati usko grlo (**engl.** bottleneck). Umesto toga, *CRDT* nudi lokalne izmene koje se kasnije spajaju pomoću unapred definisanih pravila, čime se postiže visoka dostupnost i skalabilnost. Treba da imamo u vidu da je *CRDT*, iako dobro radi svoj posao, nesavršeno rešenje, koje je korisno samo za određene situacije, gde komutativnost operacija može biti osigurana. Neke vrste ovih tipova zahtevaju stalno čuvanje istorije operacija što može izazvati memorijsko opterećenje.[2]

Postoji veliki broj implementacija za ove tipove i u teoriji su se izdvojila dva osnovna modela:

- *CRDT* baziran na stanju (**engl.** Convergent Replicated Data Type, CvRDT):
  - Brojač tipa *G* – Brojač dozvoljava samo povećavanje vrednosti.
  - Brojač tipa *PN* – Brojač koji dozvoljava i povećavanje i smanjivanje vrednosti.
  - Skup elemenata tipa *LWW* – Skup gde svaki element ima vremensku oznaku i oslanja se na pravilo da poslednji upis pobeđuje (**engl.** last write wins, LWW).
- *CRDT* baziran na operacijama (engl. Commutative Replicated Data Type, CmRDT):
  - Skupa tipa *G* – Skup koji samo raste.
  - Skupa tipa *OR* – Skup koji prati i dodavanje i uklanjanje elemenata.



Slika 3: Konvergencija stanja (preuzeto sa **https://vlcn.io/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fbusiness-doc-central.83ec55f0.png&w=3840&q=75**)

Klasičan primer za konvergenciju stanja je korpa za kupovinu. Jedan korisnik doda dva proizvoda u svoju korpu, ali se dodavanja obrađuju na različitim serverima zbog particionisanja. Sada postoje dve

verzije korpe, jedna sa proizvodom A i druga sa proizvodom B. Logika spajanja je jednostavna: spojiti obe korpe kako bi se dobila ispravna vrednost. Ali šta se dešava sa uklanjanjem? Kako utvrditi da li proizvod A nedostaje u drugoj korpi zato što je dodat na prvom serveru, ali još nije sinhronizovan, ili je zaista uklonjen, ali prvi server to još ne zna? Ako aplikacija postane složenija, dodaje se brojanje instanci proizvoda u korpi. Funkcija spajanja tada mora odlučiti, ako jedna verzija ima šest primeraka proizvoda, a druga četiri, da li je korisnik dodao šest pa uklonio dva, ili je prvo dodao četiri, a zatim još dva? Možda je rešenje evidentiranje svake operacije, ali sistem brzo postaje složen. Svaki novi model podataka zahteva novu, ručno definisanu funkciju spajanja. Ako na primer želimo da u distribuiranoj bazi Riak skladištimo novi model podataka, recimo korisničke profile, ponovo moramo izgraditi novu logiku spajanja od nule. [3]

Plastičniji primer može biti situacija gde postoji deljeni poslovni dokument. U idealnom slučaju, radnik i njegove kolege imaju svoje kopije dokumenta, i oni ih mogu modifikovati i u slučaju kada nema pristupa internetu. Kada se pristup internetu vrati, sve promene se beskonfliktno spajaju u jednu konačnu verziju. Ova priča neodoljivo liči na sisteme za kontrolu verzija, kao što je git, ali bez noćne more konflikta spajanja.

Sa fokusom na praktičnu definiciju, ovaj tip podataka možemo opisati u tri tačke. *CRDT* je tip podataka:

- Koji može biti kopiran na više čvorova.
- Koji može nezavisno biti modifikovan od tih čvorova bez koordinacije i strogo definisanog vremena.
- Čije se sve divergentne kopije mogu spojiti u jedno stanje u bilo kom redosledu i od strane bilo kog čvora. Kada svi čvorovi vide sve divergentne kopije, zagarantovano je da će svi konvergirati u isto konačno stanje.

Jedan od jednostavnijih tipova je skup tipa G, tj. skup koji samo raste. To je struktura podataka koja omogućava dodavanje elemenata, ali ne i njihovo uklanjanje. Ovakav skup može se lako deliti između čvorova, svaki čvor može nezavisno dodavati elemente, a svi skupovi se mogu spojiti bez konflikata. Kada se svi skupovi konačno usklade, dobijeni rezultat predstavlja uniju svih pojedinačnih skupova, čime se postiže eventualna konzistentnost.

Međutim, realni sistemi zahtevaju više od jednostavnog dodavanja podataka. U aplikacijama kao što su baze podataka, potrebno je omogućiti ažuriranje i brisanje podataka na način koji je otporan na konflikte. Jedan od najčešće korišćenih pristupa je princip poslednjeg upisa, pravilo po kojem se uzima samo poslednje upisano stanje. Iako vrlo intuitivan, ovaj pristup ima svoje probleme, naročito kada se koristi sistemsko vreme za određivanje poslednje promene. Nesinhronizovani satovi među uređajima, ručna podešavanja vremena i sistemski problemi mogu dovesti do nekonzistentnog stanja u distribuiranom sistemu.

Da bi se rešili ovi problemi, umesto oslanjanja na realno vreme, distribuirani sistemi koriste logičke satove. Svaki čvor održava sopstveni brojač događaja i povećava ga pri svakoj izmeni podataka. Kada se dva čvora sinhronizuju, razmenjuju i ažuriraju svoje vrednosti tako da osiguraju konzistentnost. Ovo omogućava jasan odnos uzročno-posledičnih događaja i izbegava greške nastale zbog razlika u sistemskom vremenu.

Dodatni problem u distribuiranim sistemima je sinhronizacija podataka preko posrednika (**engl.** proxy). Kada jedan čvor prenese podatke drugom, ali istovremeno preuzima podatke iz treće replike, može doći do neslaganja u tome šta je koji čvor već video. Rešenje je korišćenje dve vremenske oznake: jedne koja označava stvarno vreme izmene podataka i druge koja pokazuje kada je podatak stigao na čvor. Na ovaj način, podaci mogu biti pravilno filtrirani i sinhronizovani bez gubitka informacija. [4] *CRDT* izbacuje potrebu za vremenskom koordinacijom i fokusira se na pravila spajanja podataka koja garantuju doslednost. Umesto oslanjanja na spoljašnje faktore, koriste se kauzalni odnosi između događaja, čime se kreira stabilno i deterministično rešenje za replikaciju podataka u distribuiranim sistemima.

4.Praktična upotreba

Kolaborativne aplikacije (uređivanje dokumenata) – Jedna od najpoznatijih primena *CRDT*-a je u sistemima za kolaborativno uređivanje teksta, kao što su Google Docs i Figma. Kada više korisnika istovremeno menja isti dokument, njihove izmene moraju biti objedinjene bez gubitaka podataka. *CRDT* sekvence omogućavaju da svi uređaji nezavisno dodaju, brišu i pomeraju tekst, dok se rezultati automatski usklađuju.

Distribuirane baze podataka (Riak, AntidoteDB, Redis) – Mnoge moderne NoSQL baze podataka koriste *CRDT*-ove za replikaciju podataka između čvorova. Na primer, Riak koristi *CRDT* za brojače, skupove i mape, i tako obezbeđuje klijentima da ažuriraju podatke bez brige o konfliktima. Slično tome, AntidoteDB koristi *CRDT* mape i liste za implementaciju bankarskih sistema gde je važno da sve transakcije budu dosledne širom distribuirane mreže.

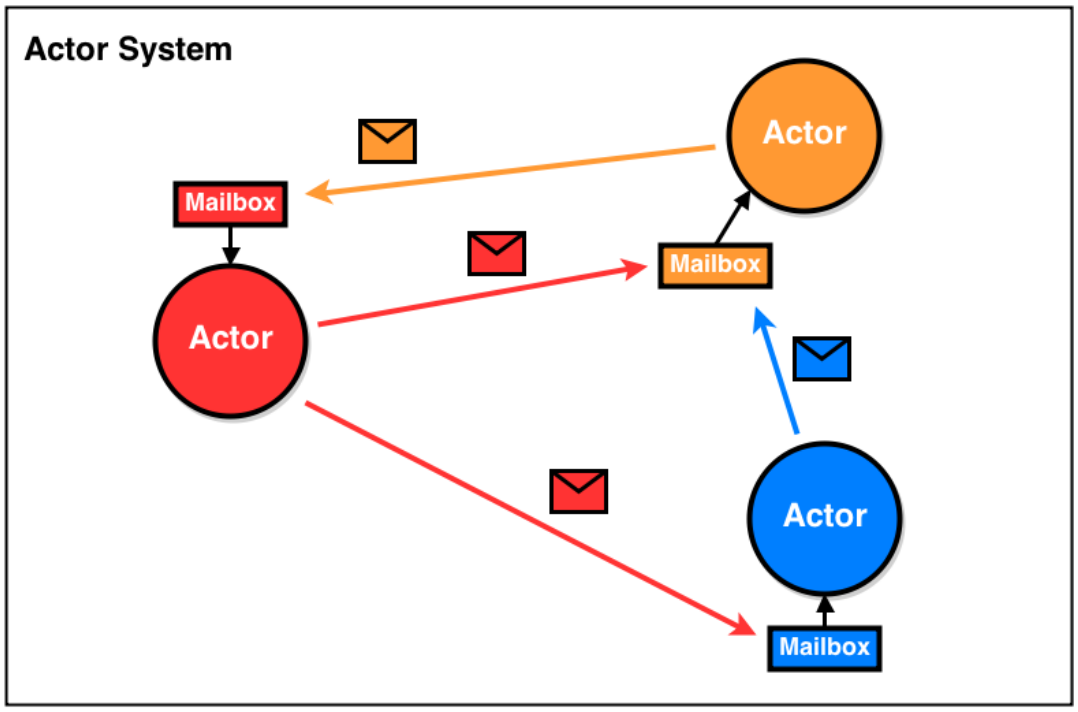
4.3. **Društvene mreže** – Sistemi za razmenu poruka, kao što je WhatsApp, oslanjaju se na *CRDT* tehnologiju kako bi obezbedili da svi učesnici vide iste poruke čak i u slučaju prekida veze. Konkretno, koriste skup tipa *OR* kako bi se osigurala eventualna konzistentnost poruka i notifikacija između različitih klijenata.

4.4. **Internet stvari (engl.** Internet of Things, IoT) i senzorski sistemi – *IoT* uređaji često rade u uslovima niskog protoka podataka i periodičnih mrežnih prekida. *CRDT* kreira lokalno ažuriranje stanja uređaja, a kasnije, kada se ponovo povežu na mrežu, njihove promene se usklađuju bez konflikata. Na primer, senzori temperature u pametnim zgradama mogu koristiti brojač tipa *G* za praćenje prosečne temperature na različitim lokacijama.

4.5. **Federativno učenje sa aktorima** – Aktori su entiteti koji asinhrono obrađuju poruke, nudeći konkurentnost bez potrebe za zaključavanjem. U radnim okvirima kao što su Akka i ProtoActorGo, aktori su izolovane jedinice koje komuniciraju isključivo slanjem i primanjem poruka, što ih čini idealnim rešenjem za federativni model mašinskog učenja.

Federativno učenje je realizovano kao *P2P* sistem sa n brojem aktora, gde svaki poseduje svoj deo podataka i trenira model lokalno. Aktori razmenjuju težine neuronske mreže putem poruka, koristeći interfejs aktora i aktora za usrednjavanje težina.

Ovde su *CRDT*-ovi korisni za sinhronizaciju modela bez centralizovane koordinacije. Na primer, *CRDT* brojači i mape mogu omogućiti sigurno agregiranje težina modela čak i u uslovima nesinhronizovanih ažuriranja. Ovo rešenje omogućava skalabilnost i otpornost na kvarove, što je ključno za distribuirane sisteme poput mreža baziranih na federativnom učenju.



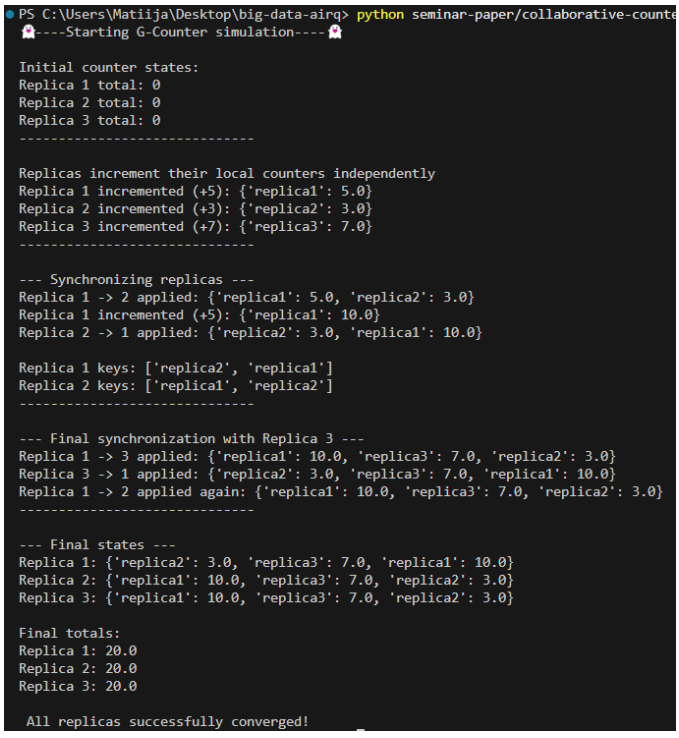
Slika 4: Aktorski sistem (preuzeto sa **https://i0.wp.com/i.postimg.cc/G2rBsFv7/akka-Actor-Model.png?w=1300&ssl=1**)

Pokazni primer

U ovom delu seminarskog rada, ilustrovaćemo principe kroz konkretnu implementaciju brojača tipa *G*. U možda najplastičnijem primeru, *CRDT*-ovi omogućavaju kolaborativne izmene podataka i automatski rešavaju konflikte. Za ovu skriptu korišćena je Pajton biblioteka *pycrdt*, koja se oslanja na efikasnu implementaciju *CRDT*-a u Rastu.

Brojač tipa G (**engl.** Grow-only Counter) je jednostavan CRDT koji podržava samo operaciju uvećavanja broja. Svaka replika vodi sopstvenu lokalnu vrednost, a prilikom sinhronizacije vrednosti se kombinuju tako što se po ključevima uzima maksimalna vrednost, a ukupan rezultat predstavlja zbir svih lokalnih vrednosti.

U sledećem primeru imamo tri korisnika (replike): Matija, Anastasia i Milica, koji nezavisno uvećavaju svoje lokalne brojače. Nakon toga se replike sinhronizuju i postiže se konzistentno stanje bez konflikata.



Slika 5: Brojač tipa *G* i konvergencija stanja

6. Zaključak

Jasno je da *CRDT* predstavlja moćno rešenje za postizanje eventualne konzistentnosti u distribuiranim sistemima, tj. rešenje koje nudi visok stepen dostupnosti i otpornosti na greške. Njihova primena eliminiše potrebu za centralizovanom koordinacijom i složenim mehanizmima zaključavanja, što ih čini pogodnim za aplikacije koje zahtevaju efikasnu replikaciju podataka.

Ipak, njihova implementacija nije univerzalno rešenje i zahteva pažljivo planiranje. Povećana upotreba resursa, kompleksnost dizajna i ograničenja u primeni zahtevaju da se oni koriste samo u situacijama gde njihove prednosti dolaze do izražaja. U situacijama gde je potrebna stroga konzistentnost ili gde konflikti ne mogu lako biti rešeni automatski, tradicionalni pristupi mogu biti bolji izbor.

Uprkos izazovima, *CRDT* je ključna tehnologija za izgradnju modernih distribuiranih sistema. Njegova upotreba u kolaborativnim alatima, distribuiranim bazama podataka i sistemima za razmenu poruka pokazuje da mogu značajno poboljšati pouzdanost i performanse sistema. Kako se tehnologija razvija, očekuje se da će ovaj tip podataka igrati još važniju ulogu u oblikovanju budućnosti distribuiranih aplikacija.

"Tehnologija je moćna sila u našem društvu. Podaci, softver i komunikacija mogu se koristiti na loše načine: za učvršćivanje nepravednih struktura moći, podrivanje ljudskih prava i zaštitu određenih interesa. Ali mogu se koristiti i na dobre načine: da omoguće glas onima koji se ne čuju, da stvore prilike za sve i da spreče katastrofe." [5] Ovaj seminarski rad posvećen je svima koji rade na tome da tehnologija služi dobru.

7. Literatura

[1] “What is a Distributed System?”, https://www.geeksforgeeks.org/what-is-a-distributed-system, poslednji pristup 21.2.2025.

[2] Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski: “A comprehensive study of Convergent and Commutative Replicated Data Types”, https://inria.hal.science/inria-00555588v1/document, poslednji pristup 27.2.2025.

[3] “A Bluffers Guide to CRDTs in Riak“, https://gist.github.com/russelldb/f92f44bdfb619e089a4d, poslednji pristup 10.3.2025.

[4] “A Gentle Introduction to CRDTs”, https://vlcn.io/blog/intro-to-crdts , poslednji pristup 10.3.2025.

[5] “Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems”, Martin Kleppmann, poslednji pristup 20.3.2025.

[6] https://github.com/MatijaMax/big-data-airq, poslednji pristup 17.7.2025.