



FER3

Diplomski studij

Raspodijeljeni sustavi

Druga laboratorijska vježba —
Praćenje senzorskih očitavanja u
vremenu

Ak. god. 2023./2024.

1 Uvod

Cilj ove vježbe je izgraditi decentralizirani raspodijeljeni sustav s ravnopravnim sudionicima (od sad nadalje se koristi riječ **čvor**) koristeći komunikaciju protokolom UDP i kontrolni čvor Kafka¹ kako je definirano u odjeljku 2, te naučiti praktičnu uporabu tehnologije Kafka. Pritom će se primjeniti mehanizam za sinkronizaciju procesa tijekom vremena. Ova se vježba sastoji od tri dijela:

- Proučavanja priloženih primjera koda s predavanja (Kafka i komunikacija protokolom UDP).
- Programiranje čvora koji je dio sustava s ravnopravnim sudionicima.
- Programiranje kontrolnog čvora Kafka.

Studenti trebaju programski izvesti čvor opisanog raspodijeljenog sustava i kontrolni čvor Kafka, a prilikom demonstracije rješenja će pokrenuti **više instanci** čvora (**minimalno 3 procesa**, ali mora biti moguće pokrenuti i više procesa).

1.1 Predaja

Studenti su dužni putem sustava Moodle predati arhivu koja se sastoji od:

- izvornog kôda čvora sustava s ravnopravnim sudionicima,
- izvornog kôda upravljačkog čvora Kafka.

Arhiva se mora predati do roka koji će biti objavljen na stranici predmeta. Uz predaju digitalne arhive, organizirat će se usmena prezentacija rješenja, gdje će studenti morati objasniti implementaciju i pokazati svoje razumijevanje primijenjenih komunikacijskih načela i tehnologija korištenih za implementaciju rješenja. Termin usmene prezentacije bit će objavljen na stranici predmeta.

Sve komponente sustava mogu se implementirati u bilo kojem programskom jeziku. Imajte na umu da su primjeri prikazani na predavanjima implementirani u programskom jeziku Java. Arhivu koja sadrži izvorni kôd treba nazvati **ime_prezime_jmbag**. Arhiva treba sadržavati 1 direktorij koji sadrži datoteke izvornog kôda. Dopušteno je slanje zip arhive koja sadrži izvoz projekta ako je implementiran u IDE-u (Eclipse, NetBeans itd.).

Studenti koji predaju vježbu nakon navedenog roka dobit će 0 bodova iz vježbe. Konzultacije se održavaju **utorkom od 10:00 do 11:00**, uz prethodnu **najavu** na MS Teamsu (kanal Laboratorijske vježbe) ili e-mailom.

2 Arhitektura raspodijeljenog sustava

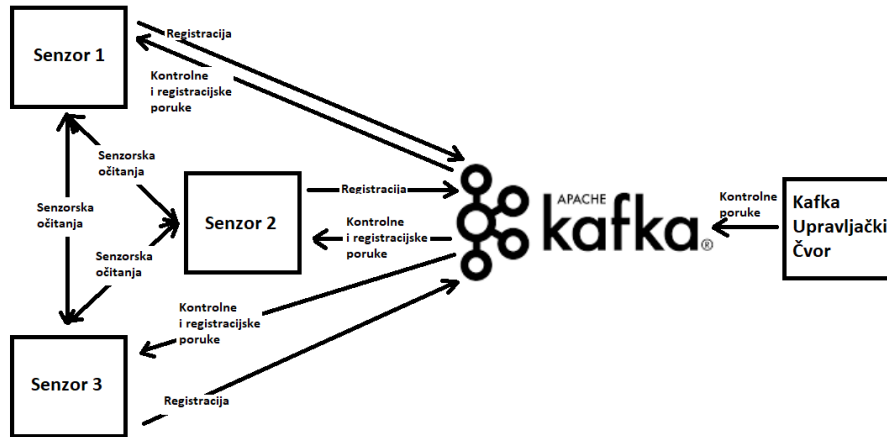
Kratak pregled funkcionalnosti sustava: Raspodijeljeni sustav koristi se za praćenje senzorskih očitavanja u vremenu uz pomoć skupine ravnopravnih čvorova, gdje svaki čvor **"glumi"** senzor. Čvorovima se upravlja pomoću kontrolnih poruka koje šalje upravljački čvor Kafka (odnosno **koordinator**). Pregled arhitekture raspodijeljenog sustava prikazan je na slici 1. Čvorovi se povezuju direktno sa svim ostalim čvorovima u mreži i čine potpuno povezanu mrežu.

Zadaća koordinatora i poslužitelja Kafka je koordinirati formiranje potpuno povezane mreže čvorova, tj. omogućiti **razmjenu identifikatora** između čvorova (kako bi se čvorovi mogli povezati) te **pokretanje i zaustavljanje** rada čvorova nakon što je mrežna topologija uspostavljena. Funkcionalnost koordinatora i komunikacija koordinatora i čvorova detaljno su objašnjeni u poglavlju 4.

Zadaća čvorova je razmjena senzorskih očitavanja sa svim susjednim čvorovima, te **sortiranje primljenih i vlastitih očitavanja** u vremenu koristeći **skalarne i vektorske** oznake vremena

¹<https://kafka.apache.org/>

te **izračunavanje srednje vrijednosti** očitavanja u vremenskom prozoru od **5 sekundi**. Senzori međusobno komuniciraju protokolom **UDP**, odnosno svaki senzor je **UDP klijent** i **UDP poslužitelj**. Dodatno, svaka UDP konekcija simulira gubitak UDP paketa u mreži. Svaki izgubljeni UDP paket potrebno je ponovno poslati, odnosno potrebno je implementirati **retransmisiju**. Funkcionalnost čvorova i komunikacija između čvorova detaljno su objašnjeni u poglavlju 5.



Slika 1: Arhitektura raspodijeljenog sustava

Napomena: Čvorove je potrebno realizirati u zasebnim procesima kako bi se omogućilo njihovo pokretanje na 2 ili više računala, tj. **NIJE dozvoljeno** pokretanje više čvorova **unutar jednog procesa**.

3 Instalacijske upute

U nastavku se opisuje instalacija Kafke, Zookepera (alata za praćenje statusa čvorova spojenih na Kafku te za održavanje tema i poruka) te postavljanje razvojne okoline za operacijski sustav Windows. Preduvjet za pokretanje poslužitelja Zookeeper i Kafka je **Java 8** SDK ili novija verzija. Najnovije verzije Zookepera i Kafke se mogu preuzeti na **poveznici**. Arhivu ekstrahirajte u proizvoljni direktorij (u daljnjem tekstu **KAFKA_HOME**). Primjer vrijednosti varijable **KAFKA_HOME** je **C:\kafka**.

Napomena: Preporučljivo je koristiti verziju Kafke 3.6.1 Scala 2.13 za Windows operacijski sustav.

3.1 Kafka

Kafka se konfigurira korištenjem konfiguracijske datoteke **server.properties** koja se nalazi u direktoriju **KAFKA_HOME\config** (npr. **C:\kafka\config\server.properties**). Kafka pohranjuje sadržaj na lokaciju koja je definirana u datoteci **server.properties** pod vrijednošću **log.dirs**. Vrijednost **log.dirs** postavite proizvoljno, npr. **KAFKA_HOME\data\kafka**. Sve ostale postavke mogu ostati na pretpostavljenim vrijednostima.

3.2 Zookeeper

Zookeeper se konfigurira korištenjem konfiguracijske datoteke **zookeeper.properties** koja se nalazi u direktoriju **KAFKA_HOME\config** (npr. **C:\kafka\config\zookeeper.properties**). Zookeeper pohranjuje sadržaj na lokaciju koja je definirana u datoteci **zookeeper.properties** pod vrijednošću **dataDir**. Vrijednost **dataDir** postavite proizvoljno, npr. **KAFKA_HOME\data\zookeeper**. Sve ostale postavke mogu ostati na pretpostavljenim vrijednostima.

3.3 Pokretanje poslužitelja Zookeeper i Kafke

Zookeeper i Kafka pokreću se korištenjem terminala. Prvo je potrebno pokrenuti Zookeeper, a nakon njega Kafku. Skripte za pokretanje nalaze se u direktoriju **KAFKA_HOME\bin\windows**. Zookeeper se pokreće korištenjem skripte **zookeeper-server-start** koja kao argument prima putanju do konfiguracijske datoteke (npr. **C:\kafka\bin\windows\zookeeper-server-start.bat C:\kafka\config\zookeeper.properties**). Kafka se pokreće korištenjem skripte **kafka-server-start** koja kao argument prima putanju do konfiguracijske datoteke (npr. **C:\kafka\bin\windows\kafka-server-start.bat C:\kafka\config\server.properties**).

4 Funkcionalnost koordinatora

Ključne funkcije upravljačkog čvora Kafka jesu slanje kontrolnih poruka za razmjenu identifikatora čvorova, pokretanje i zaustavljanje grupe čvorova. Radi jednostavnosti, možete pretpostaviti da prije slanja prve poruke od strane koordinatora svi čvorovi su pretplaćeni na dvije teme koje održava Kafka, **"Command"** i **"Register"**, te da nakon pokretanja čvorova u sustav neće biti dodani novi čvorovi, tj. topologija mreže čvorova se ne mijenja nakon početnog postavljanja.

Napomena: Kafka poslužitelj i koordinatorski proces su dva različita procesa. Koordinator je objavitelj poruka koji upravlja kreiranjem mrežne topologije, dok Kafka poslužitelj prima te poruke i prosljeđuje ih dalje.

4.1 Pokretanje grupe čvorova

Koordinator pokreće grupu čvorova slanjem kontrolne poruke **"Start"** koja se šalje na temu **"Command"**. Funkcionalnost čvorova prije i nakon primanja poruke **"Start"** objašnjena je u poglavljima 5.1 i 5.2.

4.2 Zaustavljanje grupe čvorova

Koordinator zaustavlja grupu čvorova slanjem kontrolne poruke **"Stop"** koja se šalje na temu **"Command"**. Nakon slanja poruke **"Stop"**, koordinatorski proces mora ugasiti. Funkcionalnost čvorova nakon primanja poruke **"Stop"** objašnjena je u poglavlju 5.8.

5 Funkcionalnost čvorova

Ključne funkcionalnosti čvora uključuju stvaranje komunikacije sa svim susjednim čvorovima korištenjem protokola UDP, generiranje vlastitih očitavanja, razmjenu vlastitih očitavanja sa drugim čvorovima putem UDP veze, mogućnost sortiranja primljenih i vlastitih očitavanja koristeći skalarne i vektorske oznake vremena i izračunavanje srednje vrijednosti očitavanja u vremenskom prozoru od 5 sekundi. Prilikom implementacije čvora možete koristiti priloženi kôd dostupan na stranicama predmeta kao primjer za razvoj UDP klijenta i UDP poslužitelja. Nakon što se formira povezana mreža čvorova, postupak generiranja očitavanja, dohvaćanja očitavanja od drugih čvorova, sortiranje i izračunavanje srednje vrijednosti se neprestano ponavlja sve dok koordinatorski proces ne pošalje kontrolnu poruku za zaustavljanje grupe čvorova.

5.1 Inicijalizacija čvora

Nakon pokretanja, čvor traži upis svog **identifikatora** (odnosno **id**) i **UDP port** putem kojeg komunicira s ostalim čvorovima od korisnika. Nakon toga pretplaćuje se na teme **"Register"** i **"Command"** i čeka kontrolnu poruku **"Start"** od koordinatora. Radi jednostavnosti, pretpostavite da

koordinator šalje poruku **"Start"** samo jedanput, te da nakon slanja poruke **"Start"** novi senzori neće biti dodani u sustav.

5.2 Registracija čvora

Nakon što čvor primi kontrolnu poruku **"Start"**, šalje registracijsku poruku na temu **"Register"** u JSON formatu. Registracijska poruka sastoji se od **identifikatora** čvora, njegove **IP adrese** i **porta**. Primjer registracijske poruke prikazuje JSON 1.

JSON 1: Primjer registracijske poruke u JSON formatu

```
1 {  
2   "id": "1",  
3   "address": "localhost",  
4   "port": "8000"  
5 }
```

5.3 Uspostavljanje UDP komunikacije

Nakon što čvor primi registracijske poruke koje sadrže identifikatore svih ostalih čvorova putem teme **"Register"**, čvor započinje komunikaciju sa svim ostalim čvorovima korištenjem protokola UDP.

Za simuliranje gubitka paketa u mreži potrebno je koristiti klasu **SimpleSimulatedDatagramSocket** koju možete pronaći u arhivi na stranicama predmeta. Ova klasa koristi se na način identičan klasi **DatagramSocket**. Razlika između ove dvije klase je u tome što klasa **SimpleSimulatedDatagramSocket** ima drugačiji konstruktor: **SimpleSimulatedDatagramSocket(double lossRate, int averageDelay)**.

Ovaj konstruktor prima 2 parametra: postotak izgubljenih paketa u mreži te prosječno kašnjenje paketa u jednome smjeru izraženo u milisekundama. Postotak izgubljenih paketa u mreži postavite na 30%, Parametar kašnjenja paketa postavite na $1000ms$.

Napomena: S obzirom na to da je sve komponente sustava dozvoljeno implementirati u bilo kojem programskom jeziku, funkcionalnost klase **SimpleSimulatedDatagramSocket** potrebno je reimplementirati u odabranom jeziku ako se ne koristi Java.

5.4 Generiranje senzorskih očitavanja

Budući da pravi senzori nisu spojeni na sustav, njihova se funkcija emulira pripremljenim očitanjima iz ulazne datoteke **readings.csv**, gdje očitavanja (retci) imaju više vrijednosti (stupaca). Za generiranje očitavanja iz ulazne datoteke upotrijebite jednadžbu 1 za dobivanje broja retka koji ćete koristiti za očitavanje koncentracije plina **NO2** iz ulazne datoteke:

$$red \leftarrow (brojAktivnihSekundi \% 100) + 1 \quad (1)$$

Varijabla *brojAktivnihSekundi* je vrijeme rada senzora (vrijeme se mjeri od trenutka pokretanja klijenta i izražava se u sekundama). Potrebno je očitati samo **NO2** očitavanja iz datoteke.

5.5 Razmjena očitavanja i ažuriranje vremenskih oznaka

Nakon što čvor generira vlastito očitavanje, stvara UDP paket koji sadrži generirano očitavanje zajedno s **ažuriranom vektorskom i ažuriranom skalarnom vremenskom oznakom** (u daljnjem tekstu podatkovni paket), te ga šalje preko UDP-a svim ostalim čvorovima u mreži.

Za emuliranje različitih fizičkih satova na istom računalu potrebno je koristiti priloženu klasu **EmulatedSystemClock**. Instancu klase **currentTimeMillis()** kreirajte prilikom pokretanja čvora te

ju koristite za dohvaćanje skalarne oznake vremena. Za **ažuriranje skalarne vremenske oznake**, dozvoljeno je modificirati klasu **EmulatedSystemClock**.

Napomena: S obzirom na to da je sve komponente sustava dozvoljeno implementirati u bilo kojem programskom jeziku, funkcionalnost klase **EmulatedSystemClock** potrebno je reimplementirati u odabranom jeziku ako se ne koristi Java.

Za svaki uspješno primljeni podatkovni paket, čvor X šalje potvrdu čvoru Y, također u obliku paketa (u daljnjem tekstu potvrda). Čvor Y čeka potvrde za sve pakete koje šalje.

Napomena: Potvrda mora biti jednostavnijeg formata od podatkovnog paketa. Svaki čvor mora koristiti **ista vrata** (eng. *port*) za primanje potvrda i podatkovnih paketa, tj. logika razlučivanja radi li se o potvrdi ili podatkovnom paketu mora biti implementirana **na aplikacijskoj razini** (npr. **nije dozvoljeno** koristiti jedna vrata za podatkovne pakete a druga za potvrde).

5.6 Retransmisija izgubljenih paketa

Ako čvor ne primi potvrdu za poslani podatkovni paket, ponovno šalje taj podatkovni paket. Prilikom ponovnog prijenosa podatkovnog paketa, izvorne vremenske oznake moraju biti sačuvane. Također je moguće da se neki podatkovni paketi šalju i primaju više puta. Podatkovne pakete primljene više puta treba zanemariti, ali se za svaki od tih paketa mora poslati potvrda.

5.7 Sortiranje i računanje srednje vrijednosti očitavanja

Nakon 5 sekundi svaki čvor sortira sva očitavanja (vlastita i primljena od drugih čvorova u mreži) u zadnjih 5 sekundi koristeći vektorske i skalarne oznake vremena te se **na konzoli ispisuju oznake, zajedno sa sortiranim očitanjima**. Također računa i ispisuje srednju vrijednost primljenih očitavanja tijekom prozora od 5 sekundi. Sortiranje je potrebno provesti koristeći obje oznake vremena te pojedinačno ispisati svaki sortirani skup. Sortiranje se može obaviti silazno ili uzlazno.

5.8 Zaustavljanje rada čvorova

Nakon što čvorovi prime kontrolnu poruku "**Stop**", prvo se zaustavlja UDP komunikacija između čvorova, te se nakon toga sortiraju i ispišu svi očitani i primljeni podaci. Nakon ispisa se čvor zaustavlja.

6 Često postavljena pitanja

6.1 Smijem li koristiti jezik i/ili razvojno okruženje po svom odabru?

Smijete, uz opasku da će za 3. laboratorijsku vježbu biti obavezno koristi Spring Boot odnosno Java.

6.2 Koji jezik i/ili razvojno okruženje nam preporučujete?

Preporučujemo korištenje:

- Java 19
- Gradle 8.5

6.3 Koje knjižice nam preporučujete za rješavanje labosa?

Preporučujemo ove knjižice:

- Kafka: <https://mvnrepository.com/artifact/org.apache.kafka/kafka>
- Csv: <https://mvnrepository.com/artifact/org.apache.commons/commons-csv>
- Json: <https://mvnrepository.com/artifact/org.json/json>

6.4 Što to znači da potvrda mora biti jednostavnijeg formata?

Potvrda mora sadržavati manje podataka nego podatkovni paket. Npr. ako podatkovni paket sadrži sva slova *lowercase*, poruka potvrde ne smije biti *uppercase*. Kako bi se izbjeglo slanje takvih poruka, preporučujemo korištenje jedinstvenog identifikatora.

6.5 Pitanja studenata prijašnjih godina

Q1: Kada se čvorovi registriraju, šalju JSON sa podacima id, address i port. Smijemo li taj JSON prilikom publish-a na register topic pretvoriti u String?

A1: Možete pretvoriti taj JSON u string i onda ga kasnije parsirati kao JSON kada drugi čvorovi prime tu poruku. S tim da poruka mora biti u JSON formatu, bez obzira šaljete li ju kao string.

Q2: Trebamo li imati dva projekta - jedan za čvor i jedan za koordinator ili je u redu da sve bude u istom projektu?

A2: Možete to sve u 1 projekt, međutim senzori i koordinator se moraju moći odvojeno pokrenuti (ne koristeći dretve)!

Q3: Koordinator će biti *producer* za topic "Command" i samo će on objavlјivati naredbe "Start" i "Stop" na taj topic?

A3: Tako je, da!

Q4: Kako funkcionira *topic* "Register"? Znači li to da će svaki senzor biti *producer* za *topic* "Register" i na njega objaviti svoje podatke?

A4: Senzori će biti i Producer i Consumer, tj. uloga Producera za senzore jest slanje registracijske poruke na temu "Register", a uloga Consumer za senzore jest primanje kontrolnih poruka na temu "Command" i primanje registracijskih poruka na temu "Register"

Q5: Na što se točno odnosi ova rečenica u uputama: "*Ključne funkcije upravljačkog čvora Kafka jesu slanje kontrolnih poruka za razmjenu identifikatora čvorova*"

A5: Nakon kada upravljački čvor Kafka pošalje kontrolnu poruku "Start", senzori primaju tu poruku preko teme "Command", te nakon toga senzori šalju svoju registracijsku poruku na temu "Register".