SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5879

OBRADA TOKA SENZORSKIH PODATAKA KORIŠTENJEM PLATFORME APACHE FLINK

Sara Zorčić

Zagreb, lipanj 2018.

Na drugu stranicu stavite izvornik zadatka završnog rada.

Sadržaj

Tablica sadržaja

Uvod....1
1.Obrada toka senzorskih podataka....2
1.1.Latencija i propusnost....2
1.2.Batch i Stream Processing....3
1.3.Operacije na tokovima podataka....4
1.4. Upotreba platformi za obradu podataka....4
2.Platforma pache Flink....6
2.1.Arhitektura platforme Flink....6
2.2 Visoka dostupnost....10
2.3. Tipovi podataka u Flinku....11
2.3.1. Tuples....11
2.3.2. POJO....12
2.3.3. Value tipovi....12
2.4. Prozori....12
2.5. Upravljanje vremenom....14
3.Obrada toka senzorskih podataka korištenjem platforme Apache Flink....18
4. Zaključak....22
5. Literatura....23
6. Sažetak....24

Uvod

U tradicionalnim sustavima podatci se prvo pohranjuju i onda obrađuju. Za analizu podataka koji se konstantno stvaraju i mijenjaju to je problem. Njihova obrada zahtjeva tehnologiju za koja bi kontinuirano proizvodila rezultate dok podatci još uvijek pristižu. Interes za učenje kako analizirati podatke strujanja u velikim sustavima poput web prometa, financijskih transakcija, dnevnika strojeva, industrijskih senzora i mnogih drugih, je sve veći. Platforma Apache Flink nudi rješenja za obradu kontinuiranih tokova podataka. U ovom završnom radu dati ću osvrt na platformu Apache Flink s naglaskom na obradu podataka u stvarnom vremenu.

1. Obrada toka senzorskih podataka

Za početak, tok podataka se definira kao *potencijalno neograničen niz događaja.* Tok ima ulogu cjevovoda, nije spremište podataka već stvaranjem toka "puštamo" podatke u cjevovod koji smo prethodno složili. Događaji mogu predstavljati podatke o praćenju, mjerenja senzora, transakcije kreditne kartice, opažanje meteorološke postaje, korisnikove online interakcije, web pretrage, itd. U većini ovih slučajeva mi ne znamo koliko dugo ćemo primati podatke, pa zbog toga pričamo o beskonačnim tokovima podataka (engl. *Infinite streams*).

1.1. Latencija i propusnost

Streaming aplikacije moraju što je brže moguće pružiti rezultate za ulazne podatke i istovremeno se znati nositi s visokom učestalosti pristizanja tih podataka. Latencija je vremenski interval od trenutka kad smo primili događaj do trenutka kad smo dobili rezultat njegove obrade, tj. koliko je vremena potrebno za obradu događaja. Kod obrade podataka latencija se mjeri u vremenskim jedinicama. Niska latencija je ključna karakteristika obrade tokova podataka i ona je jedna od glavnih karakteristika koja nam omogućuje obradu podataka u stvarnom vremenu. Apache Flink nudi latenciju svega par milisekundi. To je nešto najbitnije streaming aplikacijama za npr. otkrivanje prijevara, pokretanje alarma, praćenje mreža, itd. Jer za njih to znači da se problemi mogu predvidjeti, dok aplikacije s latencijom iznad stotine milisekundi često samo detektiraju događaje. Latencija i propusnost, naravno, nisu nezavisni mjerni podaci. Ako podacima treba dugo da stignu na obradu, ne možemo osigurati visoku propusnost, a ako je kapacitet sustava mali, podaci će biti u *buffer*-u i čekati na obradu. Propusnost je mjera kapaciteta obrade sustava, tj. njezina brzina obrade. Propusnost nam govori koliko događaja sustav može obraditi u vremenskoj jedinici. Želimo da latencija bude što niža, a protok da bude što je moguće veći. Propusnost se mjeri u događajima ili operacijama po jedinici vremena. Važno je napomenuti da brzina obrade ovisi o stopi prispijeća podataka. Niska propusnost ne mora nužno ukazivati na lošu izvedbu. Kod *streaming* sustava, cilj nam je da mogu podnijeti događaje koji dolaze maksimalnom očekivanom brzinom. Zato smo usredotočeni na

određivanje vršne propusnosti (*peak throughput*). To je propusnost kad je sustav u maksimalnom opterećenju. Ako zamislimo da na početku raspolažemo svim resursima sustava, kad dođe prvi događaj kojeg treba obraditi, to će se izvesti s minimalnom latencijom. No, u trenutku kad događaji budu dolazili većom frekvencijom, tolikom da se svi resursi sustava koriste, morat ćemo početi spremati negdje te događaje. U ovom trenutku sustav je postigao vrhunac propusnosti i ako se frekvencija pristizanja događaja poveća toliko da spremnici ne mogu primiti više podataka, oni će se izgubiti. Ova situacija se naziva *backpressure* i Apache Flink ima mehanizam koji se bavi time.

1.2. Batch i Stream Processing

Postoje dva osnovna tipa obrade podataka. Jedan je obrada u skupinama. Naziva se i odgođena ili off-line obrada. To je obrada blokova podataka koji su pohranjeni tijekom nekog vremenskog razdoblja. Ona podrazumijeva obradu kod koje se sve što je u vezi sa jednim kompletnim poslom sakuplja i prenosi radi obrade (lokalno, u istoj zgradi, ili na većim udaljenostima, u nekom drugom dijelu zemlje), pomoću određene programske aplikacije. Tipičan primjer obrade u skupinama podataka je obrada transakcija koje je neka velika financijska firma obavila taj tjedan. Takva obrada ne dopušta interakciju između korisnika i programa kada se program učita u računalo. Ako je riječ o obradi u skupinama u telekomunikacijama (o slanju nekog zadatka koji će se završiti preko telefonske linije), vrijeme odziva mreže nije kritično, jer nitko ne čeka ispred ekrana da dobije odziv. S druge strane, točnost komunikacija je veoma kritična, jer nitko ne provjerava unos i odzive. Zato batch processing koristimo kad nam nije nužno dobiti neke analitičke rezultate istog trena, već obraditi ogromnu količinu podataka. Hadoop MapReduce je najbolji okvir za obradu podataka u skupinama, a Flink izvodi batch programe kao posebne slučajeve streaming programa. Drugi tip obrade podataka naziva se interaktivna obrada ili obrada u realnom vremenu. U ovom slučaju vrijeme odziva komunikacijske opreme je kritično. Broje se sekunde, posebno ako korisnik na drugoj strani čeka, na primjer, informacije o rasporedu letova neke avionske tvrtke. Razlog zašto je stream processing tako brz je taj što se podaci analiziraju prije nego se zapišu na disk.

1.3. Operacije na tokovima podataka

Okviri za obradu toka podataka obično nude skup ugrađenih operatora i izlaznih tokova. Slijed operatora implementira logiku kojom se izvodi apikacija. Postoje dvije vrste operatora. Oni koji ne prate nikakva stanja - *stateless* i oni koji prate stanja - *stateful*. *Stateless* operatori ne ovise o prošlim događajima i ne zapisuje se povijest stanja. Kako se događaji u tom slučaju mogu obraditi neovisno jedni o drugima i neovisni o svojim vremenima pristizanja, ovakve operacije se lako izvode paralelno. Štoviše, u slučaju neuspjeha, *stateless* operator se jednostavno može ponovo pokrenuti i nastaviti obrađivanje s mjesta gdje je stao. Suprotno njima, *stateful* operatori mogu sadržavati informacije o događajima koji su pristigli nekad u prošlosti, tj. pamte prijašnja stanja sustava. Stanje se ažurira dolaskom novih podataka u tok i može se iskoristiti za logiku obrade budućih događaja. *Stateful* operacije je teže izvoditi paralelno i bez greške.

1.4. Upotreba platformi za obradu podataka

Neki od primjera upotrebe Apache Flink-a je rad s mjerenjima senzora ili financijskim transakcijama. No, postoji mnogo više rasprostranjenih izvora podataka za strujanje, npr.*clickstream* podaci koji odražavaju ponašanje korisnika na web stranicama ili zapisi strojeva za vlastitni podatkovni centar. Ovo su neka od područja gdje je *stream proccesing* uobičajen:

– The Internet of Things:

Internet stvari (IoT) je područje u kojem je streaming podataka uobičajen i gdje je dostava i obrada podataka s malom latencijom, skupa s točnosti analize podataka, često kritična. Senzori u različitim vrstama opreme rade učestala mjerenja i prenose ih u podatkovne centre u kojima će aplikacije za obradu u stvarnom vremenu ili skoro stvarnom vremenu ažurirati nadzorne ploče, pokrenuti modele strojnog učenja, izdavati upozorenja i dati povratne informacije za mnoge različite usluge.

– Prometna industrija:

Prijevozna industrija je još jedan primjer gdje je važno strujanje izvoditi točno. Na primjer, najsuvremeniji sustavi vlakova se oslanjaju na podatke senzora koji se prenose s tračnica na vlakove i od vlakova do senzora na rutama. Zajedno, izvješća se također prenose natrag u kontrolne centre. Mjerenja uključuju brzinu i lokaciju vlaka, kao i informacije iz okoline za uvjete na stazi. Ako se ti podaci strujanja ne obrađuju ispravno, prilagodbe i upozorenja se ne događaju dovoljno brzo da se može prilagoditi na opasna stanja i tako izbjeći nesreće. Drugi primjer iz transportne industrije su "pametni" ili povezani automobili, koji su dizajnirani za komuniciranje podataka putem mobilnih mreža, natrag proizvođačima. U nekim zemljama (npr. Francuskoj, Velikoj Britaniji i početku u SAD-u) povezani automobili čak pružaju informacije osiguravajućim društvima i, u slučaju utrka automobila, šalju podatke natrag u auto pitove putem radio frekvencije za analizu. Neke aplikacije za pametne telefone također nude aktualna ažuriranja prometa koju dijele milijuni vozača.

– Telekom:

Telekomunikacijska industrija poseban je slučaj IoT podataka, sa svojom širokom primjenom podataka o strujanju događaja u razne svrhe kroz zemljopisne regije. Ako telekomunikacijska tvrtka nemože dobro izvesti obradu tokova podataka, neće npr. uspjeti preventivno preusmjeriti rute koje se koriste na druge stanične tornjeve ili odgovoriti brzo ispade/prekide rada.

– Financijski sektor:

Evo još jednog područja široke upotrebe stream proccesinga. Što se događa ako financijska institucija nema aplikacije koje mogu prepoznati anomalno ponašanje u podacima o aktivnostima korisnika s osjetljivom detekcijom u stvarnom vremenu? Otkrivanje prijevara kod transakcija kreditnim karticama zahtijeva pravodobno praćenje i odgovor. Biti u stanju otkriti neobične obrasce prijave koji signaliziraju napad na internetsku mrežu može dobiti ogromne uštede otkrivajući probleme na vrijeme kako bi ublažio gubitak.

2. Platforma pache Flink

Doba *Big Data* tehnologije počelo je s višestrukim programskim radnim okvirima od kojih su se svi fokusirali na neki specifičan problem. Neki od njih su:

– Hadoop - napravljen za batch obradu podataka

– MapReduce - programski model za obradu velikih količina podataka

– mongoDB - NoSQL baza podataka, čuva podatke kao JSON dokumente

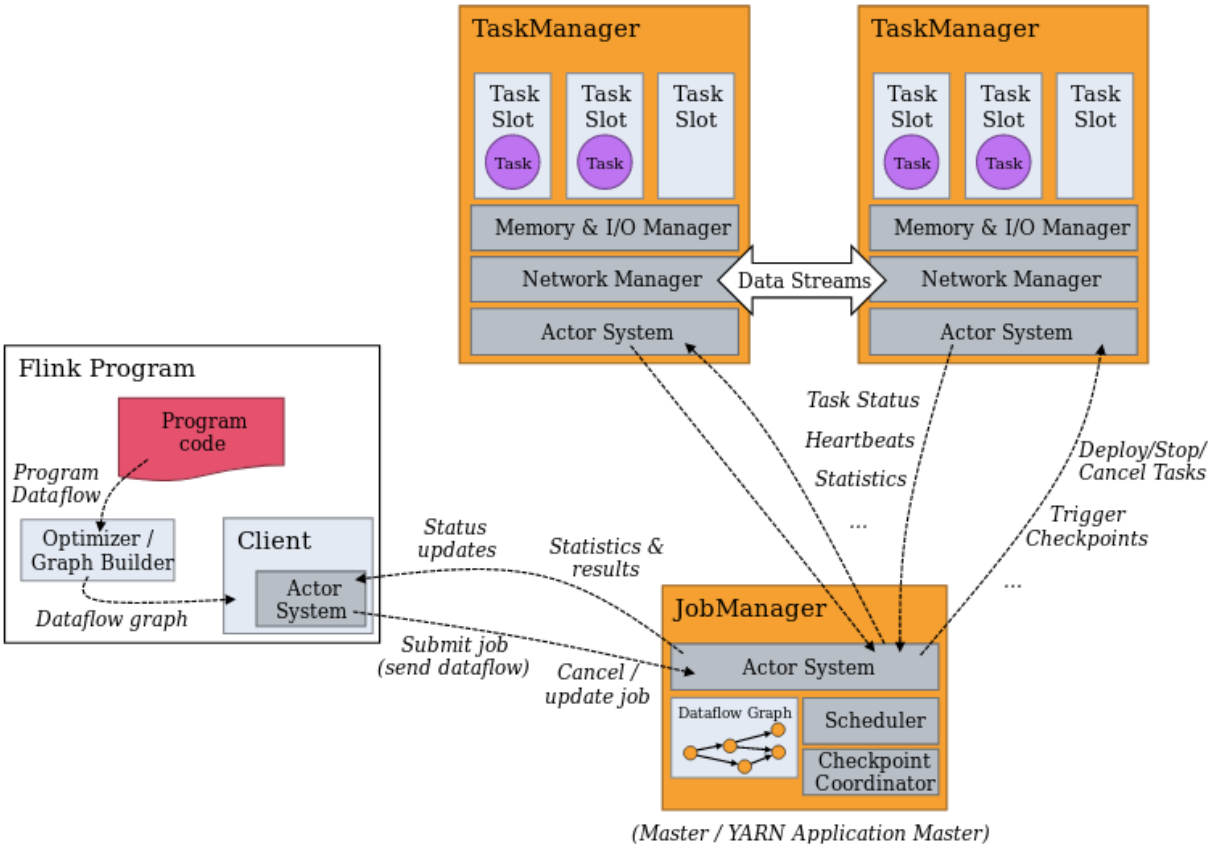
– Greenplum, Apache Storm, Teradata, itd.

Svi od ovih radnih okvira su ograničeni na određeni problem, a u industriji se susrećemo s različitim zahtjevima šarolikog karaktera pa da bi riješili stvarne probleme i to u stvarnom vremenu trebamo kombinirati ove radne okvire zajedno. Zapravo ih moramo integrirati. Što je točno problem kod višestrukih programskih okvira? Njihovom integracijom dobivamo bolju funkcionalnost, ali uskladiti njihov rad na istoj platformi nije trivijalno i jako je skupo. Stoga je rješenje jedinstvena platforma, to jest generalizirana platforma koja bi sama mogla ispuniti sve zahtjeve koje ima industrija poput *bacth* obrade, interaktivne obrade, obrade u stvarnom vremenu, iterativne obrade, itd. *Big data* industrija očekuje sva ta rješenja. Osim tih raznolikih zahtjeva također je očekivano da zadovoljava karakteristike poput distribuiranog računanja, tolerancije kvarova (kvarovi se trebaju automatski obrađivati i neuspjeh ne bi trebao utjecati na platformu), visoke dostupnosti, unaprijed definiranih operatora,itd. Platforma bi trebala biti jednostavna za korištenje i treba obrađivati podatke maksimalnom brzinom. Sve ove zahtjeve zadovoljava Apache Flink.

2.1. Arhitektura platforme Flink

Flink je distribuirani sustav za paralelnu obradu podatkovnih tokova. Sastoji se od više procesa koji se izvode na više računala. Uobičajeni izazovi kojima se distribuirani sustavi bave su dodjela i upravljanje računalnim resursima u grozdu, koordinacija procesa, izdržljiva i dostupna pohrana podataka te oporavak od neuspjeha. Flink ima arhitekturu *master/slave*. *Master* je upravljački čvor koji je središnji dio grozda, a *slaves* su radnički čvorovi. Recimo da klijent radi neki posao.

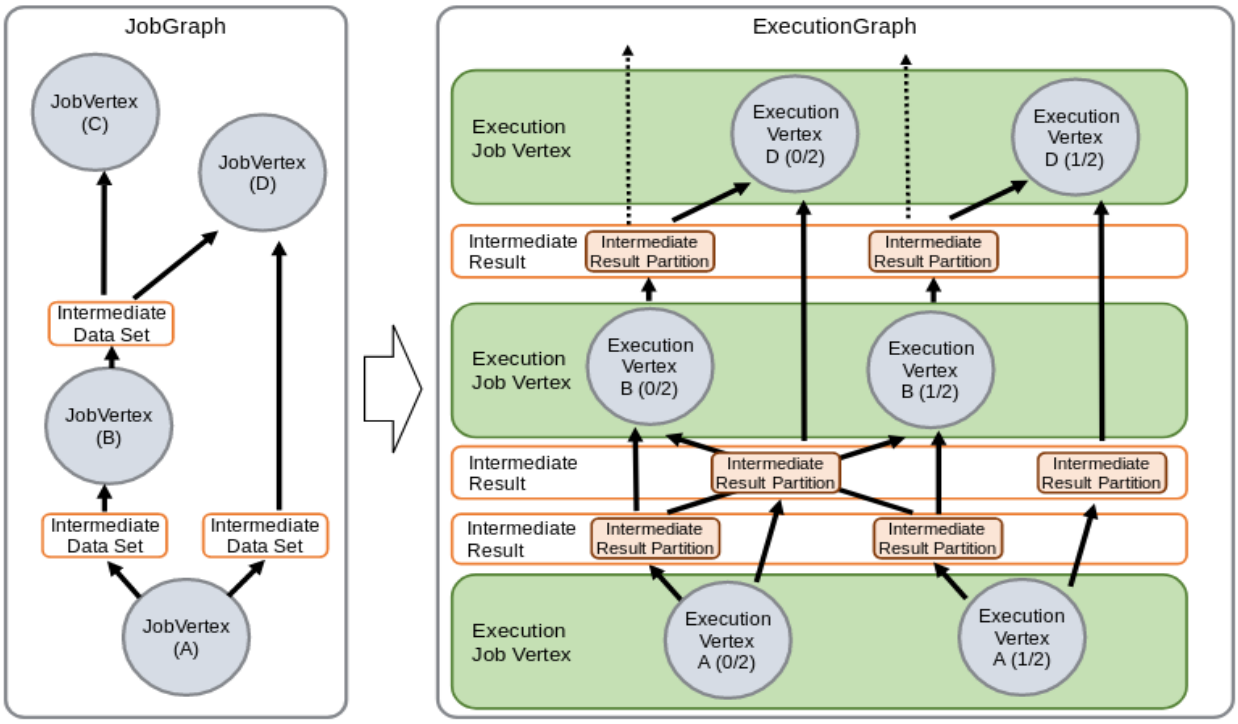
Taj isti posao predaje upravljačkom čvoru. Sada *master* koji upravlja resursima distribuira posao svim radničkim čvorovima. Radni čvorovi obavljaju zadatke i završavaju sav posao koji im je *master* dodijelio.



Slika 2.1 : Model procesa [1]

Flinkov model izvršavanja počinje s napisanim programskim kodom za aplikaciju. Prije nego se program pošalje na izvršenje, on se parsira i optimizira. Takav optimizirani kod se pretvara u logički *dataflow* graf koji se naziva *JobGraph* i predstavlja logički plan izvođenja programa. On predstavlja prikaz podataka koji se sastoji od operatora (*Job-Vertex*) i srednjih rezultata (*IntermediateDataSet*). Svaki operator ima svojstva, poput paralelizma i koda koji izvršava. Osim toga, *JobGraph* ima skup priloženih biblioteka, koji su neophodni za izvršavanje koda operatora. Sad se aplikacija koja se sastoji od *JobGraph*-a i JAR datoteke sa svim potrebnim klasama, bibliotekama i ostalim resursima šalje upravljačkom čvoru. Na njemu se nalazi ***Job Manager*** proces/dretva. On je takozvani glavni koordinator Flink sustava koji kontrolira izvršenje pojedinačne aplikacije, upravlja raspodjelom podatkovnih

blokova i čuva metapodatke sustava (konfiguracijski parametri, "zdravlje" radničkih čvorova, itd.). *Job Manager* pretvara *Job-Graph* u fizički *dataflow* graf koji se još zove i *ExecutionGraph*, a predstavlja fizički plan izvođenja programa. On se sastoji od tih istih zadataka podijeljenih na podzadatke (npr. transformacija) koji se mogu izvršavati paralelno. Za svaki *JobVertex*, on sadrži *ExecutionVertex* za svaki paralelni podzadatak. *ExecutionGraph* također sadrži status posla koji pokazuje trenutno stanje izvršenja, *IntermediateResult* i *IntermediateResultPartition* koji prati stanje srednjih rezultata. Na slici 3.2 vidimo pertvorbu *JobGraph-a* u *ExecutionGraph*.



Slika 2.2 : *JobGraph i ExecutionGraph* [3]

Sljedeći posao *JobManager*-a je osigurati potrebne resurse za izvođenje zadataka. Flinkova jedinica resursa za obradu je *Task Slot*. Poput upravljačkog čvora, radnički čvor također ima svoj proces/dretvu ***Task Manager***. On je odgovoran za posluživanje podatkovnih blokova i slanja izvještaja upravljačkom čvoru. Svaki *Task Slot* predstavlja fiksni podskup resursa *TaskManagera*, tako da mora postojati barem jedan slot. *TaskManager* sa 4 slota će svakom od njih dati 1/4 svoje upravljačke memorije. Takva podjela resursa znači da su podzadaci memorijski neovisni jedni o drugima i mogu se izvoditi paralelno. *JobManager* zatraži slotove i kad ih dobije

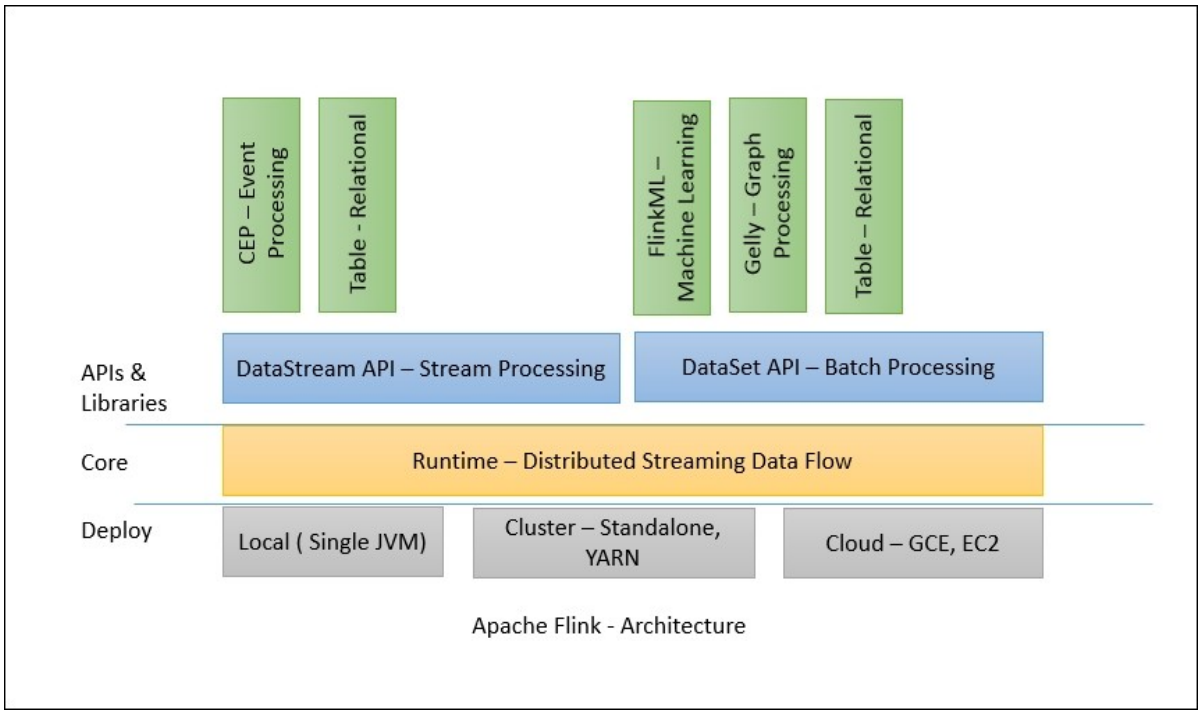
dovoljno, raspoređuje zadatke *TaskManager* ima. Kako je Flink distribuirani sustav, *JobManager* i *TaskManageri* se obično odvijaju kao odvojeni JVM procesi na različitim računalima. main() metoda programa okuplja tok podataka i šalje ga na *JobManager* pozivom metode `StreamExecutionEnvironment.execute()`. Ipak, postoji local mode način u kojem pozivom metode `ecexute()` pokrećemo *JobManager* i *TaskManager* kao odvojene dretve unutar istog JVM. U tom slučaju *JobManager* dobiva onoliko slobodnih mjesta(slots) koliko ima dostupnih CPU dretvi. Na taj način pokrećemo Flink programe u običnom IDE-u.

Na slici 3.3 vidimo od kojih se sve slojeva komponenti sastoji Flink.

– Flink Core - osnovne komponente za raspodjelu poslova i upravljanje resursima

– Deploy - Flink je dizajniran tako da se može izvoditi lokalno na računalu, na YARN i samostalnim grozdovima, ili na Cloudu

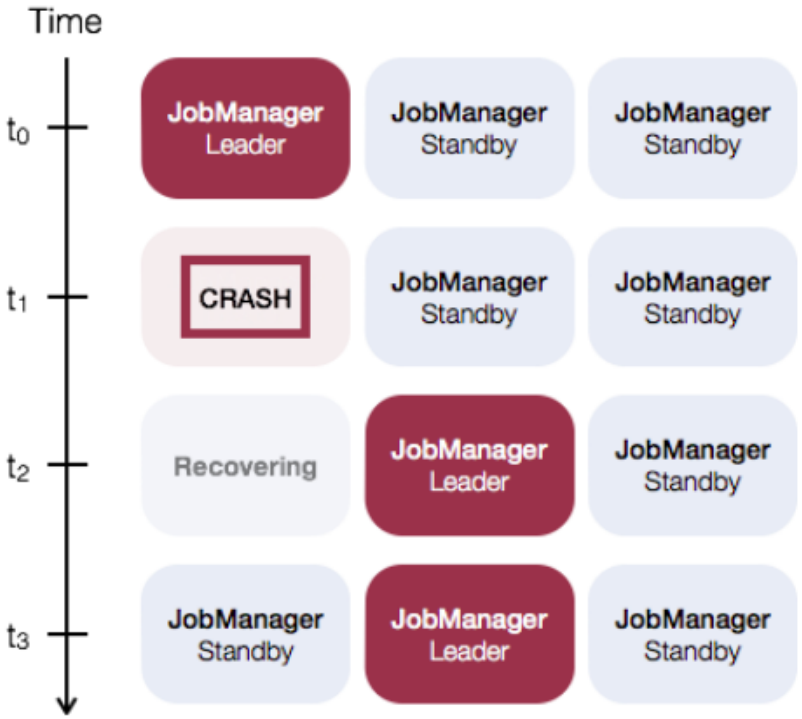
– API-ji i biblioteke - DataStream API za obradu neograničenih tokova, i DataSet API za obradu ograničenih tokova. To su sučelja koja programeri koriste za definiranje posla.



Slika 2.3: Apache Flink složaj [3]

2.2 Visoka dostupnost

Visoka dostupnost (engl.*High Availability)* zapravo znači "sustav mora biti dostupan kad god ja to želim". Prema zadanim postavkama, jednim Flink grozdom upravlja samo jedan upravljački čvor. To bi značilo da ako se njegov *Job Manager* sruši, ne mogu se primati novi programi, a oni koji su trenutno u izvođenju će se prekinuti. Ovo stvara *single point of failure* (SPOF), koju treba eliminrati. Visoku dostupnost u Flinku možemo konfigurirati u samostalnom i YARN grozdu. Opća ideja *Job Manager* visoke dostupnosti za samostalne grozdove je da u bilo koje vrijeme postoji jedan vodeći, i više *standby* Job Manager-a koji preuzimaju vodstvo u slučaju da vodeći ne uspije. Tako se osigurava da ne postoji *single point of failure* pa program može nastaviti s izvođenjem čim drugi *Job Manager* preuzme vodstvo.



Slika 2.4: Eliminiranje *single point of failure* točke [3]

Kod YARN grozda, ne pokrećemo više Job Manager instanci, već samo jednu, koju prilikom neuspjeha restarta YARN.

2.3. Tipovi podataka u Flinku

Flink podržava mnoge uobičajene vrste podataka, a najčešće korištene vrste mogu se grupirati u sljedeće kategorije:

– osnovni tipovi: svi Javini primitivni tipovi, void, String, Date, BigDecimal i BigInteger

– Java i Scala Tuples

– PoJo

– Flink Value tipovi

– Posebni tipovi

2.3.1. Tuples

Ovo je najčešće korišten tip podataka u Apache Flinku jer predstavlja lagan generički način enkapsuliranja podataka u Flinku. Tuple tipovi podataka su kompozitni tipovi koji se sastoje od fiksnog broja polja. Java Tuple može imati maksimalno 25 polja (null polja nisu podržana). Primjer Tuple objekta i takvog DataStream-a:

```
Tuple2<String, Integer> person = new Tuple2<>("Sara", 22);

String name = person.f0;

Integer age = person.f1;

DataStream<Tuple2<String,Integer>> persons = env.fromElements(

Tuple2.of("Sara", 22),

Tuple2.of("Ana", 23));
```

2.3.2. POJO

Flink analizira svaki tip podatka koji ne može svrstati u neku kategoriju i provjerava da li može biti identificiran i obrađen kao POJO tip. Uvjeti koje tip podatka mora zadovoljavati da bi se svrstao u POJO su:

– klasa mora biti javna

– klasa mora imati *deafult*-ni konstruktor

– sva polja su javna ili im se može pristupiti preko *getter*-a i *setter*-a

– sva polja imaju tipove koje podržava Flink

2.3.3. Value tipovi

Value tipovi implementiraju sučelje Value. Ono se sastoji od dvije metode read() i write() kako bi se implementirala logika serijalizacije i deserializacije. Ovakve metode se mogu iskoristiti za učinkovitije šifriranje zajedničkih vrijednosti od serije za opću namjenu. Flink dolazi s nekoliko svojih Value tipova, kao što su `IntValue`, `DoubleValue` i `StringValue`, koji pružaju zamjetne alternative za Java i Scala nepromjenjive primitivne vrste.

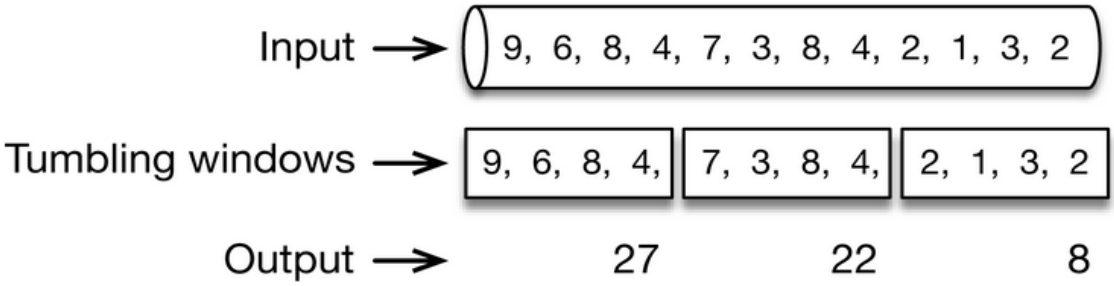
2.4. Prozori

Da bi izveli neke operacije nad beskonačnim tokom podataka koje ne obrađuju događaj po događaj, već više njih poput npr.agregacije, moramo ograničiti količinu podataka koju te operacije koriste. Prozor je dio toka podataka nad kojim izvodimo takve operacije. Predstavlja mehanizam prikupljanja grupe događaja po vremenu ili nekoj drugoj karakteristici(npr. broju objekata), kako bi se provela neka analiza nad tim događajima u cjelini (npr. zbrajanje). Najjednostavniji i najpopularniji prozori su oni temeljeni na vremenu. Flink ima dvije vrste takvih prozora, klizeći i padajući prozor. Glavna razlika je ta što se padajući prozori nemogu preklapati, dok klizeći to mogu.

Definiranje padajućeg prozora od 1 minute:

```
stream.timeWindow(Time.minutes(1))
```

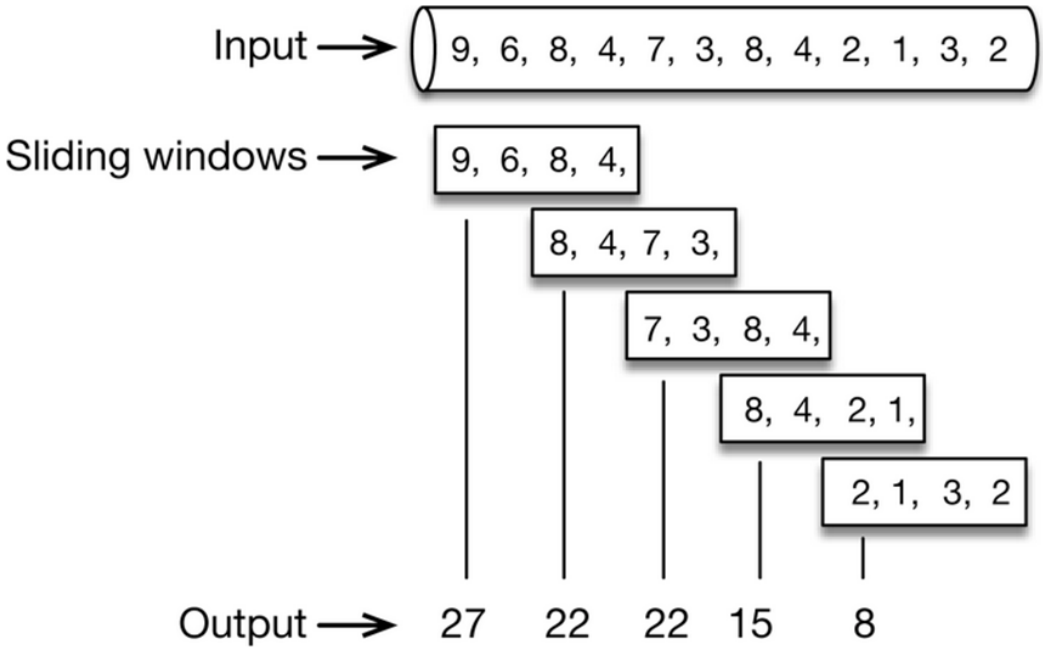
Na slici 3.5 je prikazan padajući prozor od 1 minute koji računa sumu vrijednosti na kraju minute. Svake minute trenutni prozor će se izračunati i stvoriti sljedeći prozor.



Slika 2.5: Padajući prozori [1]

Definiranje klizećeg prozora od 1 minute koji klizi svakih 30 sekundi:

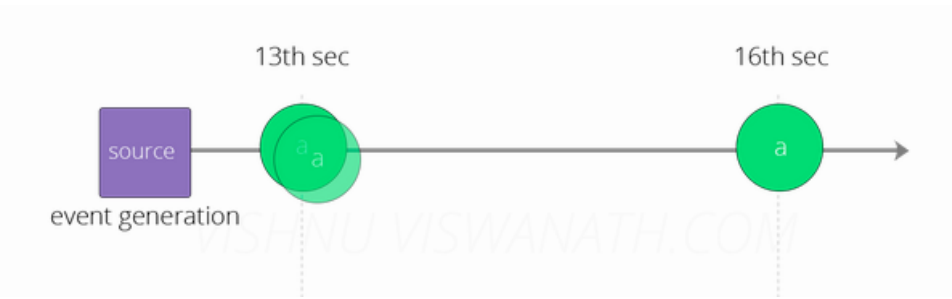
```
stream.timeWindow(Time.minutes(1), Time.seconds(30))
```



Slika 2.6: Klizni prozor [1]

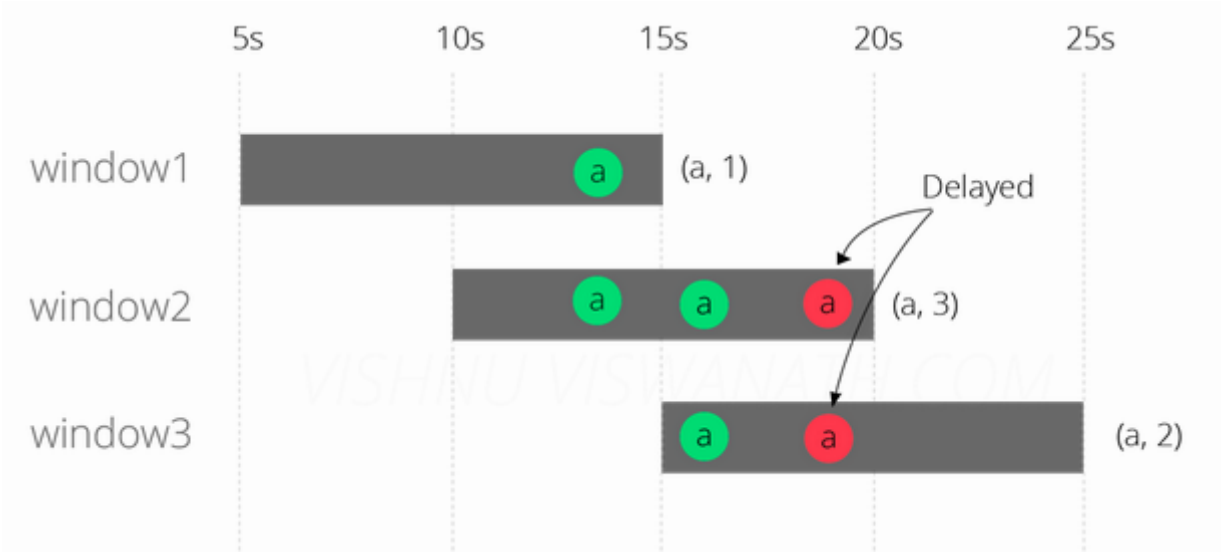
2.5. Upravljanje vremenom

Flink podržava različite pojmove vremena u programima za streaming. ***Event time,*** kao što samo ime govori, označava vrijeme kad se događaj dogodio u stvarnom svijetu. Obično podaci koje prikupljamo iz izvora kao što su senzori, u svojim zapisima imaju dodano neko vrijeme. Ovo vrijeme označava kad je taj događaj generiran na izvoru. Flink nam omogućuje da radimo s vremenom, pomoću *event time* podrške u svom programskom okviru. Temelji se na vremenskoj oznaci (engl.*timestamp*) koja je pridodana svakom događaju.



Slika 2.7: Dolazak poruka [5]

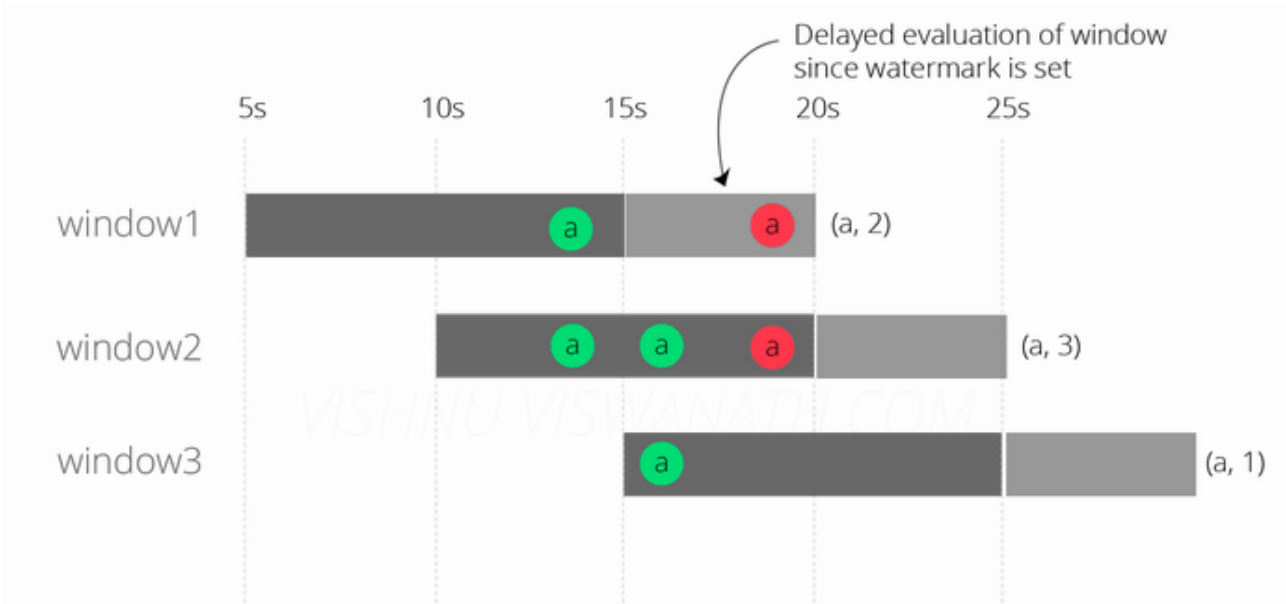
Recimo da je izvor generirao tri poruke u sekundama 13, 13, 16. Prozor je veličine 10 sekundi, tako da prve dvije poruke spadaju u prvi (5-15s) i drugi prozor(10-20s), a treća poruka u drugi i treći prozor(15-25s). Očekujemo da prvi prozor izbroji jednu poruku, drugi tri, a treći jednu poruku. Ako sad pretpostavimo da je druga poruka(u 13.s) zakasnila 6 sekundi, ona će upasti u drugi i treći prozor.



Slika 2.8: Kašnjenje poruke [5]

Promjene u izračunu kod drugog prozora nema jer je poruka i ovako trebala upasti u njega, ali zato prvi i treći prozor broje jednu manje, odnosno jednu više poruku. Ovaj problem se rješava *EventTime* obradom. Kao što je već rečeno, svaka poruka ima pridodanu vremensku oznaku, a nju možemo izvući s metodama koje implementiraju sučelje `TimestampExtractor<T>`. Po njoj Flink može provjeriti *event time* događaja, pa zato zakašnjelu poruku neće dodijeliti trećem prozoru. Međutim, kako je poruka stigla tek u 19. sekundi, evaluacija prvog prozora je završila pa se zato neće prepoznati da je trebala biti i u njemu. Za te svrhe se koristi oznaka koja se zove ***watermark***. Kada operator primi *watermark* u vremenu T pretpostavlja da neće primiti nikakve poruke starije od treutne vremenske oznake T. Nakon što postavimo *watermark* na neku vrijednost, Flink zna točno koliko sekundi poruka može zakasniti. Prozor će biti evaluiran tek kad prođe oznaku. Ako je primio zakašnjelu poruku, Flink izvuče *timestamp* i provjeri je li unutar dopuštenog kašnjenja. Možemo reći da *watermark* oznake pružaju sustavu logički sat koji ga informira o trenuntom *event time*-u. Kad se primi *watermark* operatorima je javljeno da su sve vremenske oznake za određeni vremenski interval promatrane i onda ili poredaju primljene događaje ili pokrenu računanje. U ovom primjeru *watermark* je postavljen na 5 sekundi, što znači

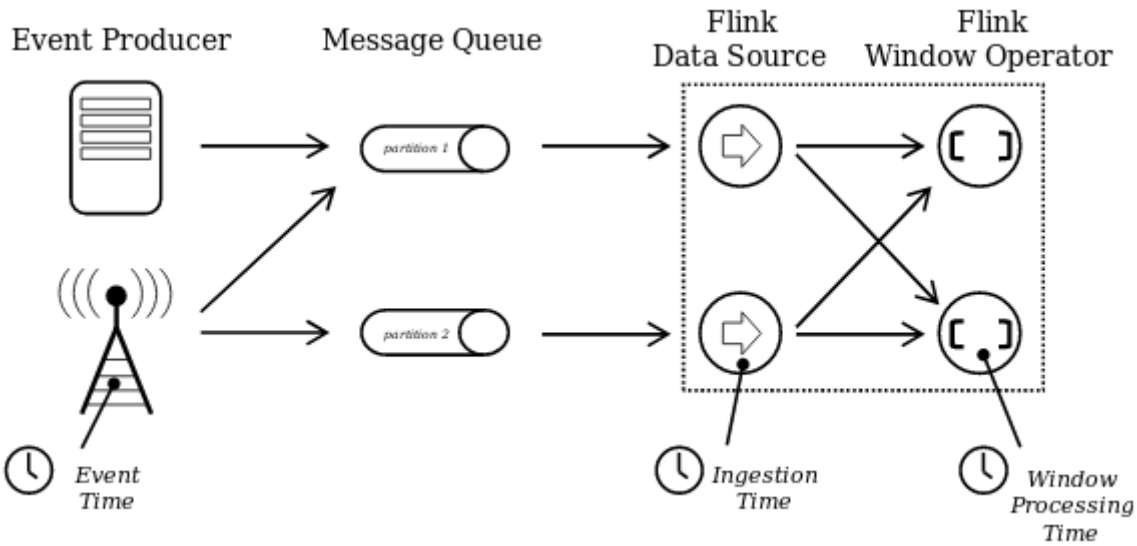
da će prvi prozor biti evaluiran u 20.sekundi pa Flink zna da ta druga poruka pripada i prvom prozoru.



Slika 2.9 : Obrada uz kontrolnu vremensku oznaku [5]

Processing time je obično lokalno vrijeme onog računala koje obrađuje tok podataka. Ako radimo sustav temeljen na ovom vremenu, ignorira se vremenska oznaka. To bi značilo da prozor uključuje sve događaje koji su stigli unutar perioda vremena mjerenog na računalu, tako da vjernu reprezentaciju stvarnog toka podataka. Kako se u obzir ne uzimaju zakašnjeli i neporedani događaji, u prozor samo trebaju pristići događaji i čim je određeno vrijeme isteklo pokrenuti računanje. Ovakvi prozori predstavljaju najmanju moguću latenciju, zato kod aplikacija gdje je brzina važnija od točnosti, *processing* time je bolji izbor. U obrnutoj situaciji nikako nebi trebali koristi ovo vrijeme jer daje nekonzistentne i nedeterminističke rezultate.

Vrijeme ulaska (eng.*Ingestion time*) je vrijeme kada se događaji učitaju u sustav. To je vrijeme između vremena događaja i vremena obrade. Obično, svako računalo u grozdu dodaje vremenske oznake da prati događaje. To može rezultirati malo nedosljednim prikazom podataka, zbog mogućih kašnjenja kroz grozd. Ali kod korištenja vremena učitavanja, vremenska oznaka se dodaje pri samom učitavanju i predstavlja trenutno vrijeme na izvoru tako da sva računala imaju isti pogled.



Slika 2.10 : Event time, Processing time, Ingestion time [3]

3. Obrada toka senzorskih podataka korištenjem platforme Apache Flink

U sklopu ovog završnog rada, kako bi se pokazala funkcionalnost programskog okvira Apache Flink, napravljena je streaming aplikacija "Fuel Consumption" koja koristi Flinkov DataStreamAPI. Zadatak aplikacije je iz senzorskih podataka o točenju goriva na benzinskim stanicama izračunati na kojem se spremniku troši najviše benzina svakih 30 minuta. Program pomoću Source funkcije generira tok podataka koji se čitaju iz .gz datoteke, zapisi su prikupljeni tijekom 3 dana i izgledaju ovako:

```
109,2013000109,2013000109,2014-02-05 00:00:55,DIESEL,0,0,20
110,2013000110,2013000110,2014-02-05 00:00:56,GASOLINE,0,0,17
111,2013000111,2013000111,2014-02-05 00:00:56,GASOLINE,0,0,45
```

Slika 3.1 : Ulazni podaci

Objekt klase FuelRecord predstavlja jedan zapis u datoteci, a ima osam atributa:

```
public long recordId;
public long stationId;
public long dispenserId; // id spremnika
public DateTime startTime;
public String fuelType; //tip goriva
public int factor;
public float tolls;
public float oiledLiters; //utoceno goriva
```

Klasa `FuelRecordsSource` implementira Flinkovu Source funkciju.Kako bi simulirala realističan tok podataka, SourceFunction servira događaje proporcionalno njihovim vremenskim oznakama. Brzina serviranja se predaje parametrom `ServingSpeedFactor` u konstruktoru klase, koji osim toga prima i put do datoteke.

```
final int servingSpeedFactor = 600; // događaji od 10 minuta su servirani u
1 sekundi
```

Logičko vrijeme serviranja se povećava za vrijednost faktora, za faktor 60, događaji od 1 minute(60s) su servirani u 1 sekundi. Ova source funkcija je EventSourceFunction i kontinuirano emitira kontrolne vremenske oznake (*watermarks)*, što znači da ovaj program radi u *event time*-u.

```
StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
```

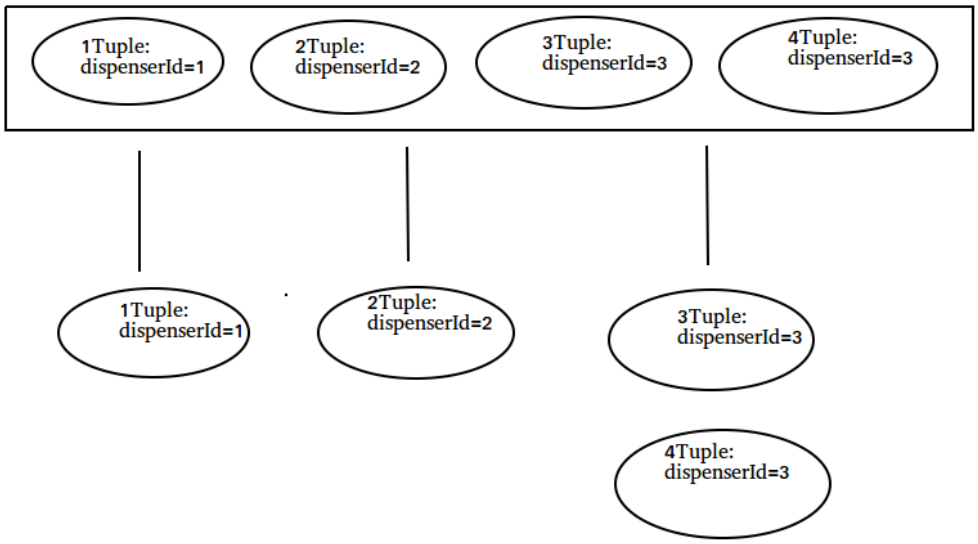
Također, implementira i Checkpointed sučelje pa se zato može oporaviti od grešaka ako se uključi checkpointing:

```
StreamExecutionEnvironment.enableCheckpointing(Long);
```

U klasi FuelConsumption u main() metodi, stvara se tok podataka s elementima koji su tipa FuelRecord

```
DataStream<FuelRecord> recordDataStreamSource = env.addSource(
new FuelRecordsSource(input, servingSpeedFactor));
```

Nad stvorenim tokom podataka se radi DataStream transformacija `keyBy` koja dijeli stream u razdvojene particije. Svi zapisi unutar istog ključa (`dispenserId)` se dodijeljuju istoj particiji. Transformacija vraća `KeyedStream` koji je onda podijeljen u prozore od 30 minuta. Kad obrađujemo prozor u takvom toku funkcija koju definiramo ima pristup samo elemntima s istim ključem. Funkcijom apply se nad svakim prozorom računa zbroj potrošenog goriva za svaki spremnik.



Slika 3.2: *KeyedStream*

```
DataStream<Tuple4<Long, Long, Integer, String>> litersPerThMin
= recordDataStreamSource.
          keyBy((FuelRecord record) -> record.dispenserId).
          timeWindow(Time.minutes(30)).
          apply(new Add());
```

Daljnjim transformacijama nalazi se spremnik s najvećim zbrojem goriva. Da bi se Flink program stvarno pokrenuo potrebno je pozvati:

```
env.execute();
```

Sve operacije do tad kao stvaranje izvora, transformacije, ispis, itd. samo grade graf unutarnjih operacija. Tek onda kad je pozvana execute() posao se počne izvršavati.

```
(1391562000000,2013009037,200)
(1391565600000,2013007636,265)
(1391572800000,2013010182,500)
(1391576400000,2013003071,500)
(1391581800000,2013011421,141)
```

Slika 4.2: Ispis izlaznih podataka

Na slici 4.2 su prikazani izlazni podaci koji predstavljaju vremensku oznaku na kraju svakih 30 min, id spremnika koji je potrošio najviše goriva, te količinu goriva.

4. Zaključak

Zbog sve većih zahtjeva za pristup informacijama u stvarnom vremenu, popularnost platformi za strujanje podataka je znatno porasla. Najvažniji dio streaming infrastrukture je procesor koji može pružiti visoku propusnost kroz širok spektar latencija i garantirati konzistentnost podataka. Apache Flink je upravo takva platforma otvorenog koda za strujanje podataka koja nudi ogromne mogućnosti za obradu podataka u stvarnom vremenu s velikom tolerancijom na greške na skali od milijun događaja u sekundi. Ključno je to što koristi minimalne resurse s milisekundnom latencijom.

5. Literatura

[1.] Introduction to Apache Flink by Kostas Tzoumas: Ellen Friedman

2. Stream Processing with Apache Flink by Vasiliki Kalavri; Fabian Hueske

[3.] https://ci.apache.org/projects/flink/flink-docs-release-1.5/

4. https://data-artisans.com/

[5.] https://vishnuviswanath.com/flink_streaming.html

6. http://www.baeldung.com/apache-flink

6. Sažetak

Obrada toka senzorskih podataka korištenjem platforme Apache Flink

Obrada podataka se razvijala paralelno sa razvojem društva. Razvojem novih hardverskih komponenti, računalna obrada podataka postaje svakodnevno korištena u svim ljudskim djelatnostima. Veliki podaci su sad već dobro poznat pojam u svijetu tehnologije i odlikuju se glavnim karakteristikama poput volumena, raznolikosti i brzine. Za obradu takvih podataka u stvarnom vremenu postoji nekoliko *open source* platforma Ovaj rad bavi se funkcionalnostima platforme Apache Flink s posebnim osvrtom na njegov DataStreamAPI za *stream processing.*

Summary

Sensor Datastream Processing with Apache Flink Platform

Data processing developed parallel to the development of society. With the development of new hardware components, computer data processing is in everyday use in all human activities. Big data is now a well-known term in the world of technology and is distinguished by the main features such as volume, diversity and velocity. There are several open source platforms for real-time data processing. This paper deals with Apache Flink platform functionality with a special focus on its DataStreamAPI for processing streams.