

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

# Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow Tehnička dokumentacija Verzija 1.0

**Studentski tim:** Ivan Bratulić  
Bernard Ibrović  
Bernard Kazazić  
Ana Keri  
Bono Tadić

**Nastavnik:** prof. dr. sc. Krešimir Pripužić  
doc. dr. sc. Martina Antonić

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## Sadržaj

1. Opis razvijenog proizvoda	4
2. Usporedba operatora između Apache Flink i Timely Dataflow	6
3. Usporedba performansi između Apache Flink i Timely Dataflow na grozdu računala	16
4. Literatura	21

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## Tehnička dokumentacija

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## Opis razvijenog proizvoda

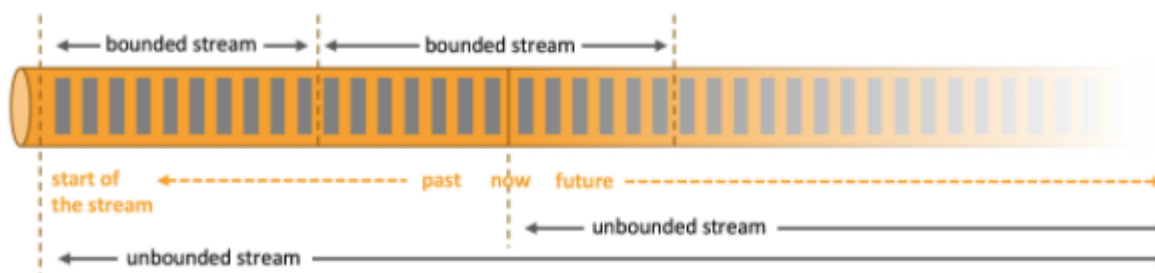
Cilj projekta je usporediti performanse između dva različita sustava za obradu tokova podataka, Apache Flink i Timely Dataflow. Tok podataka je prebrojivo beskonačan niz elemenata. Različiti modeli tokova podataka imaju različite pristupe ovisno o promjenjivosti toka i strukturi elemenata toka. Obrada toka odnosi se na analiziranje tokova podataka u stvarnom vremenu kako bi se dobio određeni rezultat. Vrijeme je važan koncept u obradi tokova podataka: u skoro svim modelima svaki element toka ima vremenski trenutak koji označava npr. vrijeme stvaranja elementa, vrijedi li još u nekom trenutku taj element ili kada će element biti spreman za obradu. Neki primjeri obrade toka su: očitavanje senzora koji mjeri temperaturu ili neke druge parametre u prirodi, očitavanje stanja vrijednosti dionica, očitavanje trenutnog stanja robe u inventaru i mnogi drugi.

## Apache Flink

Apache Flink je programski okvir koji se koristi za distribuiranu obradu preko neograničenih i ograničenih tokova.

Neograničeni tokovi – imaju početak ali nemaju definiran kraj. Moraju biti kontinuirano obrađivani, odnosno događaji se moraju obraditi odmah po dolasku. Obrada neograničenih tokova podataka često zahtijeva da se događaji unose određenim redoslijedom, kao što je redoslijed u kojem su se događaji dogodili, kako bi se moglo zaključiti o potpunosti rezultata.

Ograničeni tokovi – imaju definirani početak i kraj. Mogu se obraditi unosom svih podataka prije izvođenja bilo kakvih izračuna. Ograničeni skup podataka se uvijek može sortirati. Obrada ograničenih tokova također je poznata kao „batch“ obrada, odnosno paketna obrada.



**Slika 1: Tok podataka u Apache Flinku**

Apache Flink je distribuirani sustav i zahtijeva računalne resurse za izvršavanje aplikacija. Flink se integrira sa svim uobičajenim upraviteljima resursa klastera kao što su Hadoop YARN, Apache Mesos i Kubernetes, ali se također može postaviti da radi kao samostalni grozd računala. [1][2]

Korisnici prijavljuju impresivne skalabilne brojke Flink aplikacija koje pokreću, kao što su:

- aplikacije obrađuju nekoliko trilijuna događaja po danu
- aplikacije koje održavaju više terabajta stanja
- aplikacije koje rade na tisućama jezgri

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## Timely Dataflow

Timely Dataflow je nastao iz rada u Microsoft Researchu gdje je jedan tim radio na razvoju skalabilnih i distribuiranih platformi za obradu podataka. Cilj je bio pružiti ugodno iskustvo u kojemu se može pisati sofisticirani izračuni tokova koji se dalje prevode u sustave koji se izvode s malim opterećenjem.

Paralelizacija podataka - operateri u Timely Dataflowu mogu istovremeno raditi na neovisnim dijelovima podataka. To omogućuje temeljnom sustavu da distribuira pravovremene izračune protoka podataka među više paralelnih radnika. To mogu biti niti na računalu ili čak niti na računalima u grozdu računala. Ova distribucija obično poboljšava propusnost sustava i omogućuje skaliranje do većih problema s pristupom većem broju resursa.

Tok podataka - osnovni tip podataka u Timely Dataflowu je tok podataka, neograničena zbirka podataka koji nisu svi trenutno dostupni, ali umjesto toga pristižu kako se izračunavanje odvija. Tokovi su korisna generalizacija statičkih skupova podataka za koje se pretpostavlja da su dostupni na početku izračunavanja. Izražavajući svoj program kao izračun nad tokovima, objašnjava se programu kako bi trebao reagirati na statičke ulazne skupove podataka (ubaciti sve podatke odjednom), ali i kako bi trebao reagirati na nove podatke koji bi mogli stići kasnije.

Ekspresivnost - glavni dodatak u odnosu na tradicionalne sustave za obradu tokova podataka je njegova sposobnost izražavanja kontrolnih konstrukcija više razine, poput iteracije. To pomiče izračune toka s ograničenja pravolinijskog koda u svijet algoritama.

Timely Dataflow je *sustav protoka podataka*, a to znači da u svojoj srži voli premještati podatke. To komplicira stvari kada ne želite premještati podatke, već umjesto toga premještati stvari poput pokazivača i referenci na podatke koji inače ostaju na mjestu.

Sustavi protoka podataka s e također temelje na razbijanju izvođenja programa na dijelove koji neovisno djeluju. Međutim, mnogi programi su točni samo zato što se neke stvari događaju prije ili poslije drugih stvari.

Timely Dataflow nalazi se u prostoru između knjižnice jezika i runtime sustava. To znači da nema baš jamstva stabilnosti koju bi knjižnica mogla imati.

Timely Dataflow oslanja se na dva temeljna pojma: **vremenske oznake** i **tok podataka**, koji zajedno čine koncept **napretka**.

### Tok podataka

Namjena "programiranja u obliku toka podataka" je stvaranje programa koji se sastoji od nezavisnih komponenti, od kojih svaka obavlja određenu radnju na temelju ulaznih podataka. Također je potrebno i opisati kako su te komponente međusobno povezane.

Najvažniji dio je upravo neovisnost komponenti. Pri takvom načinu programiranja, računalu se daje fleksibilnost što se tiče načina izvođenja programa. Umjesto navođenja točno određenog slijeda uputa koje bi računalo trebalo izvršavati, računalo može obraditi svaku komponentu kako ono smatra da je najbolje.

### Vremenske oznake

Za iskorištavanje punog potencijala "programiranja u obliku toka podataka", prvo je potrebno shvatiti na koji način računalo vrši instrukcije. Primjerice, u klasičnom, imperativnom načinu programiranja, kad tijek izvođenja programa dođe do određenog dijela, sa sigurnošću se može reći da je sav posao do tog trenutka obavljen. To proizlazi iz činjenice da se instrukcije izvršavaju slijedno. Umjesto takvog pristupa, podaci koji prolaze kroz tok se označe vremenskom oznakom, te se na temelju tih oznaka može (otprilike) odrediti kad bi se one izvršile da su se izvodile slijedno.

Vremenske oznake imaju barem dvije uloge:

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

1. omogućuju komponentama razlučivanje redoslijeda kojim podaci pristižu, te korisnik pomoću oznaka može saznati jesu li svi podaci s istom vremenskom oznakom obrađeni.

2. U suštini, vremenske oznake omogućuju uvođenje slijedne strukture izvođenja naredbi u program, bez potrebe za stvarnom slijednom izvedbom.

### Napredak

U klasičnom, imperativom programu, pronaći najveći broj iz skupa brojeva je trivijalno. Potrebno je jednostavno proći kroz niz i vratiti najveći broj. Korisnik može biti siguran da su svi brojevi pregledani, budući da je konačan skup brojeva poznat unaprijed.

Ovakav zadatak postaje mnogo teži kad se izvodi u okruženju toka podataka, odnosno okruženju u kojem brojevi pristižu na ulaz u komponentu kojoj je zadatak vratiti najveći broj. Prije nego što vrati rezultat, komponenta mora znati jesu li joj dostavljeni svi podaci na ulaz, jer bi upravo sljedeći broj mogao biti najveći. Postavlja se pitanje kako pomoću toka podataka i vremenskih oznaka omogućiti komponentama da znaju je li ulazni niz potpun ili će morati obraditi još podataka.

Kombinira li se struktura toka podataka programa s vremenski označenim podacima na način da se, kako se podaci kreću niz tok, njihove vremenske oznake mogu samo povećati, moguće je odrediti napredak trenutnog izračuna. Točnije rečeno, unutar bilo koje komponente unutar strukture toka podataka, poznato je koje se vremenske oznake eventualno mogu pojaviti nekad u budućnosti. Za vremenske oznake koje više nisu moguće (one oznake čija je vrijednost manja od trenutne), kaže se da su „prošle“, a komponente se na temelju ovih informacija ponašaju kako misle da je najbolje.[3][4][5][6]

## Usporedba operatora između Apache Flink i Timely Dataflow

### Operator za brojanje riječi

<sup>i</sup>U nastavku se nalazi primjer klase za razdvajanje reda teksta na pojedinačne riječi:

```
public static final class Tokenizer
    implements FlatMapFunction<String, Tuple2<String, Integer>> {

    @Override
    public void flatMap(String value, Collector<Tuple2<String, Integer>> out) {
        // normalize and split the line
        String[] tokens = value.toLowerCase().split("\\W+");

        // emit the pairs
        for (String token : tokens) {
            if (token.length() > 0) {
                out.collect(new Tuple2<>(token, 1));
            }
        }
    }
}
```

Slika 2: Klasa Tokenizer

Proces razdvajanja teksta na pojedinačne riječi kako bi se mogle pobrojati ostvarujemo kroz statičnu klasu Tokenizer koja implementira funkcijsko sučelje FlatMapFunction s ulaznim parametrom tipa String i izlaznim parametrom tipa Tuple2<String, Integer> kako bi ju mogli koristiti tokom mapiranja. Funkcija

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

flatMap kao dva parametra ima parametar value koji predstavlja redak teksta koji se analizira i parametar out koji predstavlja Tuple2<String, Integer>, odnosno par vrijednosti (riječ i 1). U funkciji flatMap prvo razdvajamo redak teksta bez velikih slova između znakova koji nisu slova, brojevi ili podvlaka naredbom toLowerCase().split("\\W+") i spremamo ih u niz Stringova tokens. Potom prolazimo po svakom tako napravljenom tokenu i za svaki koji ima duljinu veću od 0 stvaramo par vrijednosti (riječ, 1) koji predstavlja jedno pojavljivanje te riječi. Nju vraćamo kroz naredbu out.collect(new Tuple2<String, Integer>(token, 1)). [7]

Timely Dataflow na sličan način razdvaja liniju teksta u tok riječi, ali za razliku od Apache Flinka ne stvara novu klasu nego se definira unutar Dataflow radnika:

```
// create a new input, exchange data, and inspect its output
worker.dataflow:<:use, -, _>(|scope : &mut Child<Worker<Generic>, usize> | {
  lines_input.to_stream(scope) : Stream<Child<...>, String>
  // map line from text to words.
  .flat_map(|text: String| text.to_lowercase() : String
    .split(|c : char| !c.is_alphabetic()) : impl Iterator<Item=&str>
    .filter(|word : &str | word.is_empty().not()) : impl Iterator<Item=&str>
    .map(move |word : &str | (word.to_owned(), 1)) : impl Iterator<Item=(...)>
    .collect::<Vec<_>>()
  ) : Stream<Child<...>, (...)>
```

Slika 3: Razdvajanje riječi u Timely Dataflowu

Svaki radnik prvo svoj input handle pretvori u tok linija teksta. Zatim sva slova iz linije teksta pretvori u mala slova, razdvoji ju na svim znakovima koji nisu slovo i filtrira sve prazne String podatke. Iz linije teksta kao i u Flinku stvara tok parova (riječ, 1) koji će nam kasnije služiti za prebrojavanje riječi.

Operator za prebrojavanje riječi u Apache Flinku je sljedeći:

```
DataStream<Tuple2<String, Integer>> counts =
    text.flatMap(new Tokenizer())
        .name("tokenizer")
        .keyBy(value -> value.f0)
        .sum(1)
        .name("counter");
```

Slika 4: Operator za prebrojavanje riječi u Apache Flinku

Prebrojavanje se radi tako da se nad ulaznim DataStream elementom text (tok linija teksta) poziva mapiranje klasom Tokenizer imena tokenizer koji sve linije toka text pretvori u parove (token, 1). Naredbom .keyBy(value → value.f0) grupiramo sve vrijednosti polja "0" para koji se vraća, odnosno grupiramo sve pojedinačne riječi. Potom naredbom sum(1) zbrojimo sve vrijednosti polja "1" tako grupiranih riječi kako bi dobili broj pojavljivanja za svaku riječ. DataStream counts time predstavlja tok vrijednosti (riječ, broj ponavljanja) te se jedna po jedna vrijednost ispisuje.

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

Timely Dataflow ima dosta drugačiju strukturu operatora za brojanje riječi jer mora paziti na vremenske oznake toka podataka. Svaki radnik mora obraditi podatke tako da izgledaju kao da su obrađeni redom kako su ulazili u sustav makar ih ne dobivaju ispravnim točnim redoslijedom.

```
// define word count operator.
.unary_frontier( pact: exchange, name: "WordCount", constructor: |_capability, _info| {
  // operator local storage.
  let mut queues :HashMap<Capability<usize>, Vec<Vec<...>>> = HashMap::new();
  let mut counts :HashMap<String, u64> = HashMap::new();
  let mut buffer :Vec<...> = Vec::new();

  move |input : &mut FrontieredInputHandle<...>, output : &mut OutputHandle<...> | {
    // for each input stash it in queue at 'time'.
    while let Some((time : CapabilityRef<usize>, data : RefOrMut<Vec<...>>>)) = input.next() {
      queues.entry( key: time.retain()) : Entry<Capability<usize>, Vec<...>>
        .or_insert( default: Vec::new()) : &mut Vec<Vec<...>>, Global>
        .push( value: data.replace( element: Vec::new()));
    }

    // enable data in queue that is less or equal to input frontier time.
    for (time : &Capability<usize>, vals : &mut Vec<Vec<...>>> ) in queues.iter_mut() {
      if !input.frontier().less_equal( time: time.time()) {
        let vals : Vec<Vec<...>>> = std::mem::replace( dest: vals, src: Vec::new());
        buffer.push( value: (time.clone(), vals));
      }
    }

    // drop complete time and allocations.
    queues.retain(|_, vals : &mut Vec<Vec<...>>> | vals.len() > 0);
    // sort ready updates by time.
    buffer.sort_by(|x : &(Capability<usize>, Vec<Vec<...>>>), y : &(Capability<usize>, Vec<Vec<...>>>)|
      (x.0).time().cmp( other: &(y.0).time()));

    // process inputs in order.
    for (time : Capability<usize>, mut val : Vec<Vec<...>>>) in buffer.drain( range: ..) {
      if !input.frontier().less_equal( time: time.time()) {
        let mut session : Session<...> = output.session( cap: &time);
        for mut batch : Vec<...> in val.drain( range: ..) {
          for(word : String, add : u64) in batch.drain( range: ..) {
            let entry : &mut u64 = counts.entry( key: word.clone()).or_insert( default: 0u64);
            *entry += add;
            session.give( data: (word, *entry));
          }
        }
      }
    }
  }
}
```

Slika 5: Operator za brojanje riječi u Timely Dataflowu

Operator ima svoj red čekanja za obradu, mapu sa riječima i brojem njihovog ponavljanja te buffer koji će nam kasnije koristiti. Svaki ulazni podatak (riječ, 1) se potom sprema u red čekanja sa svojom vremenskom oznakom u kojoj je nastao. Podaci čije je zapisano vrijeme veće od trenutnog vremena ProbeHandle-a koji je spojen na input neće biti proslijeđeni u buffer za daljnju obradu. Mi ćemo prazne zapise koje smo prebacili iz reda čekanja u buffer i sortiramo podatke u bufferu po vremenu dolaska.



Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

Procesiramo podatke u bufferu, spremamo ih u mapu sa podacima o broju viđenih riječi i šaljemo u sesiju sa vremenskom oznakom riječi koje obrađujemo podatak (riječ, broj ponavljanja) na inspekciju.

```
// print word and number of seen times.
.inspect(move |x : &(String, u64) | println!("{ } seen: {:?}", worker_id, x)) : Stream<Child<...>, (...)>
```

**Slika 6: Ispis informacije koliko je puta neka riječ viđena**

Radnik Dataflowa svaki par (riječ, broj ponavljanja) ispisuje na ekran sa svojom id oznakom da se zna koji radnik je obradio tu riječ.

Za razliku od operatora u Apache Flinku, Timely Dataflow operator za brojanje riječi ne stvara tok parova (riječ, broj ponavljanja) koji će se ispisati nakon što je obradio dokument nego kontinuirano kako radnik obrađuje riječ po riječ dojavljuje za svaku riječ koliko puta dosad se pojavila.

Primjer ulaza:

```
1 jedan dva tri cetiri pet
2 sest sedam osam devet deset
3 jedan dva tri pet sest
4 sedam osam devet deset jedanaest
```

**Slika 7: Primjer ulaznih podataka**

Primjer izlaza operatora u Apache Flinku:

```
(jedan,2)
(dva,2)
(tri,2)
(cetiri,1)
(pet,2)
(sest,2)
(sedam,2)
(osam,2)
(devet,2)
(deset,2)
(jedanaest,1)
```

**Slika 8: Primjer izlaznih podataka u Apache Flinku**

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

Primjer izlaza operatora u Timely Dataflowu sa jednim radnikom:

```
0 seen: ("jedan", 1)
0 seen: ("dva", 1)
0 seen: ("tri", 1)
0 seen: ("cetiri", 1)
0 seen: ("pet", 1)
0 seen: ("sest", 1)
0 seen: ("sedam", 1)
0 seen: ("osam", 1)
0 seen: ("devet", 1)
0 seen: ("deset", 1)
0 seen: ("jedan", 2)
0 seen: ("dva", 2)
0 seen: ("tri", 2)
0 seen: ("cetiri", 2)
0 seen: ("pet", 2)
0 seen: ("sest", 2)
0 seen: ("sedam", 2)
0 seen: ("osam", 2)
0 seen: ("devet", 2)
0 seen: ("deset", 2)
0 seen: ("jedanaest", 1)
```

**Slika 9: Primjer izlaznih podataka u Timely Dataflowu s jednim radnikom**

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

Primjer izlaza operatora u Timely Dataflowu sa dva radnika:

```
0 seen: ("jedan", 1)
0 seen: ("tri", 1)
1 seen: ("dva", 1)
1 seen: ("cetiri", 1)
1 seen: ("pet", 1)
0 seen: ("sedam", 1)
0 seen: ("osam", 1)
1 seen: ("sest", 1)
0 seen: ("jedan", 2)
1 seen: ("devet", 1)
0 seen: ("tri", 2)
1 seen: ("deset", 1)
0 seen: ("sedam", 2)
0 seen: ("osam", 2)
1 seen: ("dva", 2)
1 seen: ("cetiri", 2)
1 seen: ("pet", 2)
1 seen: ("sest", 2)
1 seen: ("devet", 2)
1 seen: ("deset", 2)
1 seen: ("jedanaest", 1)
```

**Slika 10: Primjer izlaznih podataka u Timely Dataflowu s dva radnika**

## Operator za filtriranje podataka (filter)

Apache Flink koristi protočni mehanizam za obradu i analizu podataka u stvarnom vremenu, zbog čega ima mogućnost čitanja neograničenih tokova podataka iz vanjskih izvora pod nazivom DataStreams. DataStreams podupiru operatore, koji se koriste za obradu podataka kao što su pridruživanje, grupiranje, filtriranje i aritmetičke operacije.

Operator filter() evaluira boolean funkciju za svaki element toka podataka i zadržava one elemente za koje funkcija vraća "true", odnosno istinitu vrijednost. U nastavku se nalazi primjer programa u kojem se koristi spomenuti operator:

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

```
package package1;

import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

public class StreamingJob {

    public static void main(String[] args) throws Exception {
        // set up the streaming execution environment
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Create a DataStream from a list of elements
        DataStream<Integer> myInts = env.fromElements(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
        // Keeping only even numbers
        DataStream<Integer> filteredInts = myInts.filter(x -> x % 2 == 0);

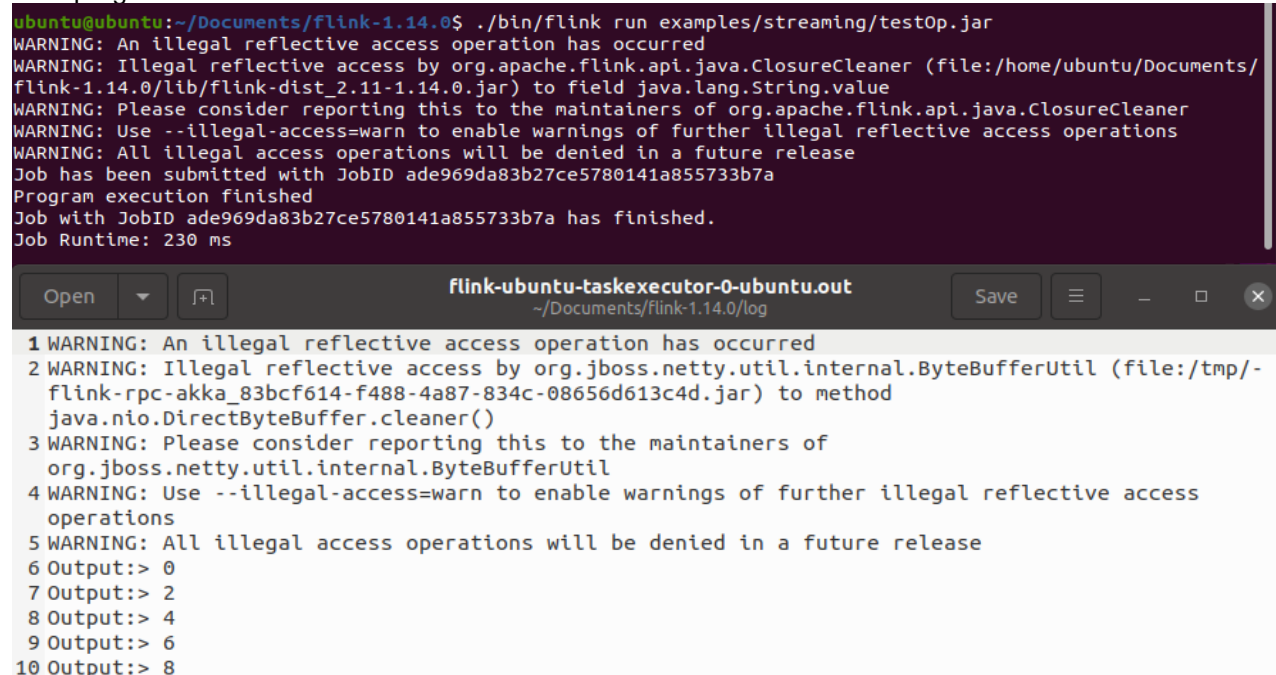
        filteredInts.print("Output:");

        // execute program
        env.execute();
    }
}
```

**Slika 11: Filter operator u Apache Flinku**

Način rada je sljedeći: naredbe uvoza omogućuju korištenje klasa koje su potrebne u programu. U metodi main(), statičkom metodom getExecutionEnvironment() omogućujemo stvaranje lokalnog okruženja koje će izvršiti program. Sljedeća metoda je fromElements(OUT...data) koja prima niz elemenata istog tipa, koji su u ovom slučaju cijeli brojevi od 0 do 9, te vraća tok podataka 'DataStream' istog tipa. Nakon toga slijedi operator filter() primjenom kojeg će se napraviti novi tok podataka koji će sadržavati parne brojeve početnog toka podataka. Filtriranjem se dobio željeni podatkovni tok koji se na kraju metodom print() ispisuje na izlaz. Kako bi se program izvršio, bilo na lokalnom računalu ili slanjem na klaster potrebno je pozvati metodu execute() na StreamExecutionEnvironment.

Izlaz programa:



```
ubuntu@ubuntu:~/Documents/flink-1.14.0$ ./bin/flink run examples/streaming/testOp.jar
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.flink.api.java.ClosureCleaner (file:/home/ubuntu/Documents/flink-1.14.0/lib/flink-dist_2.11-1.14.0.jar) to field java.lang.String.value
WARNING: Please consider reporting this to the maintainers of org.apache.flink.api.java.ClosureCleaner
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Job has been submitted with JobID ade969da83b27ce5780141a855733b7a
Program execution finished
Job with JobID ade969da83b27ce5780141a855733b7a has finished.
Job Runtime: 230 ms

1 WARNING: An illegal reflective access operation has occurred
2 WARNING: Illegal reflective access by org.jboss.netty.util.internal.ByteBufferUtil (file:/tmp/-flink-rpc-akka_83bcf614-f488-4a87-834c-08656d613c4d.jar) to method
  java.nio.DirectByteBuffer.cleaner()
3 WARNING: Please consider reporting this to the maintainers of
  org.jboss.netty.util.internal.ByteBufferUtil
4 WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access
  operations
5 WARNING: All illegal access operations will be denied in a future release
6 Output:> 0
7 Output:> 2
8 Output:> 4
9 Output:> 6
10 Output:> 8
```

**Slika 12: Izlaz programa**

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## Filtriranje u Timely Dataflowu

Filtriranje je jedna od temeljnih operacija sustava Timely Dataflow. Prilikom filtriranja, podniz koji se želi zadržati određen je predikatom. Primjer programa u kojem se koristi operator `filter()` prikazan je u nastavku:

```
extern crate timely;

use timely::dataflow::operators::{ToStream, Inspect, Filter};

fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        worker.dataflow::<<(),_,_>(|scope| {
            (0 .. 9)
                .to_stream(scope)
                .filter(|x| *x % 2 == 0)
                .inspect(|x| println!("hello: {}", x));
        });
    }).unwrap();
}
```

**Slika 13: Filter operator u Timely Dataflowu**

Način rada je sljedeći: komandom `use` je zadan niz operatora koji će biti korišteni, među njima je i `Filter`. U funkciji `main()` naredbom `execute_from_args()` se inicijalizira i pokreće tok podataka. S obzirom na složenost primjera, dovoljno je korištenje samo jednog `worker` objekta, odnosno nije potrebna višedretvenost. Objektom `scope` je definiran ulazni niz podataka, u ovom je slučaju to niz cijeli brojeva od 0 do 9. Budući da Timely Dataflow obrađuje *tokove* podataka, ulazni niz je pretvoren u tok naredbom `to_stream()`. Operacija filtriranja se izvršava naredbom `filter()`. Taj proces funkcionira na način da primi referencu na podatak (referenca se dohvati dodavanjem simbola `*` prije imena varijable), što omogućuje provođenje predikata bez mijenjanja ulaznih podataka. Predikat je u ovom primjeru broj koji je djeljiv s 2. Za ispis rezultata korišten je operator `inspect()` u kombinaciji s naredbom `println()`. [8]

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

Izlaz je prikazan u sljedećem odsječku:

```
PS C:\Users\filter> cargo run
    Compiling filter v0.1.0 (C:\Users\filter)
    Finished dev [unoptimized + debuginfo] target(s) in 1.60s
    Running `target\debug\filter.exe`
hello: 0
hello: 2
hello: 4
hello: 6
hello: 8
```

Slika 14: Izlaz programa u Timely Dataflowu

## Usporedba između operatatora

Apache Flink i Timely Dataflow dijele puno istih operatatora dok neki operatori postoje u Apache Flinku a ne postoje u Timely Dataflowu i obrnuto.

## Primjeri zajedničkih operatatora između Apache Flinka i Timely Dataflowa

### Operator **Filter**

Operator evaluira boolean funkciju za svaki element i ostavlja one elemente za koje funkcija vrati true.

### Operator **Map**

Operator uzima jedan element i proizvodi jedan element, tj. mapira tok podataka u novi tok s promjenom elemenata.

### Operator **FlatMap**

Operator uzima jedan element i proizvodi 0, 1 ili više elemenata.

### Operator **Aggregate**

Operator spaja grupu vrijednosti u jednu vrijednost. Aggregate je operator koji ima ugrađene Reduce funkcije.

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## Primjeri operatora koji postoje u Apache Flinku a ne postoje u Timely Dataflowu

### Operator **Reduce**

Operator kombinira grupu elemenata u jedan element tako što se funkcija poziva za prvi element bez ostalih, nakon čega se poziva funkcija sa drugim elementom bez ostalih sve dok ne ostane samo jedan element. Iako Reduce ne postoji u Timely Dataflowu, postoji Aggregate koji ima sličan efekt.

### Operator **Union**

Operator napravi uniju dvaju ili više tokova podataka gdje je novi tok onaj u kojem se nalaze svi elementi iz svih tokova podataka.

## Primjeri operatora koji postoje u Timely Dataflowu a ne postoje u Apache Flinku

### Operatori **Branch** i **BranchWhen**

Operatori uzimaju ulazni tok podataka i pretvaraju ga u dva izlazna toka podataka. Razlika između Branch i BranchWhen je ta što BranchWhen grana podatke na temelju uvjeta.

### Operator **OkErr**

Tok podataka za grešku postoji i u Apache Flinku, no u Timely Dataflowu postoji operator s ugrađenim sposobnostima njegovog korištenja. OkErr operator uzima tok podataka i grana ga na dva toka podataka. Za svaki element se poziva funkcija koja, ukoliko vrati true, šalje element na prvi tok podataka gdje je vraćen Ok(x) gdje x predstavlja navedeni element, inače šalje element na drugi (Err(e)) tok podataka gdje proslijeđuje e za error.

### Operator **Accumulate**

Operator zbraja broj ponavljanja nekog elementa unutar određenog vremenskog intervala.

## Operatori specifični za Apache Flink

Uz navedene iste i različite operatore, Apache Flink također ima specifične operatore koji se zovu prozori, engl. window. Postoje vremenski prozori (engl. time-based, physical window), to su „svi objekti u zadnjih 20 minuta“, i prozor temeljen na broju objekata (engl. count-based, tuple-based, logical window), to su „zadnjih 10 objekata“. Neke od funkcija su:

- Stvaranje toka s elementima u prozoru
  - o `JavaDStream window(Duration windowDuration, Duration slideDuration)`
- Brojanje elemenata u prozoru
  - o `JavaDStream countByWindow(Duration windowDuration, Duration slideDuration)`
- Reduciranje elemenata u prozoru
  - o `JavaDStream reduceByWindow(Function2 reduceFunc, Duration windowDuration, Duration slideDuration)`
  - o `JavaDStream reduceByWindow(Function2 reduceFunc, Function2 invReduceFunc, Duration windowDuration, Duration slideDuration)`

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

# Usporedba performansi između Apache Flink i Timely Dataflow na grozdu računala

## Izvođenje koda na grozdu računala

Primjeri koji se pokreću na grozdu računala sastoje se od 10 milijuna riječi Lorem ipsum teksta za operator za brojanje riječi te podataka o taxi vožnjama koje smo prvo parsirali i ostavili samo one podatke koji su relevantni za filtriranje jer nismo htjeli da parsiranje utječe na performanse budući da je cilj usporediti filter operator. Za filter operator postoje dva primjera, jedan primjer je filtriranje vrijednosti veće od 1.0, dok je drugi primjer filtriranje vrijednosti između 0.5 i 1.0.

## Operator za brojanje riječi

### Izvođenje preko Apache Flink – trenutno se ne pokreće na grozdu

### Izvođenje preko Timely Dataflow

<pre>-h hostname.txt -n 16 -p 0' (pretium, 65520) (nam, 50930) (clementum, 29110) (lacus, 87320) (metus, 94630) (cut, 203800) (sapian, 58230) (dictum, 14560) (est, 29110) It took 16665ms to finish calculation.</pre>	<pre>(curabitur, 65500) (purus, 50960) (scelusque, 21830) (cleitend, 36400) (platea, 14560) (litora, 7270) (consequat, 21840) (phasellus, 94630) (interdum, 21830) (eu, 109170) (mus, 7280) It took 15888ms to finish calculation.</pre>	<pre>(auctor, 50950) (justo, 58240) (veneratis, 43670) (aliquet, 29180) (portitor, 21840) (lacinia, 21830) (integer, 14560) (hac, 14560) (facilisi, 7280) (tempus, 43660) (conubia, 7270) It took 15165ms to finish calculation.</pre>	<pre>(penatibus, 7280) (sodales, 29120) (cubilia, 21840) (ed, 7270) (nibh, 50950) (hendrerit, 7270) (vestibulum, 167390) (dignissim, 14560) (egret, 138990) (a, 138280) (montes, 7280) It took 14408ms to finish calculation.</pre>
<pre>Running 'target/release/count_words_operator' -h hostname.txt -n 16 -p 8' (faucibus, 72780) (viverra, 50950) (pharetra, 14560) (blandit, 58230) (ullamcorper, 43680) (prinis, 21840) (malesuada, 50960) (ei, 101910) (proin, 36390) It took 8624ms to finish calculation.</pre>	<pre>(cras, 50930) (parturient, 7280) (quisque, 43670) (massa, 36390) (fames, 14560) (lobortis, 36370) (ipsam, 109190) (suscipit, 50930) (sen, 65500) (molestie, 21840) (quam, 80050) It took 7936ms to finish calculation.</pre>	<pre>(rutrum, 58220) (nostra, 7270) (congue, 21840) (gravida, 29120) (voluptat, 58210) (magnis, 7280) (id, 116450) (sociis, 7280) (diam, 21830) (netus, 14560) (fermentum, 29110) It took 7163ms to finish calculation.</pre>	<pre>Finished release [optimized] target(s) in 0.0 Running 'target/release/count_words_operator' -h hostname.txt -n 16 -p 11' (vel, 65520) (monumy, 29120) (dapibus, 36390) (mauris, 80050) (ligula, 109150) (fucae, 94590) (erat, 43660) It took 6477ms to finish calculation.</pre>
<pre>(nunc, 160130) (odio, 65400) (lectus, 21830) (pulvinar, 43670) (inceptos, 7270) (tristique, 36380) (curae, 21840) (laculis, 58230) (rhonus, 36390) (vulputate, 36400) (cum, 7280) It took 13537ms to finish calculation.</pre>	<pre>-h hostname.txt -n 16 -p 5' (laoreet, 43670) (lorem, 58230) (vitae, 87320) (neque, 65510) (celit, 58200) (et, 107420) (orci, 94600) (ultrices, 36400) (ornare, 7280) (torquent, 7270) It took 12790ms to finish calculation.</pre>	<pre>(varius, 58210) (etiam, 65510) (condimentum, 50940) (sollicitudin, 21840) (dolor, 101880) (porta, 21830) (fo, 170690) (senectus, 14560) (sociosqu, 7270) (nec, 138310) (mollis, 80060) It took 11059ms to finish calculation.</pre>	<pre>(potenti, 7270) (nisi, 87330) (urna, 43660) (augue, 80080) (sit, 94620) (morbi, 50950) (dictumst, 14560) (sed, 152830) (at, 72770) (adipiscing, 65490) (lobendum, 14560) It took 10159ms to finish calculation.</pre>
<pre>(aenean, 65520) (maecenas, 43670) (accusamus, 36390) (placerat, 36390) (habitasse, 14560) (pellentesque, 116440) (nulla, 87330) (vivamus, 50910) (natoque, 7280) (ridiculus, 7280) (tincidunt, 87340) It took 8524ms to finish calculation.</pre>	<pre>Finished release [optimized] target(s) in 0.0 Running 'target/release/count_words_operator' -h hostname.txt -n 16 -p 13' (egestas, 101870) (euismod, 43660) (nisi, 87330) (fusipat, 50960) (fringilla, 43680) (commodo, 36370) (suspendisse, 58210) It took 4798ms to finish calculation.</pre>	<pre>(arcu, 80040) (consectetur, 43670) (tempor, 21840) (class, 7270) (sagittis, 72770) (apert, 7270) (risus, 29120) (facilisi, 14560) (donec, 94620) (dis, 7280) (hymenaeos, 7270) It took 4080ms to finish calculation.</pre>	<pre>(cursus, 43670) (aliquan, 80020) (eatlis, 29110) (quis, 152860) (dui, 65520) (ante, 87350) (luctus, 36400) (curpis, 109170) (non, 94600) (magna, 50960) (velit, 72790) It took 2193ms to finish calculation.</pre>

Slika 15: Izlaz programa za Timely Dataflow koji je pokrenut na grozdu

Ukupno trajanje: 3.193 sekundi.



Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

```

17
18 ► fn main() {
19     let time : Instant = Instant::now();
20
21     // initializes and runs a timely dataflow.
22
23     timely::execute_from_args( iter: std::env::args(), func: |worker| &mut Worker<Generic> | {
24
25         // create input and output handles.
26         let mut lines_input : Handle<usize, String> = InputHandle::new();
27         let mut probe : Handle<usize> = ProbeHandle::new();
28
29         // create a new input, exchange data, and inspect its output
30         worker.dataflow::<usize, _, _>(|scope : &mut Child<Worker<Generic>, usize> | {
31             lines_input.to_stream(scope) : Stream<Child<...>, String>
32             // map line from text to words.
33             .flat_map(|text: String| text.to_lowercase() : String
34                 .split(|c : char| !c.is_alphabetic()) : impl Iterator<Item=&str>
35                 .filter(|word : &&str| word.is_empty().not()) : impl Iterator<Item=&str>
36                 .map(move |word : &str| (word.to_owned(), 1)) : impl Iterator<Item=(...)>
37                 .collect::<Vec<_>>())
38             ) : Stream<Child<...>, (...)>
39             .exchange(|(word : &String, _)| calculate_hash( t: word)) : Stream<Child<...>, (...)>
40             .aggregate(
41                 fold: |_key, val : i32, agg : &mut? | { *agg += val; },
42                 emit: |key : String, agg: i32| (key, agg),
43                 hash: |key : &String| calculate_hash( t: key)
44             ).inspect(|x : &(String, i32)| println!("{}", x.0, x.1)) : Stream<Child<...>, (...)>
45             .probe_with( handle: &mut probe);
46
47         });
48     });

```

Slika 16: Primjer koda koji se pokrenuo na grozdu računala

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## Operator za filtriranje

Izvođenje preko Apache Flink – trenutno se ne pokreće na grozdu

Izvođenje preko Timely Dataflow

```
Distance: 3.79
Distance: 5.2
Distance: 9.35
Distance: 2.13
Distance: 1.26
Distance: 3.69
Distance: 5.63
Distance: 2.22
Time elapsed in milliseconds: 94851

real    1m34.874s
user    0m1.680s
sys     0m0.871s
timely@worker01:~/timely-filter$ |
```

Slika 17: Filtriranje vrijednosti koje su veće od 1 korištenjem jednog workera

Ukupno trajanje: 94.851 sekundi.

```
Distance: 0.75
Distance: 0.89
Distance: 0.54
Distance: 0.91
Distance: 0.76
Distance: 0.65
Distance: 0.88
Distance: 0.99
Time elapsed in milliseconds: 10264

real    0m12.466s
user    0m6.576s
sys     0m0.407s
timely@worker01:~/timely-filter$ |
```

Slika 18: Filtriranje vrijednosti između 0.5 i 1 korištenjem jednog workera

Ukupno trajanje: 10.264 sekundi.

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

```

Distance: 2.46
Distance: 1.86
Distance: 2.9
Distance: 3.46
Distance: 3.26
Distance: 1.81
Distance: 2.46
Distance: 1.76
Time elapsed in milliseconds: 78300
real 1m18.321s
user 0m2.703s
sys 0m0.306s
pp2021@manager:~$ []

Distance: 1.32
Distance: 1.65
Distance: 1.8
Distance: 1.44
Distance: 5.21
Distance: 1.34
Distance: 5.85
Time elapsed in milliseconds: 57170
real 0m9.344s
user 0m5.893s
sys 0m0.370s
pp2021@manager:~$ []

Distance: 3.16
Distance: 2.88
Distance: 4.16
Distance: 1.7
Distance: 12.04
Distance: 1.05
Distance: 9.26
Distance: 4.28
Time elapsed in milliseconds: 39442
real 0m41.031s
user 0m5.905s
sys 0m0.132s
pp2021@manager:~$ []

Distance: 17.61
Distance: 3.63
Distance: 1.03
Distance: 2.57
Distance: 17.68
Distance: 1.4
Distance: 2.49
Time elapsed in milliseconds: 17905
real 0m9.088s
user 0m5.913s
sys 0m0.432s
pp2021@manager:~$ []

Distance: 1.61
Distance: 2.11
Distance: 2.86
Distance: 6.89
Distance: 2.86
Distance: 1.71
Distance: 4.51
Distance: 4.78
Time elapsed in milliseconds: 71116
real 1m13.300s
user 0m5.137s
sys 0m0.306s
pp2021@manager:~$ []

Distance: 3.04
Distance: 1.53
Distance: 1.55
Distance: 4.69
Distance: 3.15
Distance: 3.79
Distance: 1.36
Time elapsed in milliseconds: 52579
real 0m54.751s
user 0m5.846s
sys 0m0.351s
pp2021@manager:~$ []

Distance: 5.15
Distance: 1.14
Distance: 1.94
Distance: 1.14
Distance: 2.73
Distance: 1.14
Distance: 3.69
Distance: 5.63
Time elapsed in milliseconds: 32883
real 0m35.000s
user 0m5.835s
sys 0m0.181s
pp2021@manager:~$ []

Distance: 1.45
Distance: 1.27
Distance: 2.02
Distance: 3.85
Distance: 2.8
Distance: 6.17
Distance: 5.2
Time elapsed in milliseconds: 12692
real 0m14.088s
user 0m5.825s
sys 0m0.440s
pp2021@manager:~$ []

Distance: 2.2
Distance: 1.63
Distance: 6.39
Distance: 3.34
Distance: 1.69
Distance: 3.1
Distance: 1.69
Distance: 3.12
Time elapsed in milliseconds: 86228
real 1m8.388s
user 0m5.859s
sys 0m0.306s
pp2021@manager:~$ []

Distance: 4.79
Distance: 2.03
Distance: 1.48
Distance: 3.12
Distance: 6.61
Distance: 3.48
Distance: 1.39
Time elapsed in milliseconds: 48412
real 0m58.592s
user 0m5.808s
sys 0m0.351s
pp2021@manager:~$ []

Distance: 3.68
Distance: 3.77
Distance: 1.3
Distance: 3.77
Distance: 2.35
Distance: 1.37
Distance: 2.46
Distance: 9.35
Time elapsed in milliseconds: 27012
real 0m29.280s
user 0m5.845s
sys 0m0.343s
pp2021@manager:~$ []

Distance: 2.13
Distance: 2.85
Distance: 7.49
Distance: 1.35
Distance: 6.75
Distance: 3.8
Distance: 2.13
Time elapsed in milliseconds: 7519
real 0m7.785s
user 0m5.828s
sys 0m0.363s
pp2021@manager:~$ []

Distance: 3.27
Distance: 2.14
Distance: 2.58
Distance: 3.87
Distance: 1.79
Distance: 1.29
Distance: 1.21
Distance: 1.56
Time elapsed in milliseconds: 61361
real 1m3.552s
user 0m5.926s
sys 0m0.384s
pp2021@manager:~$ []

Distance: 1.5
Distance: 1.12
Distance: 1.25
Distance: 1.59
Distance: 3.28
Distance: 1.84
Distance: 1.87
Time elapsed in milliseconds: 44009
real 0m46.256s
user 0m5.924s
sys 0m0.357s
pp2021@manager:~$ []

Distance: 3.37
Distance: 3.82
Distance: 2.75
Distance: 4.32
Distance: 1.23
Distance: 3.37
Distance: 2.75
Distance: 18.66
Time elapsed in milliseconds: 22459
real 0m24.628s
user 0m5.787s
sys 0m0.329s
pp2021@manager:~$ []

Distance: 1.72
Distance: 1.66
Distance: 10.3
Distance: 1.64
Distance: 10.91
Distance: 1.64
Distance: 2.22
Time elapsed in milliseconds: 2430
real 0m4.601s
user 0m5.825s
sys 0m0.422s
pp2021@manager:~$ |

```

Slika 19: Filtriranje vrijednosti koje su veće od 1 korištenjem 16 workera

Ukupno trajanje: 2.430 sekundi.

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

```

Distance: 0.61
Distance: 0.6
Distance: 0.66
Distance: 0.61
Distance: 0.76
Distance: 0.66
Distance: 0.76
Distance: 0.76
Time elapsed in milliseconds: 67208

real 1m7.320s
user 0m2.741s
sys 0m0.040s
pp2021@manager:~$ []

Distance: 0.52
Distance: 0.91
Distance: 0.52
Distance: 0.81
Distance: 0.52
Distance: 0.91
Distance: 0.91
Time elapsed in milliseconds: 49047

real 0m0.971s
user 0m0.599s
sys 0m0.115s
pp2021@manager:~$ []

Distance: 0.96
Distance: 0.96
Distance: 0.96
Distance: 0.96
Distance: 0.96
Distance: 0.96
Distance: 0.96
Time elapsed in milliseconds: 31284

real 0m31.316s
user 0m0.586s
sys 0m0.093s
pp2021@manager:~$ []

Distance: 0.78
Distance: 0.92
Distance: 0.92
Distance: 0.59
Distance: 0.89
Distance: 0.89
Distance: 0.89
Time elapsed in milliseconds: 14802

real 0m14.809s
user 0m0.730s
sys 0m0.146s
pp2021@manager:~$ []

Distance: 0.88
Distance: 0.84
Distance: 0.84
Distance: 0.58
Distance: 0.58
Distance: 0.84
Distance: 0.84
Time elapsed in milliseconds: 62812

real 1m2.843s
user 0m0.607s
sys 0m0.092s
pp2021@manager:~$ []

Distance: 0.99
Distance: 0.99
Distance: 0.72
Distance: 0.72
Distance: 0.99
Distance: 0.72
Distance: 0.99
Time elapsed in milliseconds: 45490

real 0m45.523s
user 0m0.628s
sys 0m0.097s
pp2021@manager:~$ []

Distance: 0.77
Distance: 0.74
Distance: 0.73
Distance: 0.77
Distance: 0.73
Distance: 0.77
Distance: 0.77
Time elapsed in milliseconds: 26932

real 0m26.968s
user 0m0.685s
sys 0m0.139s
pp2021@manager:~$ []

Distance: 0.85
Distance: 0.71
Distance: 0.57
Distance: 0.57
Distance: 0.85
Distance: 0.71
Distance: 0.85
Time elapsed in milliseconds: 10080

real 0m10.138s
user 0m0.615s
sys 0m0.101s
pp2021@manager:~$ []

Distance: 0.75
Distance: 0.75
Distance: 0.75
Distance: 0.75
Distance: 0.51
Distance: 0.51
Distance: 0.75
Time elapsed in milliseconds: 58404

real 0m58.427s
user 0m0.561s
sys 0m0.081s
pp2021@manager:~$ []

Distance: 0.85
Distance: 0.85
Distance: 0.95
Distance: 0.95
Distance: 0.85
Distance: 0.95
Distance: 0.95
Time elapsed in milliseconds: 41522

real 0m41.559s
user 0m0.574s
sys 0m0.111s
pp2021@manager:~$ []

Distance: 0.87
Distance: 0.54
Distance: 0.54
Distance: 0.87
Distance: 0.79
Distance: 0.87
Distance: 0.54
Time elapsed in milliseconds: 36292

real 0m36.353s
user 0m0.677s
sys 0m0.095s
pp2021@manager:~$ []

Distance: 0.62
Distance: 0.62
Distance: 0.62
Distance: 0.8
Distance: 0.8
Distance: 0.62
Distance: 0.62
Time elapsed in milliseconds: 22474

real 0m22.516s
user 0m0.628s
sys 0m0.072s
pp2021@manager:~$ []

Distance: 0.56
Distance: 0.56
Distance: 0.56
Distance: 0.45
Distance: 0.65
Distance: 0.56
Distance: 0.45
Time elapsed in milliseconds: 17952

real 0m17.075s
user 0m0.557s
sys 0m0.089s
pp2021@manager:~$ []

Distance: 0.93
Distance: 0.53
Distance: 0.53
Distance: 0.53
Distance: 0.93
Distance: 0.93
Distance: 0.93
Time elapsed in milliseconds: 6440

real 0m0.477s
user 0m0.558s
sys 0m0.115s
pp2021@manager:~$ []

Distance: 0.98
Distance: 0.83
Distance: 0.98
Distance: 0.98
Distance: 0.98
Distance: 0.98
Distance: 0.98
Time elapsed in milliseconds: 2470

real 0m2.508s
user 0m0.603s
sys 0m0.098s
pp2021@manager:~$ []

```

Slika 20: Filtriranje vrijednosti između 0.5 i 1 korištenjem 16 workera

Ukupno trajanje: 2.470 sekundi.

Usporedba performansi obrade tokova podataka između Apache Flink i Timely Dataflow	Verzija: <1.0>
Tehnička dokumentacija	Datum: <21/01/2022>

## 1. Literatura

- 
- [1] <https://flink.apache.org/flink-architecture.html>
  - [2] <https://flink.apache.org/stateful-functions.html>
  - [3] <https://timelydataflow.github.io/timely-dataflow/>
  - [4] <https://docs.rs/timely/latest/timely/>
  - [5] <https://doc.rust-lang.org/book/foreword.html>
  - [6] <https://doc.rust-lang.org/rust-by-example/>
  - [7] <https://nightlies.apache.org/flink/flink-docs-release-1.0/apis/batch/examples.html>
  - [8] <https://github.com/TimelyDataflow/timely-dataflow>