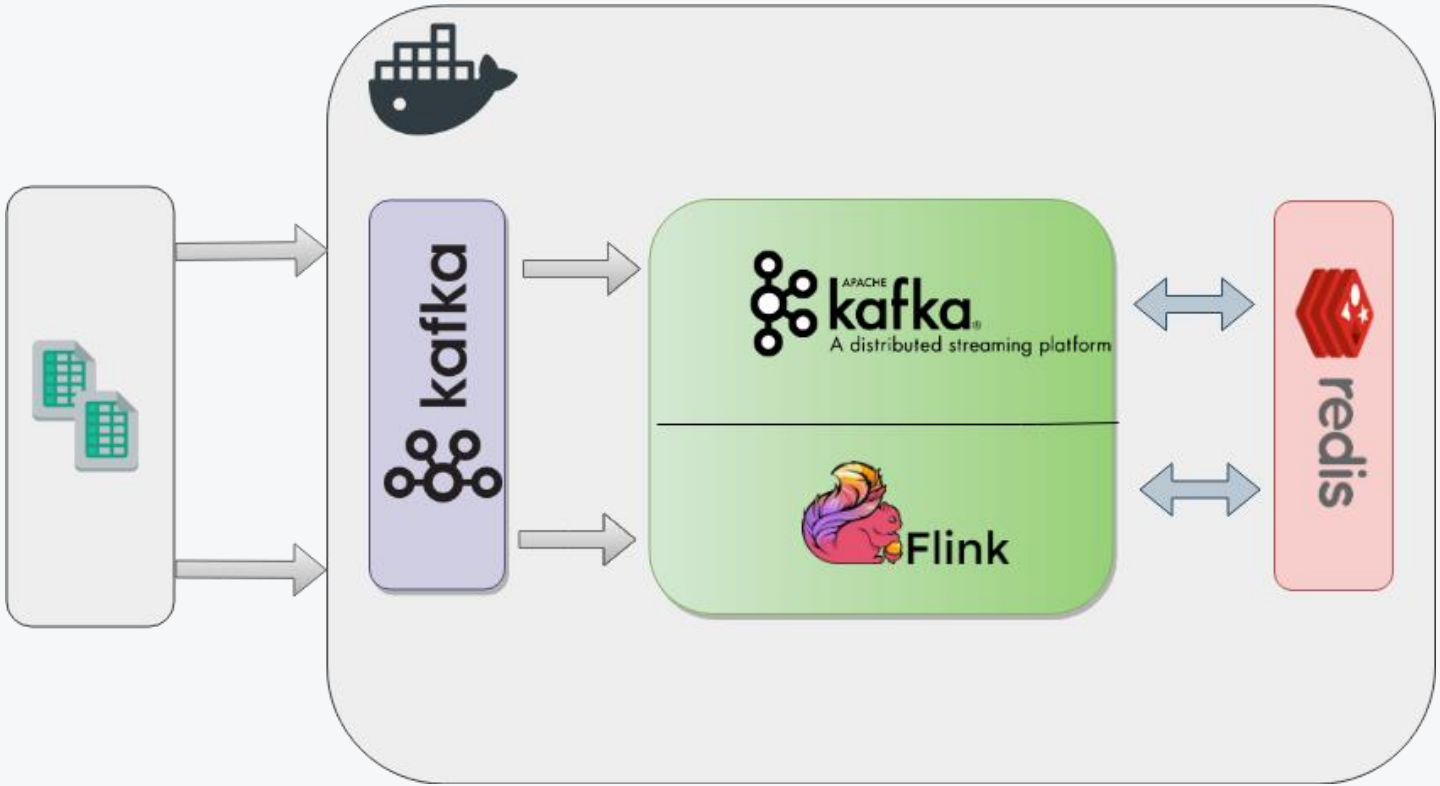
Progetto Stream processing

SABD 2018-2019

Montesano,Perrone,Pusceddu

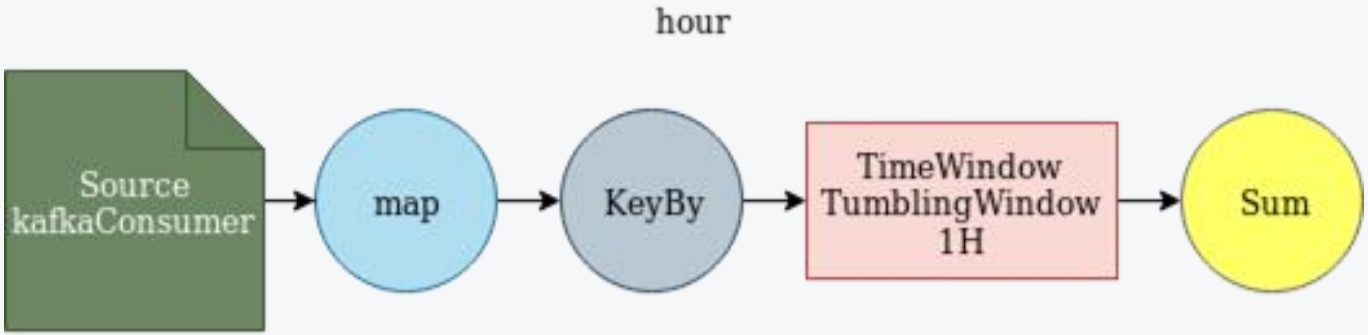
Architettura



Configurazione

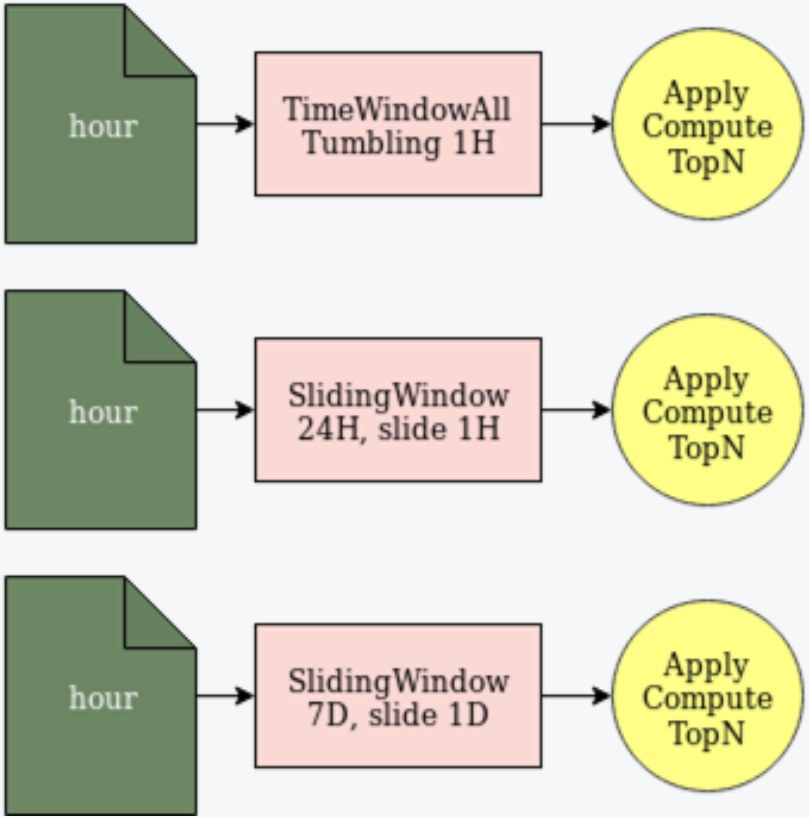
- Kafka Producer: configurazione dei serializzatori per chiave e valore. La chiave viene serializzata come Long, mentre il valore ha un deserializzatore custom che rappresenta una riga del dataset, chiamata Post.
- L'invio delle tuple è stato accelerato: a partire dalla createDate del primo record viene scelto il *rate* (createDate - currentTime), dopodiché viene scalato di un fattore 10^5
- Flink Kafka Consumer: riceve un Post.
- KafkaStreams: le properties di configurazione cambiano per ogni query, poiché è necessario cambiare il serializzatore di default del valore per Kafka. Per la prima query il serializzatore del valore è ByteArray, per la seconda e terza query è Integer, mentre il serializzatore di chiave è sempre String. Inoltre ad ogni operazione di groupBy e map è necessario specificare che serializzatore usare.

Query 1



- Map: si prendono solamente i campi di interesse del post:
  - (ArticleId, 1).
- KeyBy: si raggruppano le tuple per id dell'articolo.
- TumblingWindow: si aggregano tutte le tuple su finestra di 1H.
- Sum: somma delle occorrenze degli articoli in 1H.

Query 1



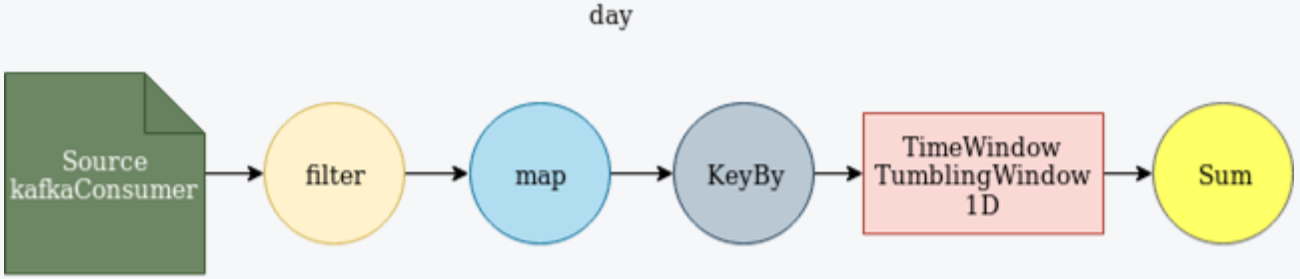
windowAll: si riaggregano le tuple in base alla grandezza della finestra.

Per finestre di 24H e 7D sono state usate sliding windows: nel primo caso l’intervallo di scorrimento è di 1H, nel secondo caso è di 1D.

TopN function

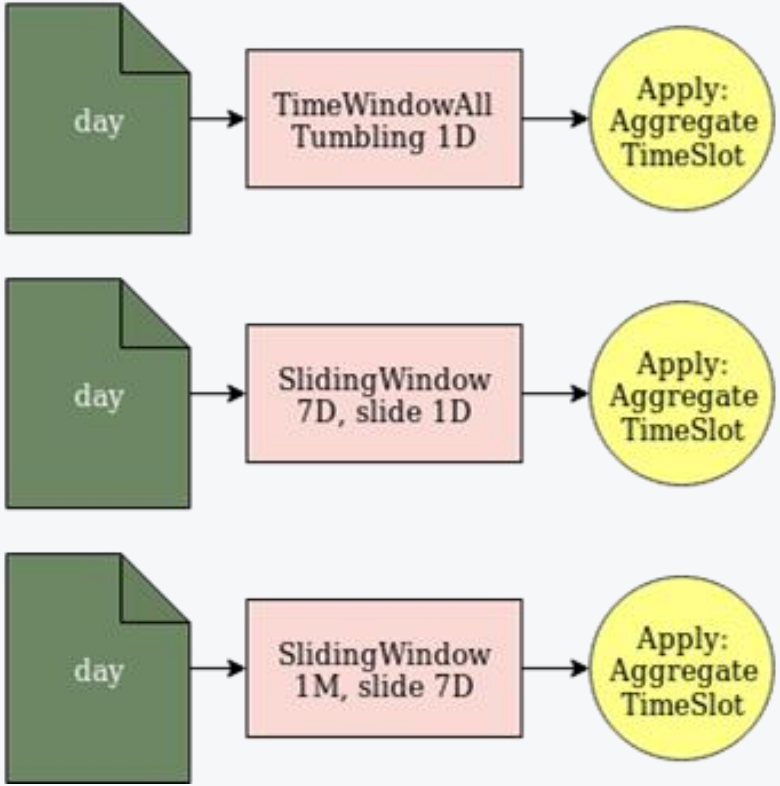
- Le funzioni TopN Implementano AllWindowFunction, aggregando tutte le tuple nella finestra.
- Input: lista di Tuple2 composta da id dell’articolo e le sue occorrenze nella finestra.
- Si ordinano gli elementi in base alle occorrenze in senso decrescente e si prendono le prime tre tuple.
- Si utilizza un hash map per aggregare 24 finestre da 1 ora.
- Si utilizza lo stesso metodo per la finestra da 7 giorni.

Query 2



- Filter: si prendono solamente i commenti diretti (depth == 1).
- Map: si assegna ad ogni commento, una chiave in base alla fascia oraria di creazione.
- KeyBy: si raggruppano le tuple per chiave.
- TumblingWindow: si aggregano tutte le tuple su finestra di 24H.
- Sum: somma delle occorrenze dei commenti in 24H per fascia oraria.

Query 2



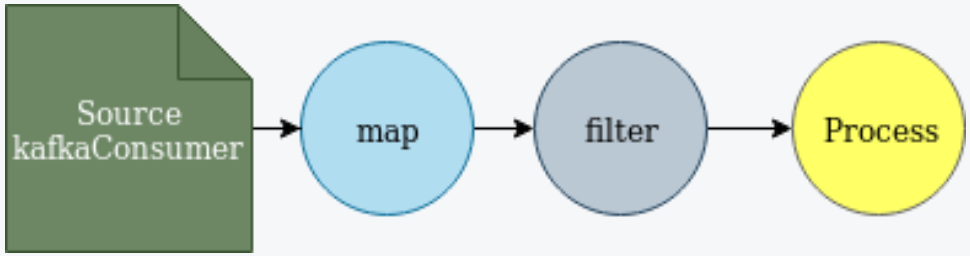
windowAll: si riaggregano le tuple in base alla grandezza della finestra.

Si utilizza l'apply per contare i commenti in ogni fascia oraria su base finestra: la prima con intervallo di scorrimento di 1D, la seconda con intervallo di 7D.

Aggregate function

- Le funzioni aggregate Implementano AllWindowFunction, aggregando tutte le tuple nella finestra.
- Input: lista di Tuple2 composta da un intero che rappresenta la fascia oraria e le occorrenze dei commenti in quella fascia oraria nella finestra.
- Si ritorna il numero complessivo di commenti in ogni fascia oraria.
- Si utilizza un hash map per aggregare 7 finestre da 24 ore.
- Si utilizza lo stesso metodo per la finestra da 1 mese.

Query 3



- Map: si prendono solamente i campi di interesse del post
  - (UserId, Depth, Like, InReplyTo, CommentID)
- Filter: si eliminano tutte le tuple malformate.

Ranking process function

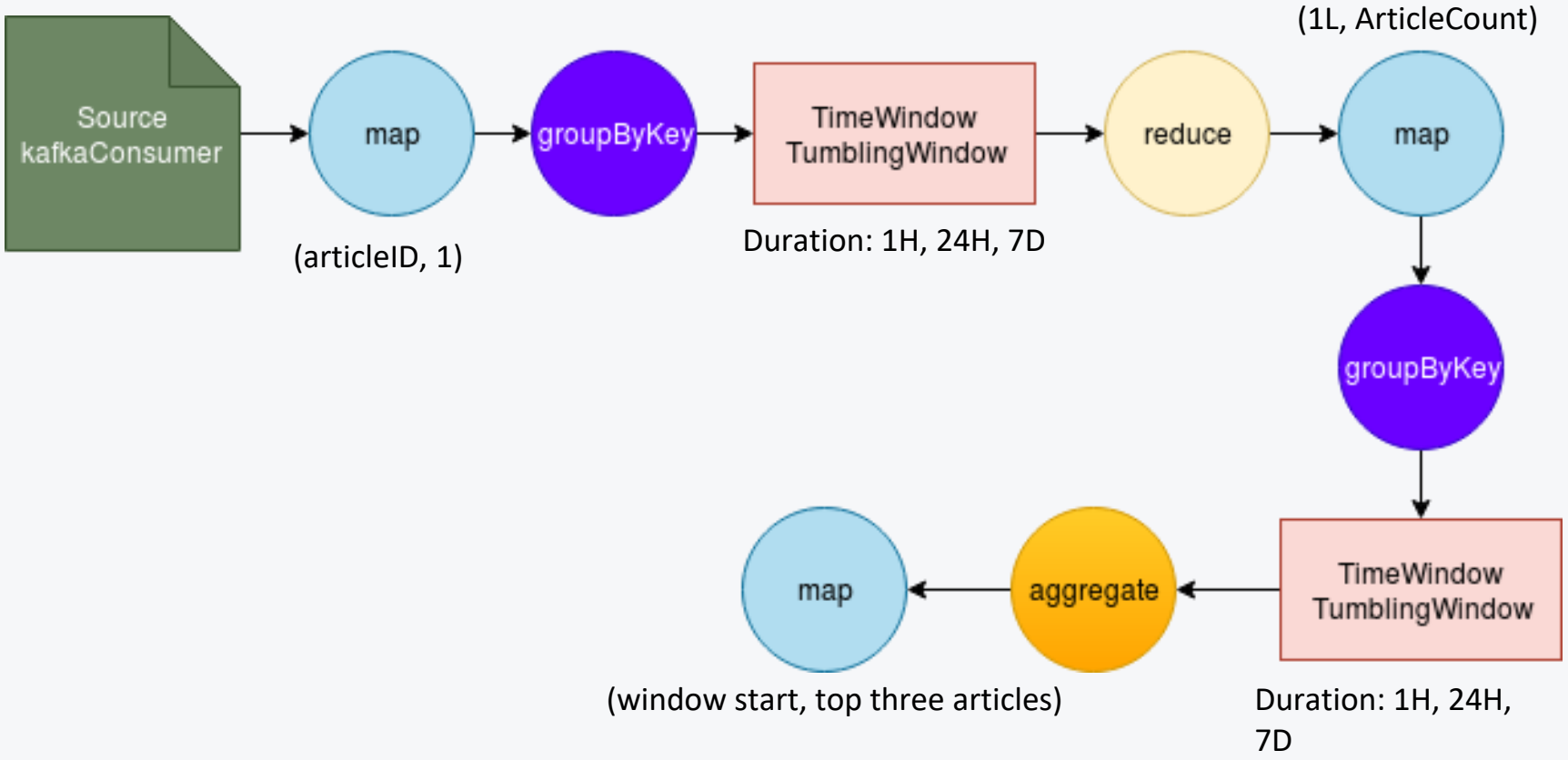
- Operatore con stato, composto da 3 HashMap:
  - <IdUtente,Score> : la classe Score mantiene il punteggio di ogni utente.
  - <IdCommentoLvL1,IdUtente> : mantiene il riferimento all'utente che ha scritto un commento diretto.
  - <IdCommentoLvL2,IdCommentoLvL1> : mantiene il riferimento alla risposta ad un commento diretto
- In base al livello di un commento in arrivo, viene aggiornato lo stato (una o più delle tre hashmap) nel seguente modo:
  - per tutte le risposte di livello 2, si incrementa il count dello score dell'utente.
  - per tutte le risposte di livello 3, si risale all'utente del commento diretto e si aumenta il count dello score.
- Alla fine della finestra, si ordina l'hashmap in base allo score e si azzera lo stato locale, mantenendolo solamente su Redis.

Redis

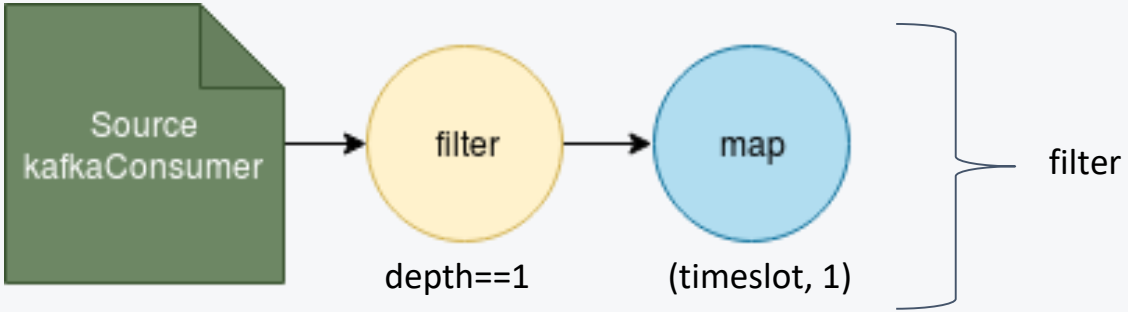
- L’utilizzo di Redis è dovuto al problema di dover mantenere i riferimenti dei commenti delle finestre precedenti poiché è possibile ricevere un nuovo commento di secondo o terzo livello che si riferisce ad un commento rispettivamente di primo o secondo livello di una delle finestre precedenti.

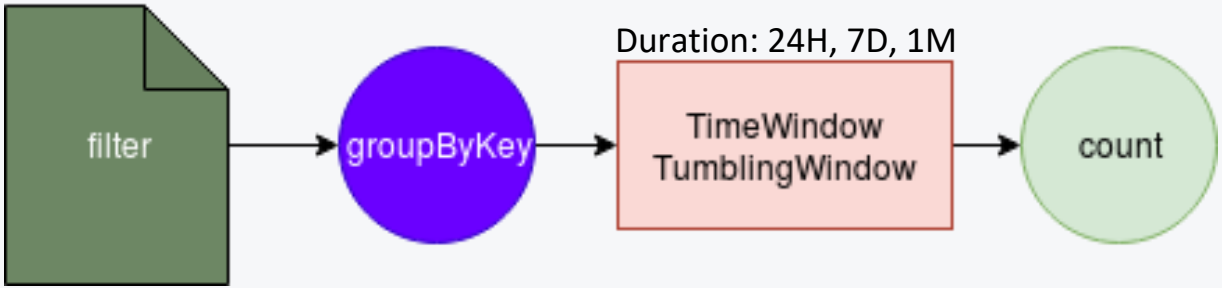
- Strutture dati mantenute in Redis:
  - <IdCommentoLvL1,IdUtente> : mantiene il riferimento all’utente che ha scritto un commento diretto.
  - <IdCommentoLvL2,IdCommentoLvL1> : mantiene il riferimento alla risposta ad un commento diretto
- Quindi durante l’arrivo di una nuova tupla si cerca l’id del commento nelle strutture locali, se non presente viene aggiunto alle strutture locali e si cerca in redis.
- Le strutture di Redis non vengono mai azzerate.

Query 1 - Kafka Streams

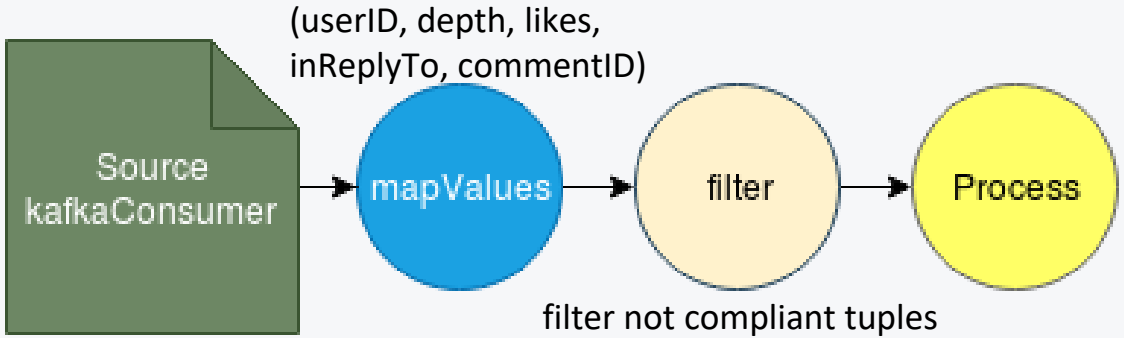


Query 2 - Kafka Streams





Query 3 - Kafka Streams

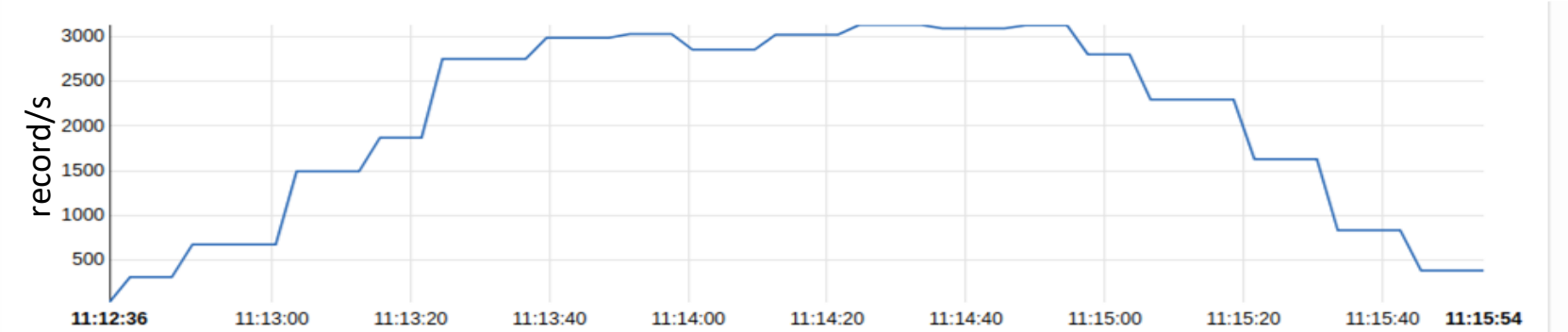


- La funzione Process è la stessa della Query3 spiegata precedentemente.
- Per questa query è stato creato uno Store Builder per memorizzare lo stato localmente, fornendogli i serializzatori e deserializzatori.

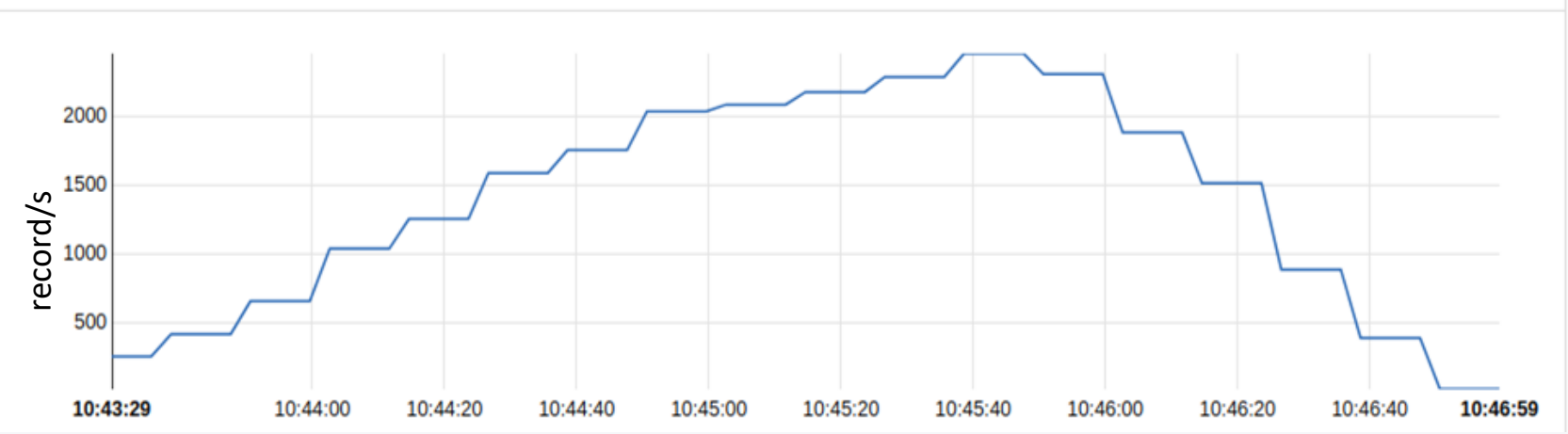
Flink: Throughput e latenza

- Throughput ottenuto tramite le web ui.
- Latenza ottenuta misurando il tempo di attraversamento di un dato operatore.Sono stati considerati solamente gli operatori prima delle finestre temporali.

Query1 throughput :



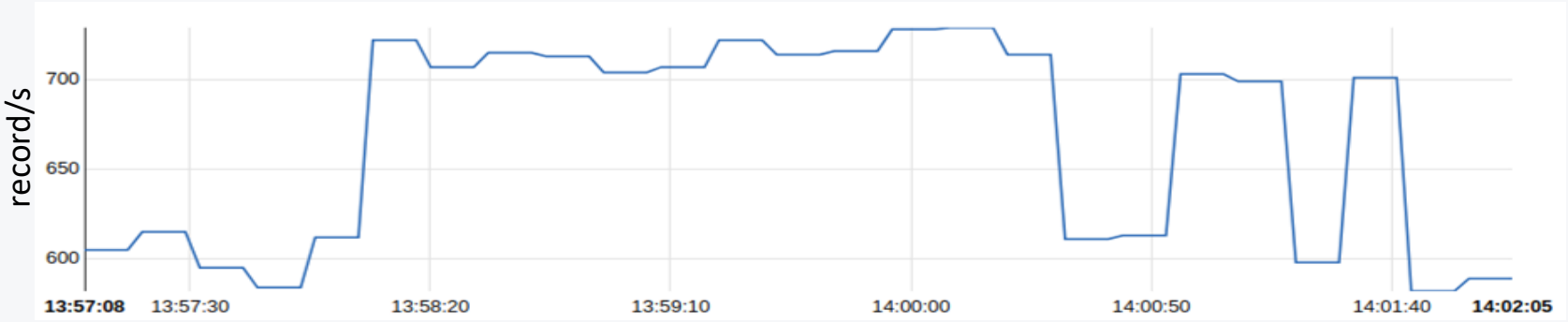
Flink: Throughput - Query2



Flink: Throughput - Query3

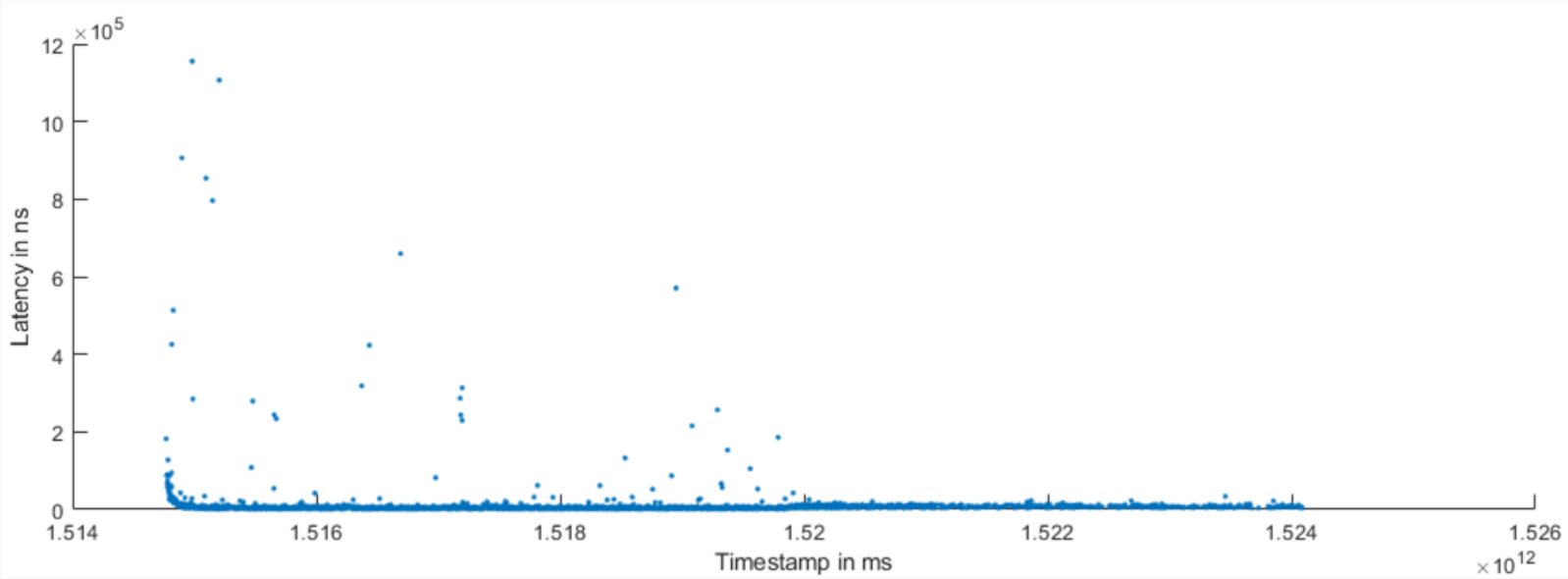
(parte1)

(parte2)



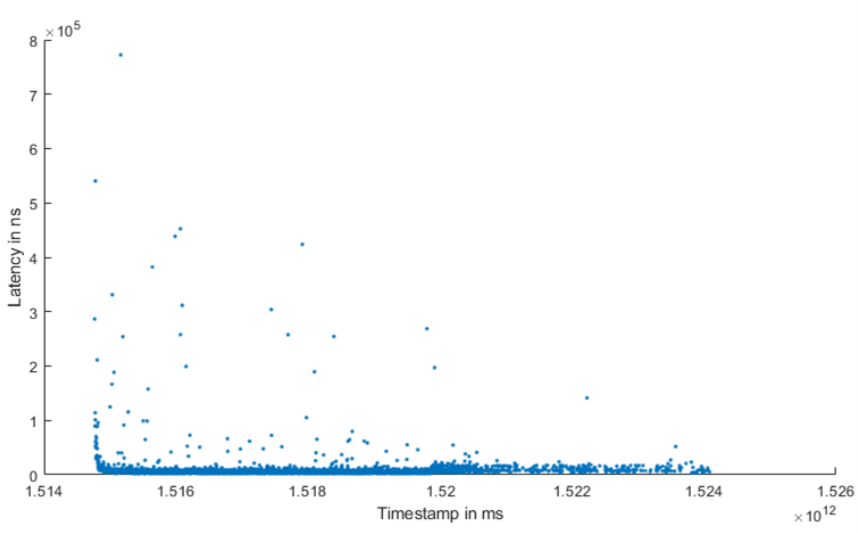
Flink: Latenza - Query 1

Map operator

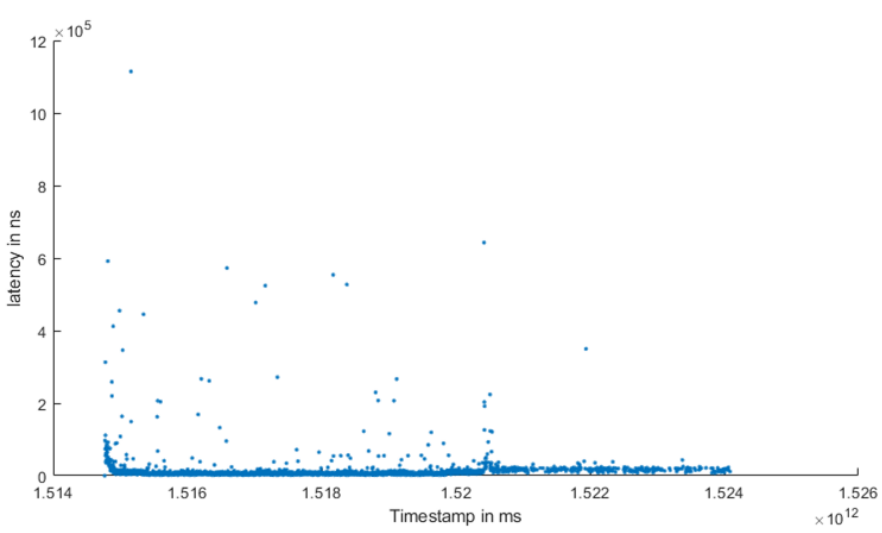


Latenza - Query 2

Filter operator

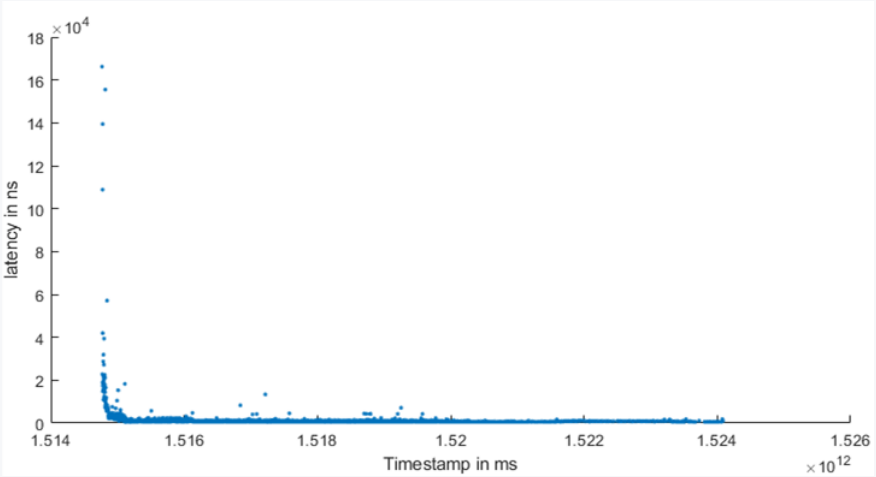


Map operator

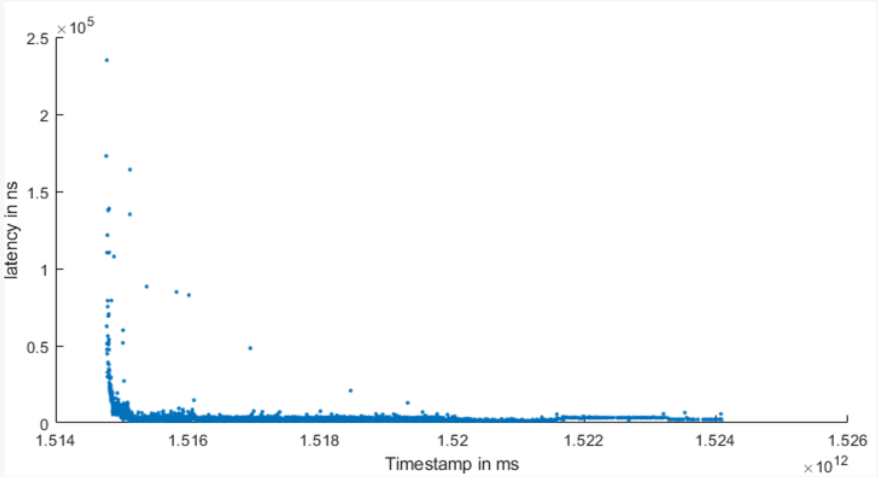


Latenza - Query 3

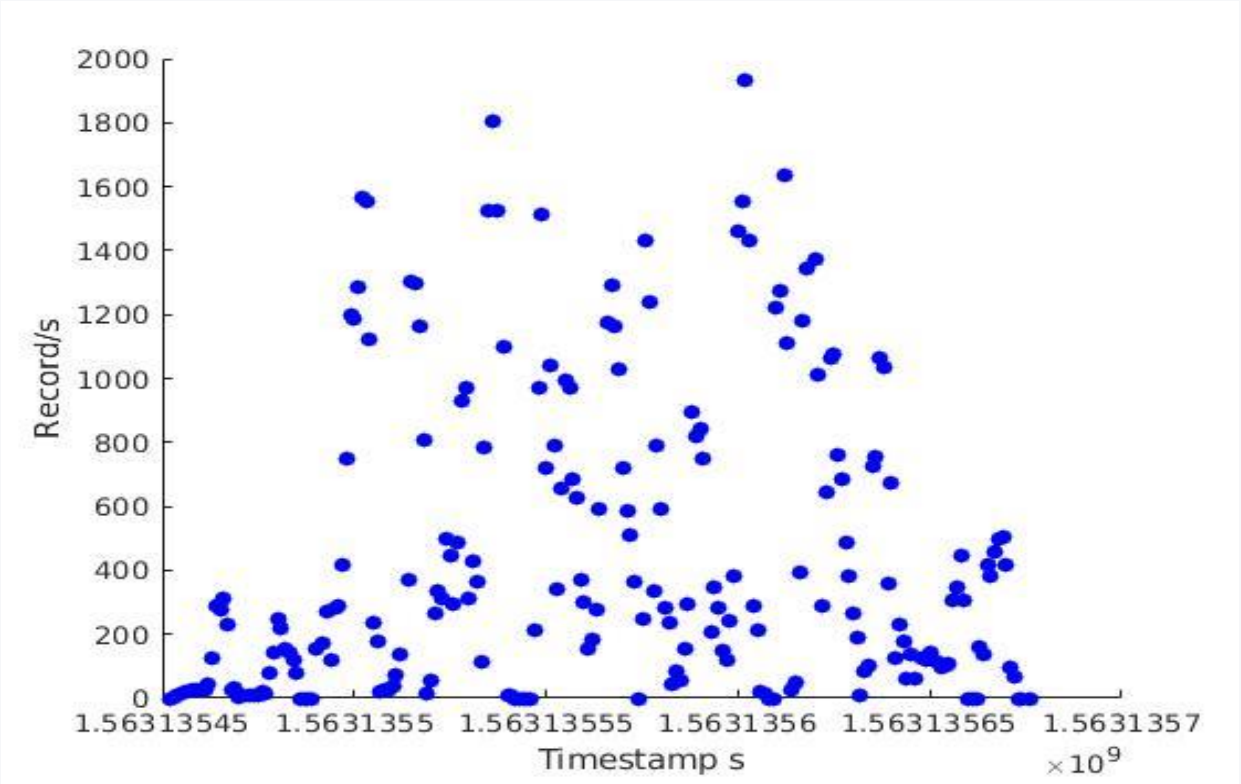
Filter operator



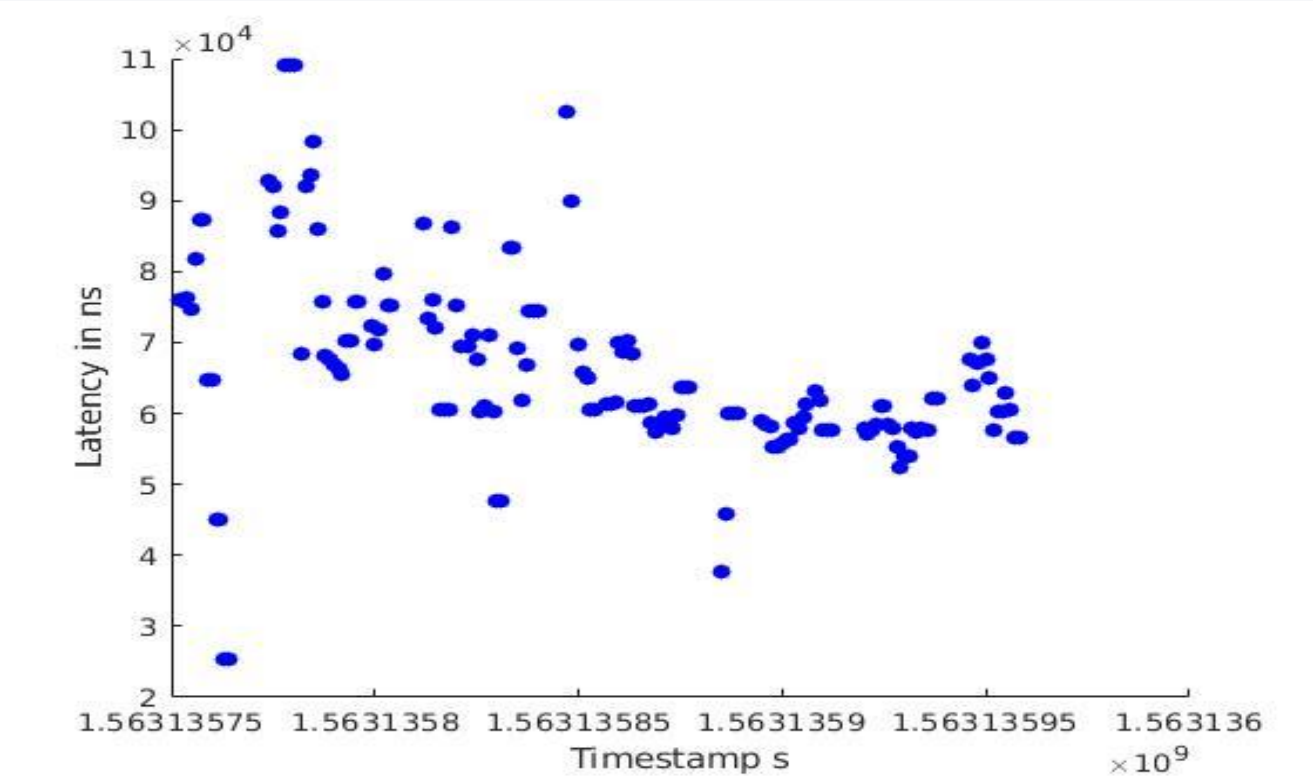
Map operator



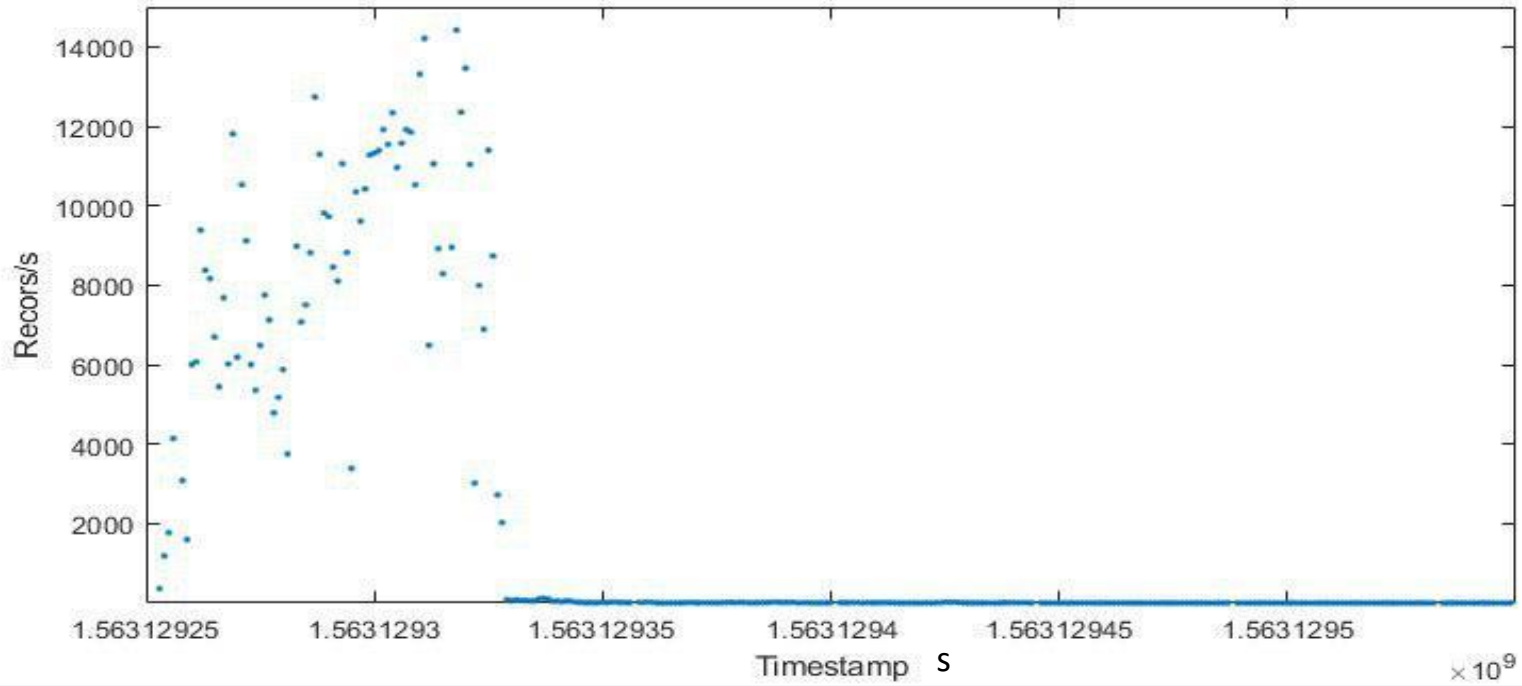
Kafka Stream: Throughput - Query 1



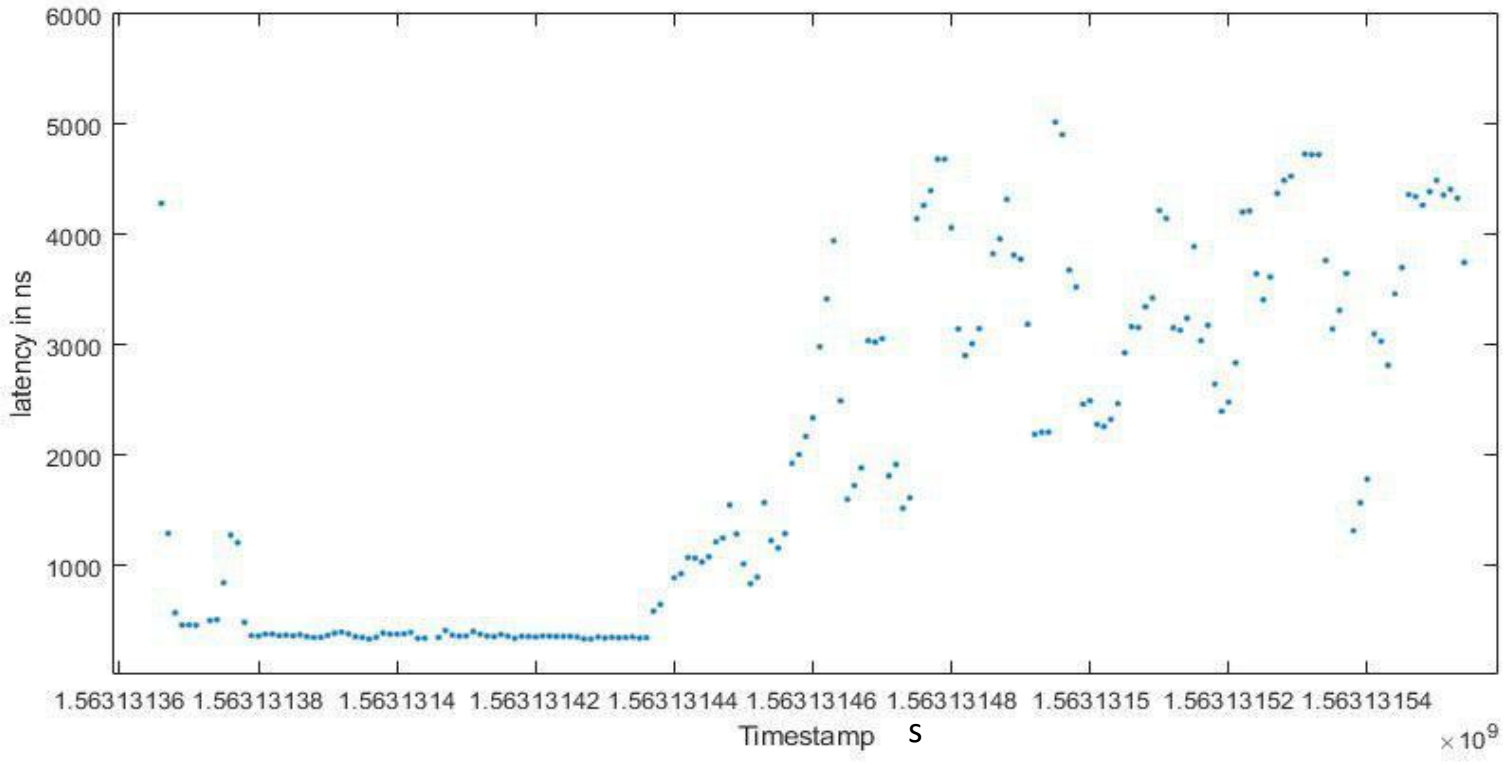
Kafka Stream:
Latenza - Query 1



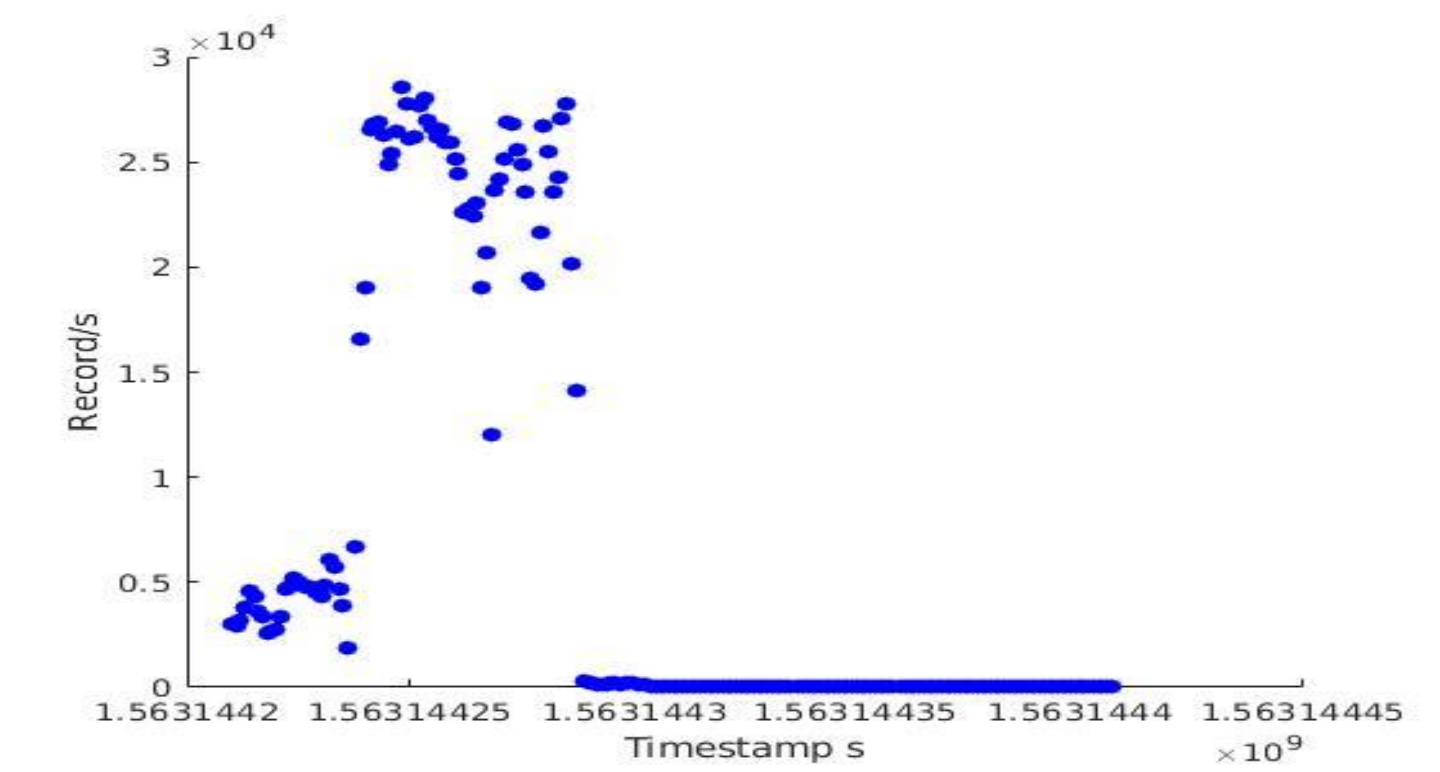
Kafka Stream: Throughput - Query 2



Kafka Stream:
Latenza - Query 2



Kafka Streams: Throughput - Query 3



Kafka Stream:
Latenza - Query 3