Flink Stream Processing

Progetto di Sistemi e Architetture per Big Data

Matteo Federico
0321569

Elisa Verza
0311317

***Abstract*—Questo documento si pone lo scopo di spiegare il metodo adoperato e lo stack architetturale utilizzato per rispondere alle tre query assegnate per il secondo progetto del corso di "Sistemi e Architetture per Big Data" della facoltà di Ingegneria Informatica magistrale dell'università di Roma Tor Vergata.**

I. Introduzione

Il progetto consiste nello svolgimento di tre query su dati, elaborati tramite processamento stream, relativi ad azioni scambiate sui mercati di Parigi, Francoforte\Xetra e Amsterdam nel periodo 08/11/2021 - 14/11/2021.
Il lavoro è stato diviso in cinque fasi:

- Creazione di un producer per i dati streaming
- Progettazione dell'architettura
- Preprocessing dei dati
- Analisi e svolgimento delle query
- Analisi delle performance.

I framework utilizzati sono:

- Apache Flink per il processamento stream
- Apache Kafka come coda di messaggi per il data ingestion
- Apache Zookeeper come servizio di coordinazione distribuito operante insieme Kafka
- Apache Nifi per effettuare il pre-processing
- Apache Spark streaming per rispondere alla prima query e fornire un confronto prestazionale con Flink
- Prometheus come framework di monitoraggio
- Grafana come framework di visualizzazione delle prestazioni delle tre query L'intera architettura inoltre è stata sviluppata in modo distribuito su container Docker in un cluster locale utilizzando Docker Compose come orchestratore.

II. Architettura

Uno schema architetturale di alto livello è mostrato in 1. Il sistema è containerizzato ed ogni container espone le porte necessarie per poter accedere all'UI e permettere agli altri container di sfruttare il servizio offerto.
Ai fini del progetto è stato creato un producer che permettesse di leggere i dati da inviare in modo stream dal csv fornito. I dati vengono poi inviati ad un topic Kafka dal producer in modo da rispettare la proporzionalità temporale reale delle tuple. Il consumer del topic è Nifi che prende le tuple e applica un processamento iniziale per poi scriverle su un nuovo topic Kafka il cui consumer è direttamente Flink. Quest'utlimo procede al processamento dei dati sulle finestre specifiche richieste dalle query. Infine, invia i risultati su un topic Kafka il cui consumer é uno script python che preleva i risulati e li scrive su file csv.

III. Dataset

Per lo svolgimento delle query è stata asseganata una porzione di un dataset originariamente messo a disposizione per l'evento "Call for Grand Challenge Solution" di Debs del 2022. Il dataset contiene un insieme di dati finanziari dove ogni riga rappresenta uno scambio di strumenti finanziari, suddivisi in 39 campi. Quelli di maggior interesse, utili per rispondere alle query sono i seguenti:

- ID: una stringa formata da due parti separate da un punto, la prima parte rappresenta la sigla dello strumento finanziario, la seconda rappresenta il suo mercato di appartenenza.
- SecType: un campo che può assumere solo due valori: "E" (equities) e "I" (indices), e permette di differenziare le azioni dagli indici.
- Last: é il campo contenente il prezzo dell'ultima offerta per lo strumento finanziario.
- Trading Time e Trading Date: questi due campi sono presenti solo per le azioni, quindi per le entry con SecType posto ad "E", e rappresentano rispettivamente l'orario in cui occorre l'evento (in formato "HH:MM:ss.mmm") e la data (in formato "dd-mm-yyyy").
- Time e Date: si differenziano dai precedenti campi perchè rappresentano rispettivamente l'orario (in formato "HH:MM:ss.mmm") e la data (in formato "dd-mm-yyyy") utili a simulare i tempi reali di ricezione dell'evento.

IV. Producer

E' stato scritto in python e non viene containerizzato, ma eseguito localmente. Come prima operazione viene eliminato l'header del dataset che contiene le righe iniziali di commento che spiegano le finalità del dataset, una riga con l'effettivo header del file csv e un'ulteriore riga che va a dare una spiegazione di ogni colonna del dataset. A questo punto viene

letto tutto il file restante in un dataframe, viene creata una nuova colonna contenente i campi Date e Time concatenati in modo tale da poter ordinare temporalmente il dataset. Infine si crea un'altra colonna in cui viene specificato in millisecondi la differenza temporale tra due eventi consecutivi, il dataset viene salvato sul file.
Il file viene poi letto e vengono inviate al topic Kafka 'user' le tuple: in blocco quelle con differenza temporale di zero millisecondi, mentre dopo una sleep le altre. Il tempo è stato scalato in modo tale che un'ora di tempo simulato corrisponda a 4 secondi di tempo reale.
Sono state riscontrate problematiche nella gestione della grande quantità di dati nel producer, infatti si è reso necessario leggere ed effettuare tutte le operazioni possibili tramite lettura del file (iniziale eliminazione dell'header e sacansione finale degli eventi per l'invio a Kafka) mentre il passaggio iniziale da file a dataframe e finale da dataframe a file è stato effettuato per chunk.

V. NIFI

NIFI é stato utilizzato per il preprocessing delle tuple applicando una serie di filtraggi in modo tale da far arrivare a Flink solo i dati da processare e non le tuple senza payload o le colonne inutilizzate. In particolare è stato:

- Ridotto il numero di colonne, al fine di lasciare solo le informazioni necessarie (ovvero le 5 colonne descritte in precedenza)
- Eliminare le righe che avevano uno dei campi selezionati al punto prima vuoti o privi di informazione.

Con l'apposito connettore Nifi svolge il compito di consumer sul topic 'user' e da producer sul topic 'user2' dove scrive le tuple utili.
All'avvio del sistema è stata automatizzata la creazione e l'avvio del template Nifi. In particolare si sfruttano le api Rest che Nifi mette a disposizione per poter effettuare il collegamento tra un Token generato e un secure_cookie, inseguito si utilizza il token per caricare il gruppo di processori che si occupa del pre-processamento e avviarlo, per l'automatizzazione se ne occupa uno script bash che utilizzando il comando `curl` permette di inviare richeiste.
Nifi arrivava presto a saturazione, si suppone per la scala temporale utilizzata per l'invio dei dati, rappresentando quindi un collo di bottiglia, per questo motivo per l'esecuzione delle query e l'analisi delle preformance è stato escluso dalla pipeline lasciando a Flink il compito di filtrare gli eventi. Sono presenti entrambe le versioni delle query eseguibili in modo indipendente. Si suppone comunque che in un sistema in cui non viene velocizzato il tempo di arrivo degli eventi Nifi possa essere inserito nella pipeline senza rappresentare un rallentamento, lasciando quindi a Flink il solo compito di processmento delle finestre.
In figura 2 è possibile vedere lo schema dei processori utilizzati.

VI. CONSUMER

E' un altro script python eseguito localmente. Flink dopo aver processato una finestra scrive i risultati in un topic Kafka denominato con il seguente formato 'resultQuery*-tipoDiFinestra', quindi per ogni query è presente un consumer che legge da tre topic (uno per tipo di finestra) e scrive i risultati su un file csv. Nello scenario corrente, considerato che i dati a disposizione hanno un limite temporale, è stata inserita una condizione di uscita per il consumer che altrimenti alla fine dell'invio dei risultati sarebbe rimasto il attesa di nuovi messaggi. Il consumer quindi si chiude e effettua la close dei file contenenti i risultati dopo tre minuti in cui non vede arrivare altri messaggi su nessuno dei tre topic.

VII. IMPLEMENTAZIONE QUERY

Nel caso in cui si considerino i processamenti in cui Flink legge le tuple direttamente dal topic Kafka del producer python, si omette di specificare per ogni query la fase di pulizia dei dati che è comune a tutte e tre e viene effettuata in fase di lettura dei dati, quindi prima di suddividerli in finestre o di iniziare ad effettuare operazioni proprie delle query.
In questa prima fase vengono prese solo le colonne di interesse (con una map) e poi filtrati i dati (con una filter) che presentano stringa vuota come data e orario nullo, ovvero in formato 00:00:00.000. A questo punto si procede con la normale esecuzione delle query.

A. Query 1

L'obiettivo della prima query è quello di trovare per ogni finestra temporale specificata (un'ora, un girono e finestra globale), il valore medio del prezzo di vendita per tutte le azioni (aventi SecType="E") che iniziassero con la lettera "G" scambiate sul mercato di Parigi("FR") e il numero di eventi.
A tale scopo è stato creato un flusso che avesse come sorgente il topic Kafka sul quale Nifi produce le tuple preprocessate. In seguito sono state effettuate delle operazioni di filtraggio e di trasformazione per ottenere solo le tuple inerenti al mercato di Parigi che iniziassero con la lettera G e fossero azioni. Per la gestione delle tuple fuori ordine nello stream è stato inserito un watermark trasformando in timestamp il Trading Time concatenato con la Trading Date.
In seguito é stato effettuato un ragruppamento sull'ID che ha permesso di ottenere un keyed datastream sul quale sono state applicate le finestre tumbling, ottenendo così dei buffer su cui effettuare il calcolo delle statistiche richieste.
Sulle tre finestre é stata applicata una funzione di reduce per sommare tutti i valori raggruppati e contare il numero di valori presenti per ogni chiave e con la funzione di finestra offerta dalla reduce è stato preso il timestamp di inizio finestra. Infine abbiamo applicato una funzione di map per effettuare la divisione tra la somma dei valori e il numero dei valori ed ottenere così la media.
Dopo la computazione delle tuple all'interno delle finestre i risultati vengono scritti in un sink, ovvero in un topic Kafka, per poi essere presi dal consumer a cui é affidato il compito

di creare il file csv con i risultati.
Il DAG viene riportato in figura 3.

B. Query 2

La seconda query chiede di trovare per ognuna delle finestre richieste (mezz'ora, un'ora e un giorno) la classifica delle cinque migliori e peggiori azioni in base alla variazione del prezzo di vendita.
Anche in questo caso i dati vengono letti dal topic Kafka dove Nifi inserisce gli eventi preprocessati e viene successiavamente assegnato il watermark. Prima di dividere i dati in finestre è stato effettuato un raggruppamento per ID dell'azione. Una volta create le finestre si è proceduto nel calcolo della variazione di prezzo per le azioni tramite la funzione process con la quale i dati sono stati ordinati per timestamp ed è stata effettuata la differenza tra il primo e l'ultimo valore della finestra. Infine è stato preso il timestamp dell'inizio della finestra. Si è reso necessario per la classifica effettuare una window all, ovvero una finestra non raggruppata per chiave in modo tale da poter selezionare tramite una process i primi e gli ultimi cinque elementi. Tramite un sink i risultati vengono pubblicati sul relativo topic Kafka per poi essere presi dal consumer a cui é affidato il compito di creare il file csv con i risultati.
E' possibile vedere il DAG della query due in figura 4.

C. Query 3

La terza query richiedeva di calcolare il 25-esimo, 50-esimo e 75-esimo percentile per mercato (Francia, Amsterdam, Francoforte) sulla la variazione del prezzo delle singole azioni su tre finestre temporali (mezz'ora, un'ora, un giorno).
Dopo aver letto i dati dal topic Kafka dove Nifi inserisce gli eventi preprocessati e aver assegnato il watermark come per le due query precedenti i dati vengono raggruppati per ID dell'azione e successivamente divisi nelle finestre. Per il calcolo del precentile è stata utilizzata la libreria python `psquasre` che fornisce un'euristica sul percentile senza bisogno di memorizzare e ordinare tutti i dati all'interno della finestra, ottimizzando così il calcolo a scapito dell'accuratezza del risultato.
Per ogni finestra con una process si calcola la variazione del valore dell'azione ordinando sulla finestra e facendo la differenza tra il primo e l'ultimo valore, inoltre per effettuare il calcolo del quantile vengono aggiunti: tre campi contenenti lo stimatore del percentile, tre inizializzati come stringhe vuote utilizzati in seguito per inserire i valori dei percentili e un contatore a zero utile nella fase successiva in quanto l'algoritmo non può iniziare il calcolo se non sono stati passati allo stimatore almeno cinque valori.
Si è reso necessario effettuare un raggruppamento per paese di appartenenza dell'azione, una volta divisi i dati così raggruppati in finestra è stata applicata una reduce che aggiornasse gli stimatori tupla per tupla e arrivato a cinque elementi producesse il valore del quantile aggiornato. Tramite un sink i risultati vengono pubblicati sul relativo topic Kafka per poi essere presi dal consumer a cui é affidato il compito di creare il file csv con i risultati.
E' possibile vedere il DAG della query due in figura 5.

VIII. Spark Streaming

E' stato utilizzato anche Spark streaming per eseguire la query uno. Anche Saprk è stato containerizzato con un container per il master e uno per il worker.
I dati vengono presi dal topic Kafka in cui pubblica i messaggi direttamente il producer python senza passare per Nifi. I dati da Kafka vengono memorizzati in un dataframe, da cui vengono selezionate solo le colonne utili e successivamente filtrate le righe con campo data vuoto e orario nullo. Viene creato un campo in cui si inserisce la data concatenata all'ora e poi convertito in timestamp per utilizzarlo come watermark per le finestre. A questo punto inizia il processamento per la query, si effettuano tre filter: sul campo SecType dev'essere 'E', l'ID dell'azione deve iniziare per 'G' e le azioni devono appartenere al mercato Francese, quindi il campo ID deve terminare in 'FR'. Si può ora dividere in finestre raggruppando sul campo ID ed usare come funzioni di aggregazione la media e count per ottenere il numero di eventi. Infine vengono selezionate le righe di interesse e scritti i risultati sul topic Kafka da cui leggerà il consumer python locale che si ocuperà di salvarli su file csv.

IX. Prometheus e Grafana

Le metriche sono prese utilizzando le classi di monitoarggio native di Flink, in particolare sono stati creati due `Gauge` uno per il calcolo della latenza e un altro per il calcolo del throughput.
Flink espone queste metriche autonomamente sulla sua Web UI.
Si é deciso di utilizzare Prometheus come framework di monitoraggio, in modo da prelevare queste metriche ogni 200ms.
Grafana é stato utilizzato come framework di visualizzazione di quest'ultime connettendosi direttamente a Prometheus mette a disposizione una dashboard che monitora sia le metriche customizzate, che ulteriori indici di performance dei nodi su cui vengono eseguiti gli operatori.

X. Analisi dei risultati e performance

L'analisi dei risultati si basa su due metriche: throughput, latenza.
Sono stati effettuati dei confronti tra l'esecuzione dei job utilizzando nifi e l'esecuzione in cui è lasciato a Flink il compito del preprocessamento. E' stato effettuato anche un confronto prestazionale tra la prima query eseguita con Flink e con Spark streaming:

- **Latenza preprocessing Nifi vs. Flink**
  Per quanto riguarda la query uno, come si può osservare dai grafici in figura 6 e 7, la latenza sembra raggiungere la stazionarietà in entrambe i casi, senza il preprocessing di Nifi aumenta di circa il 40%. I risultati coincidono con quelli attesi, in quanto il sistema non va in saturazione e la

latenza senza Nifi risulta maggiore a causa delle ulteriori operazioni preliminari delegate a flink.
Nella query due e tre la finestra di una giornata risulta più alta rispetto alla finestra di mezz'ora e oraria sia per quanto riguarda il preprocessamento con Nifi che senza, per analizzare meglio le differenze delle prime due finestre sono riportati in figura 8 e 9 i grafici senza la terza finestra per la query 2 e in figura 12 e 13 per la query 3. Per la query due si perde il miglioramento ottenuto nella prima query con Nifi, infatti per la finestra di un'ora la latenza è uguale, mentre si ha un miglioramento di circa il 33%. Per la finestra giornaliera il miglioramento rimane costante sul 30% come si può vedere dalle figure 10 e 11. Per la terza query per tutte e tre le finestre i risultati sono circa uguali, si nota un aumento senza nifi intorno al 5%. Il 14 e 15 sono mostrati i grafici completi.

- **Throughput preprocessing Nifi vs. Flink**
  Sono riportati i grafici del throughput con e senza il preprocessing di Nifi in figura: 16 e 17 per la query uno, 18 e 19 per la query due e 20 e 21 per la query tre. Si possono fare le stesse considerazioni fatte per la latenza sulle tre query, in questo caso si vede come la differenza tra i throughput sia maggiore nel caso della query uno con il preprcessing di nifi e questa differenza vada diminuendo nelle query due e tre.
- **Query uno Spark vs. Flink**
  Nelle figure 22, 23, 24, sono riportate le metriche di Spark streaming. Si nota subito dalla latenza (grafico batch duration) come le prestazioni siano nettamente peggiori, in media la latenza sta circa a 2.5 secondi, leggermente inferiore per le finestre di un'ora e un giorno e superiore per la finestra globale. La latenza media della query uno eseguita con Flink senza il preprocessing di Nifi si attestava intorno ai 600ms con prestazioni nettamente superiori rispetto a spark.

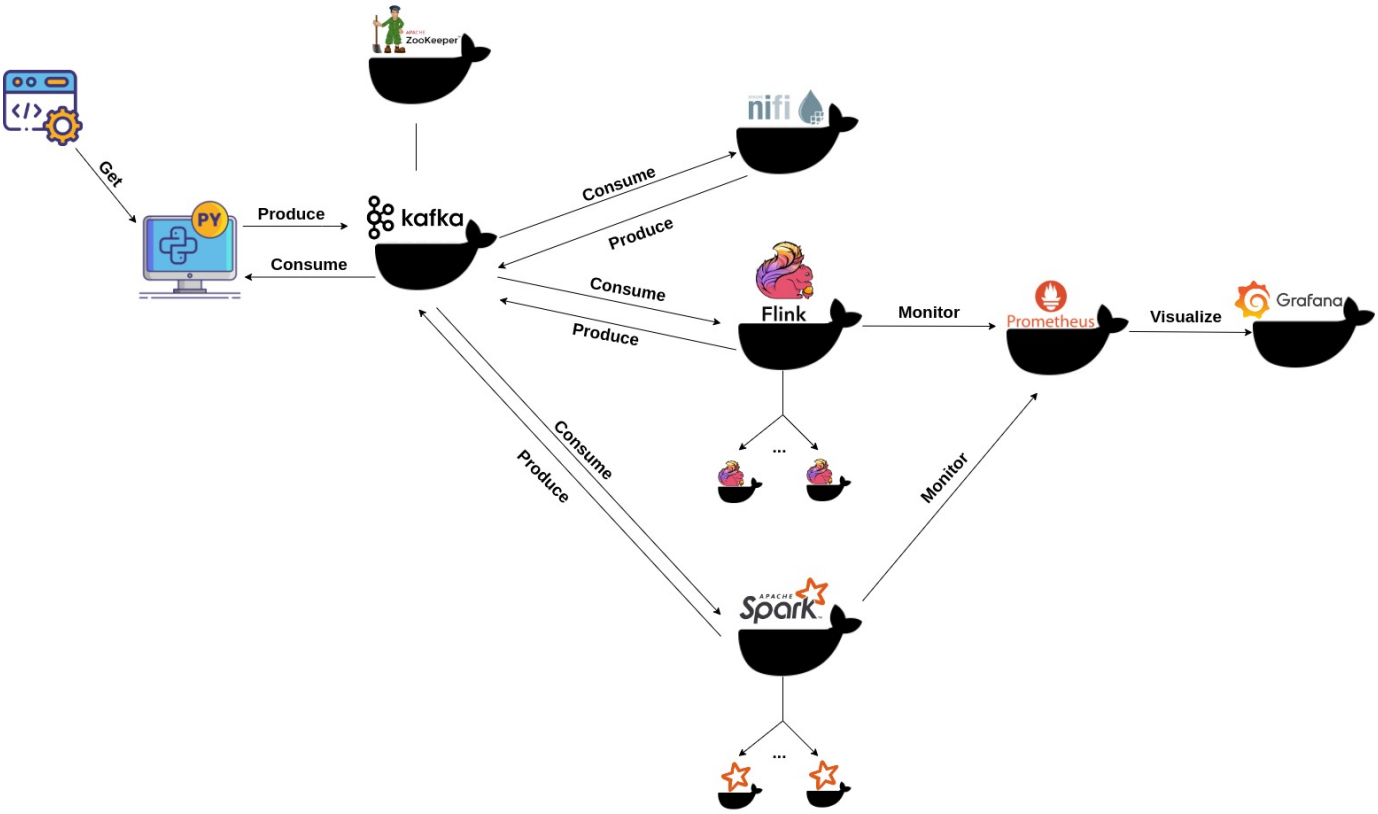

Fig. 1. Architettura

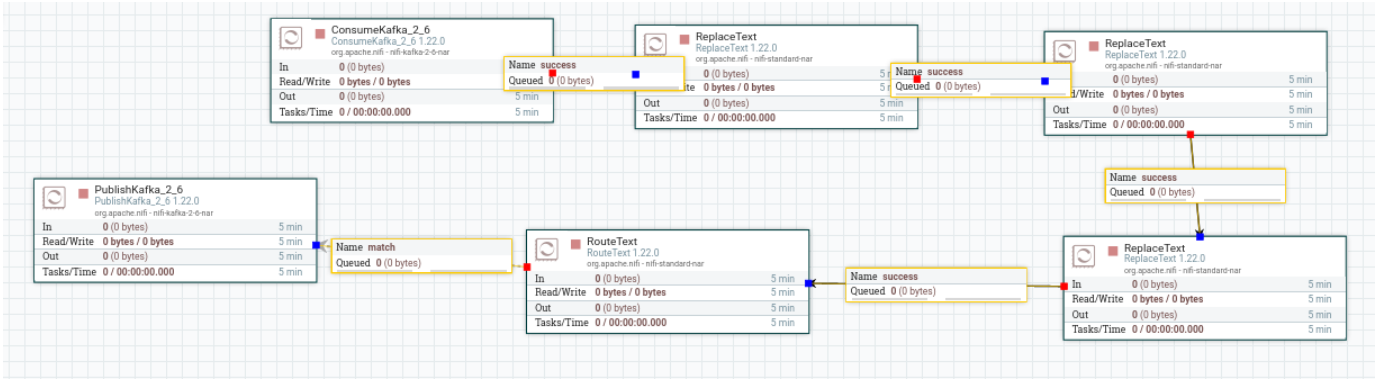

Fig. 2. Schema NIFI

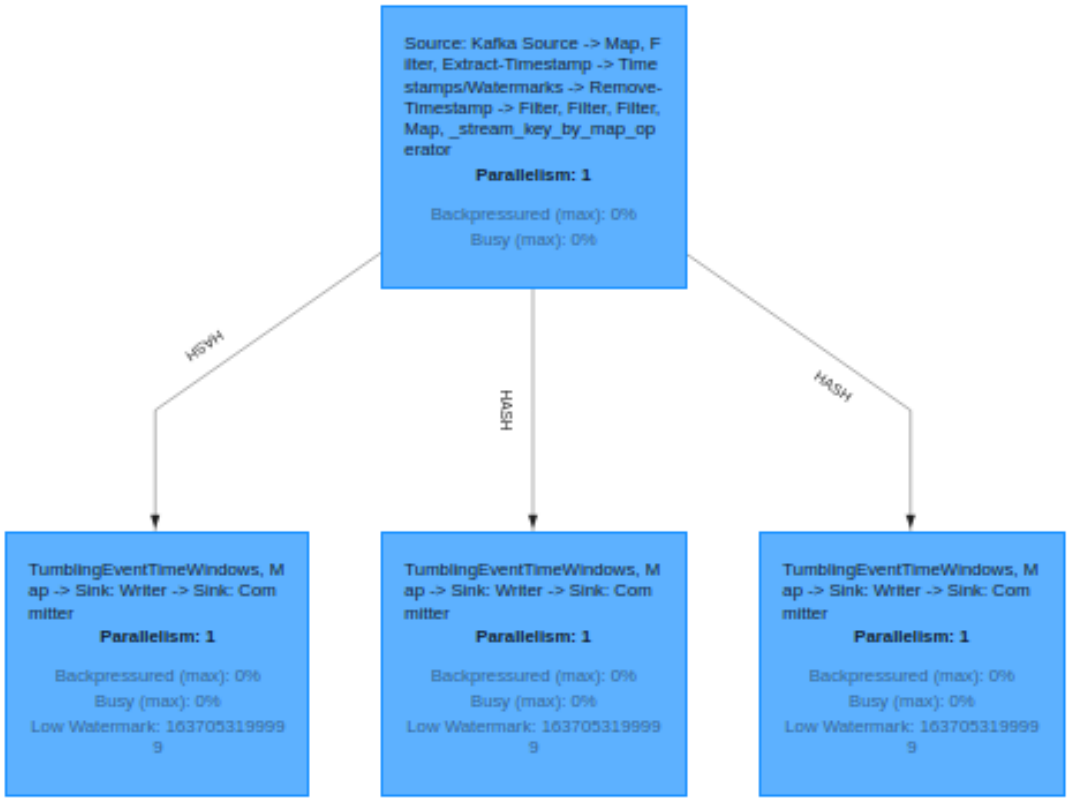

Fig. 3. DAG query 1

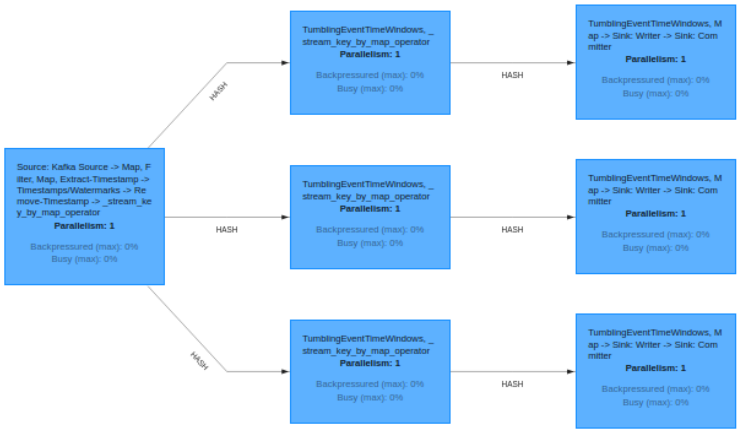

Fig. 4. DAG query 2

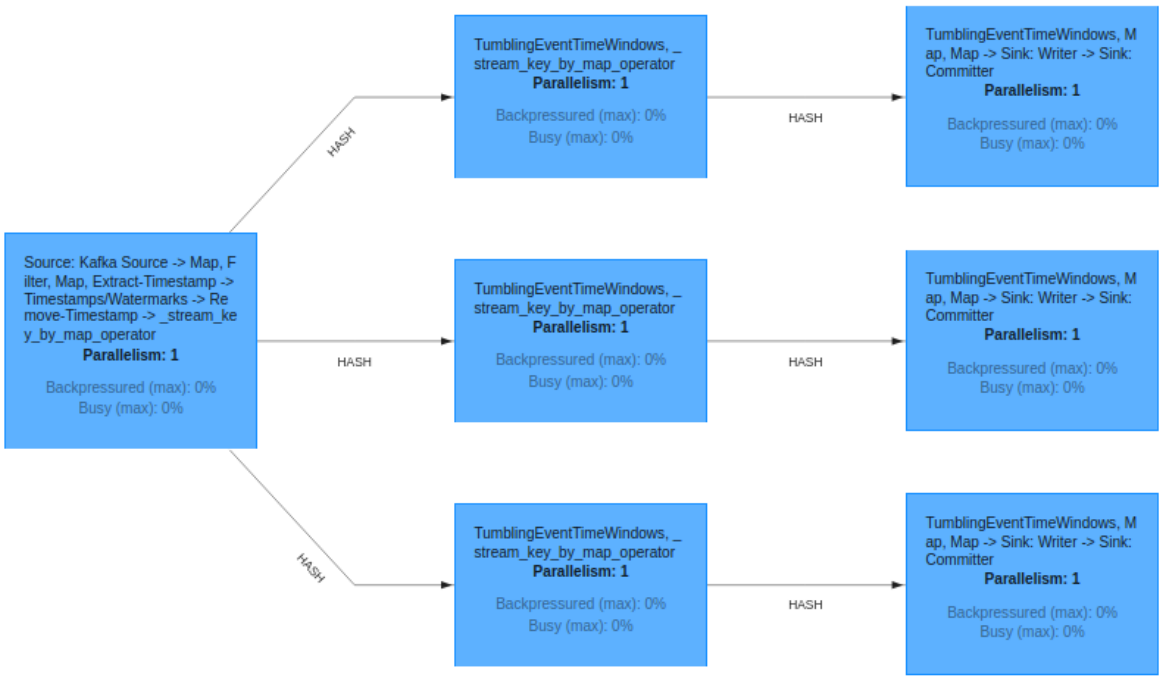

Fig. 5. DAG query 3

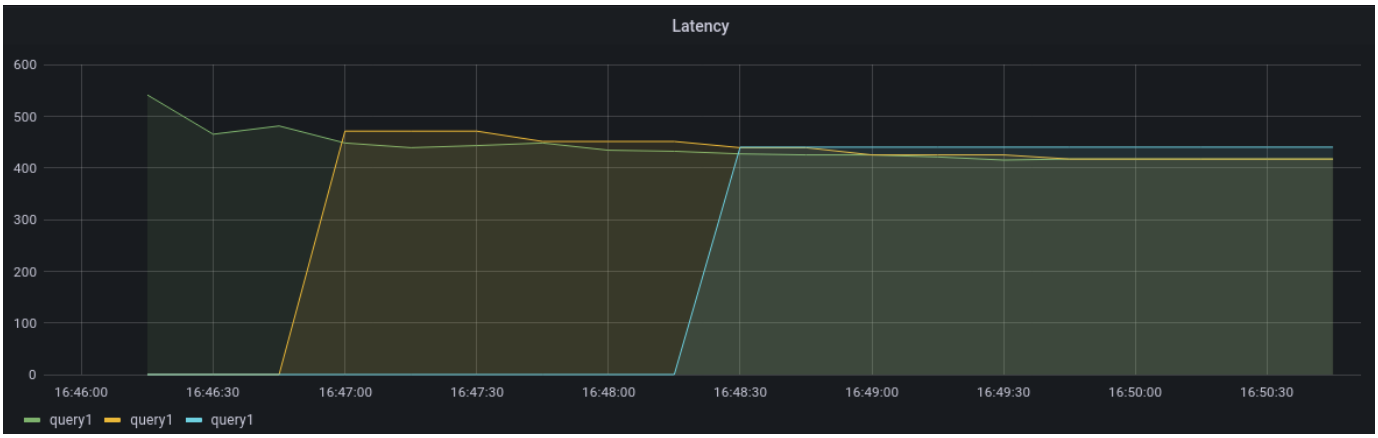

Fig. 6. Latenza query 1 con Nifi

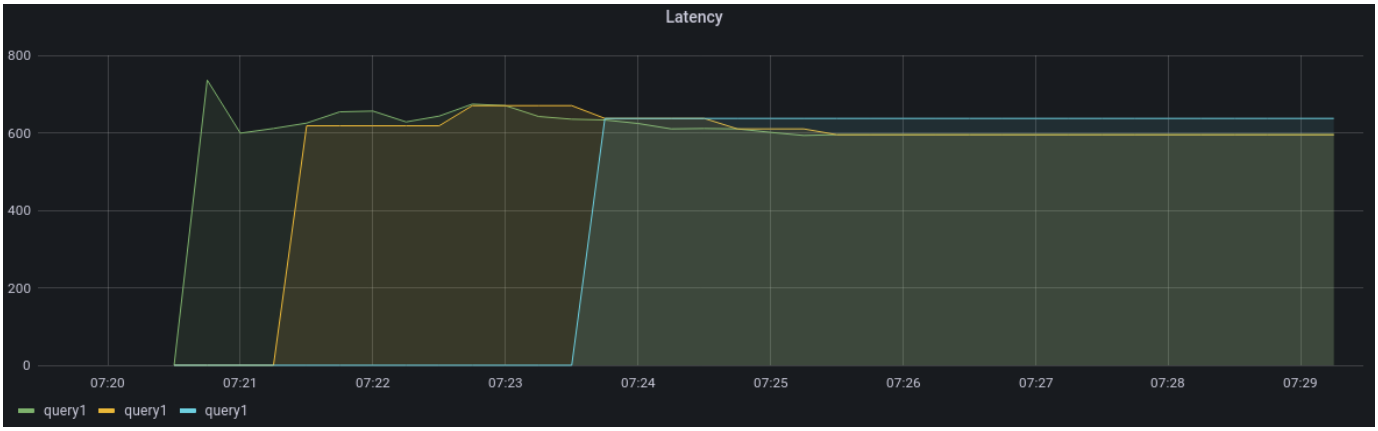

Fig. 7. Latenza query 1 senza Nifi

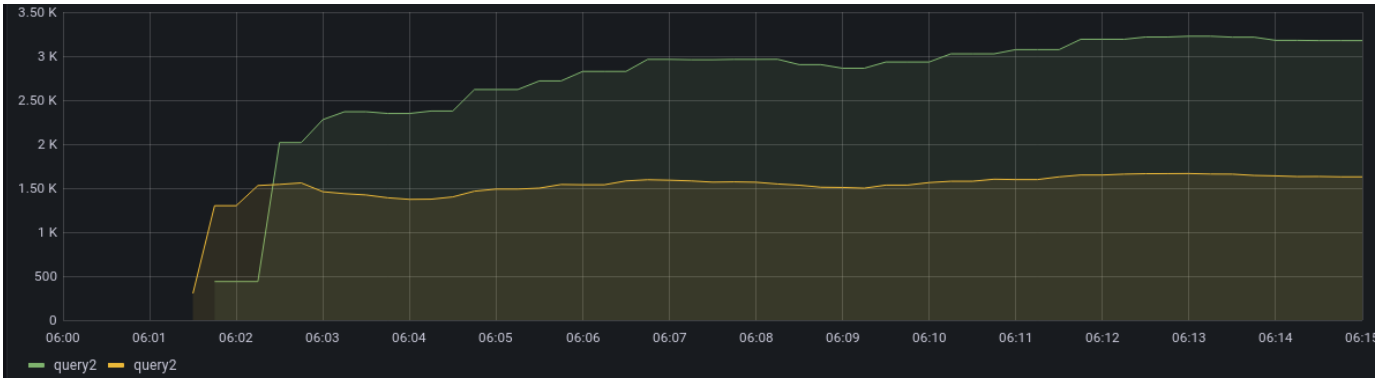

Fig. 8. Latenza query 2 Nifi finestre 30 min 1 ora

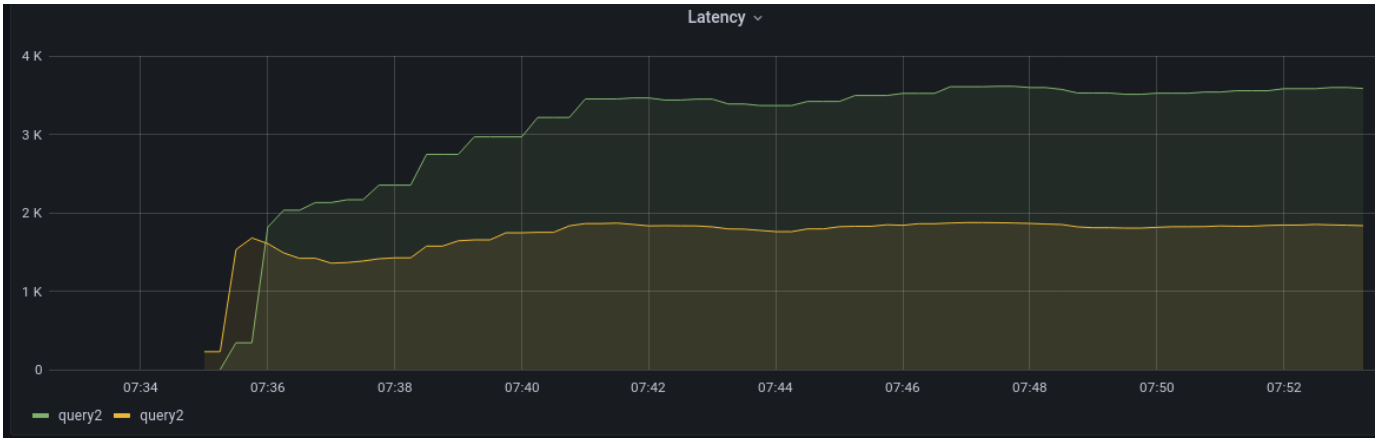

Fig. 9. Latenza query 2 senza Nifi finestre 30 min 1 ora

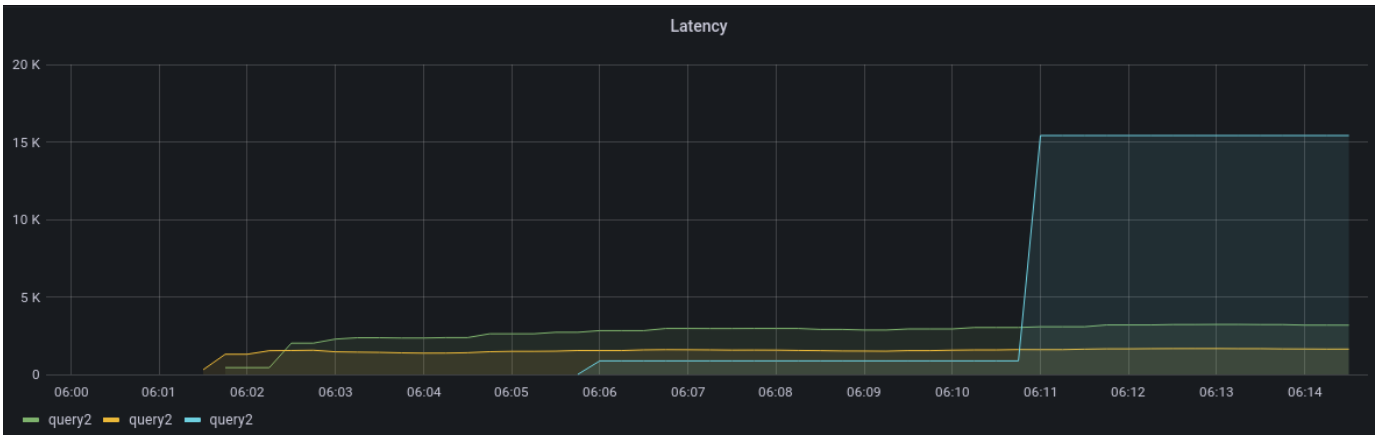

Fig. 10. Latenza query 2 Nifi

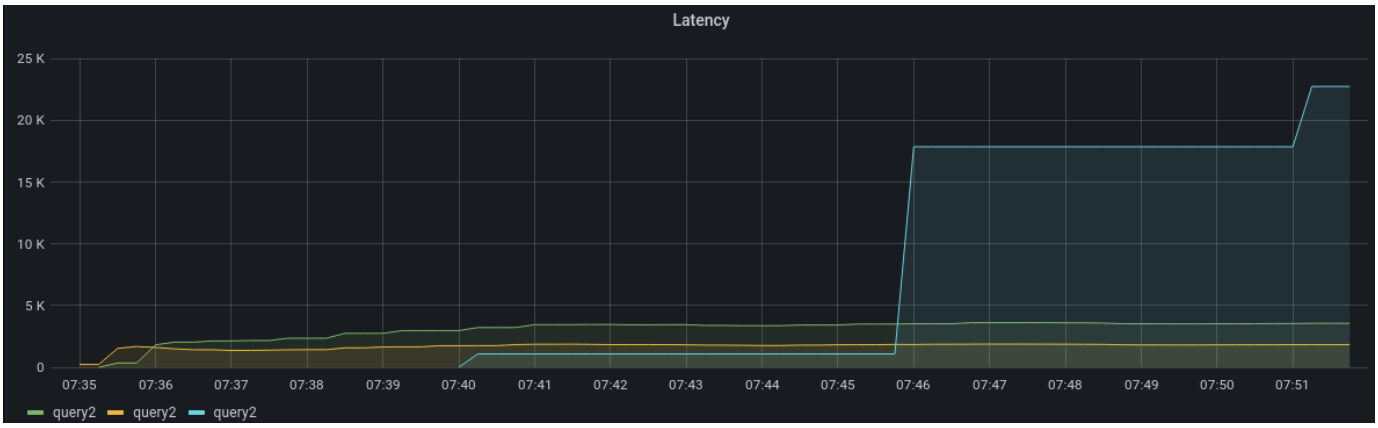

Fig. 11. Latenza query 2 senza Nifi

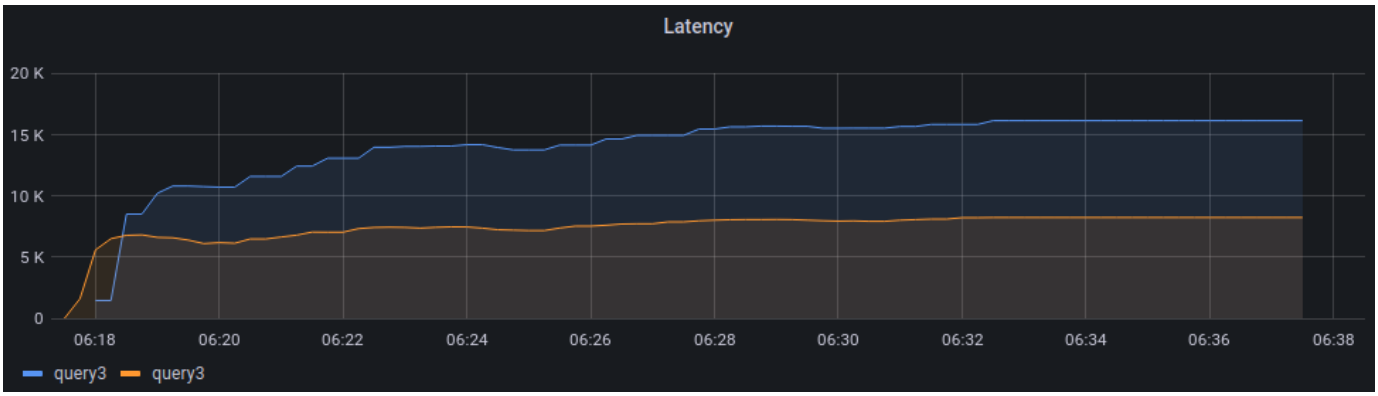

Fig. 12. Latenza query 3 Nifi finestre 30 min 1 ora

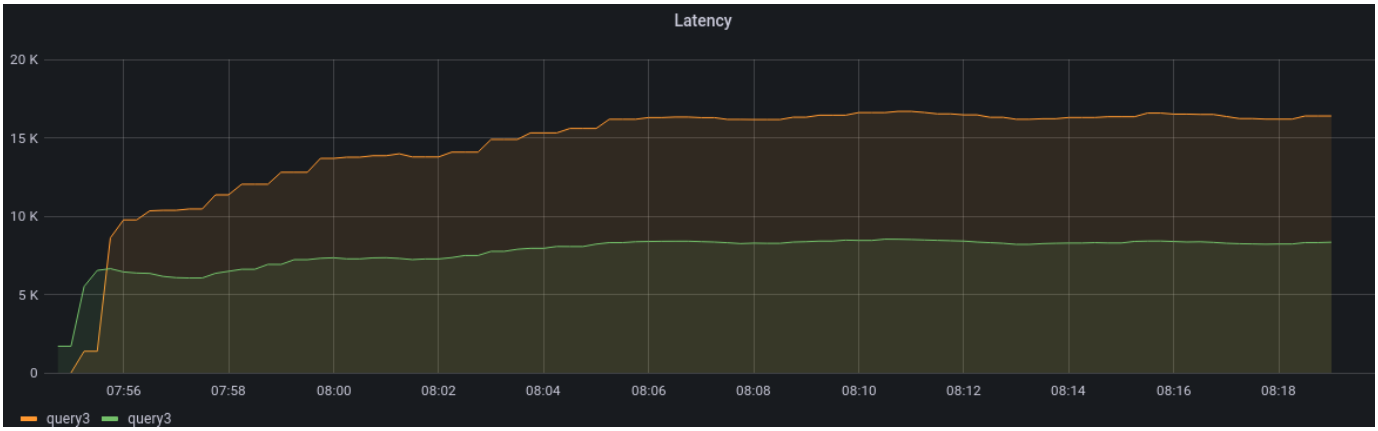

Fig. 13. Latenza query 3 senza Nifi finestre 30 min 1 ora

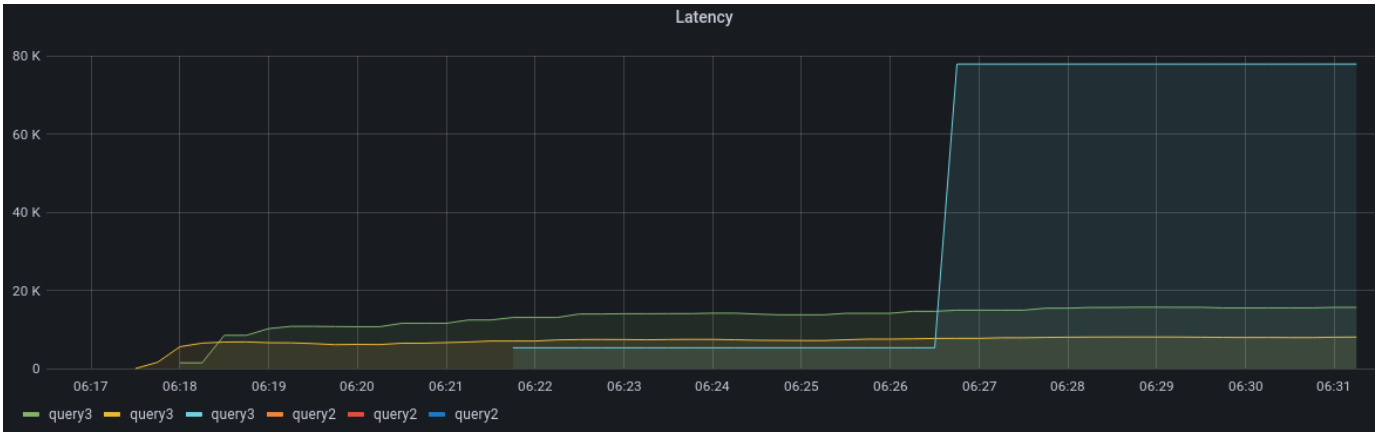

Fig. 14. Latenza query 3 Nifi

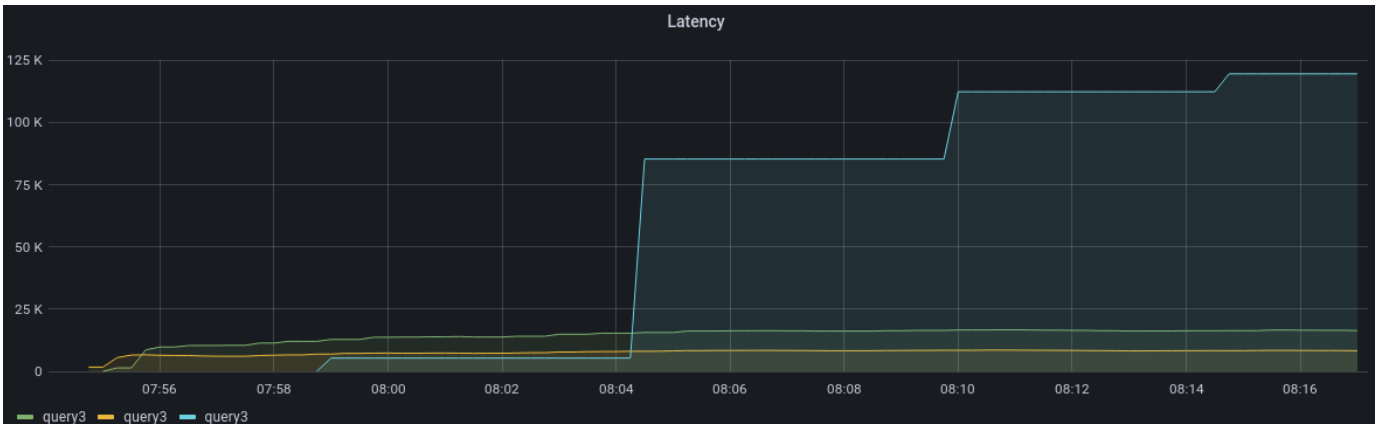

Fig. 15. Latenza query 3 senza Nifi

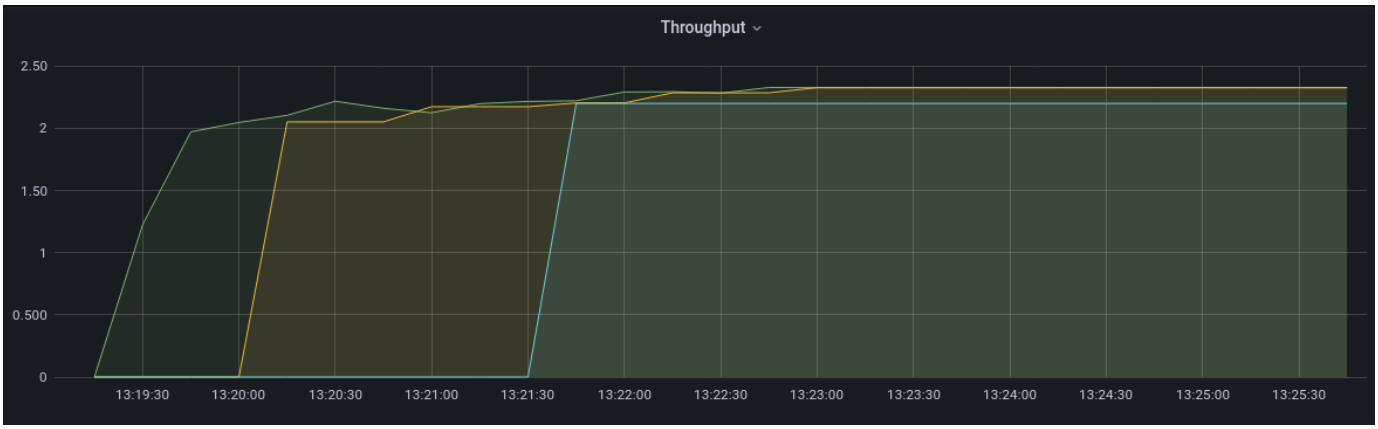

Fig. 16. throughput query 1 Nifi

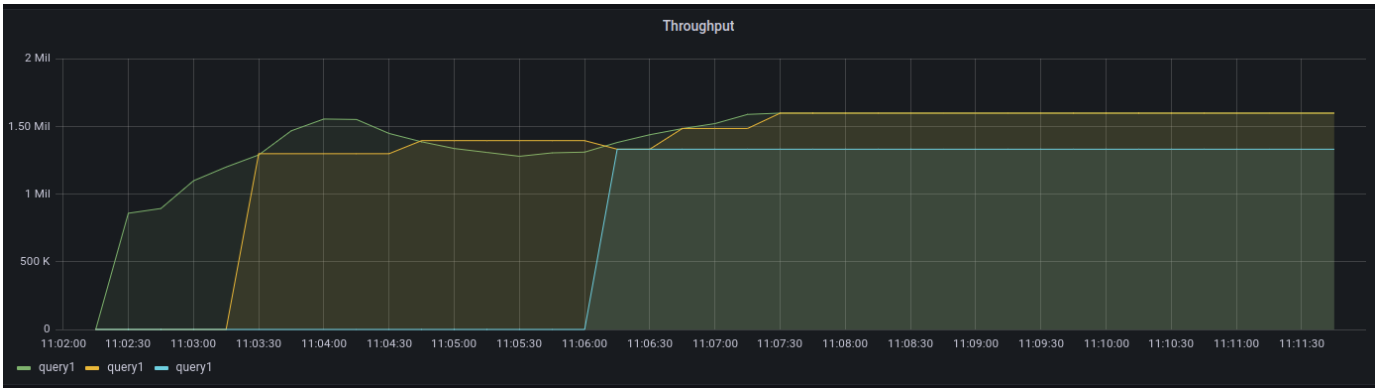

Fig. 17. Throughput query 1 senza Nifi

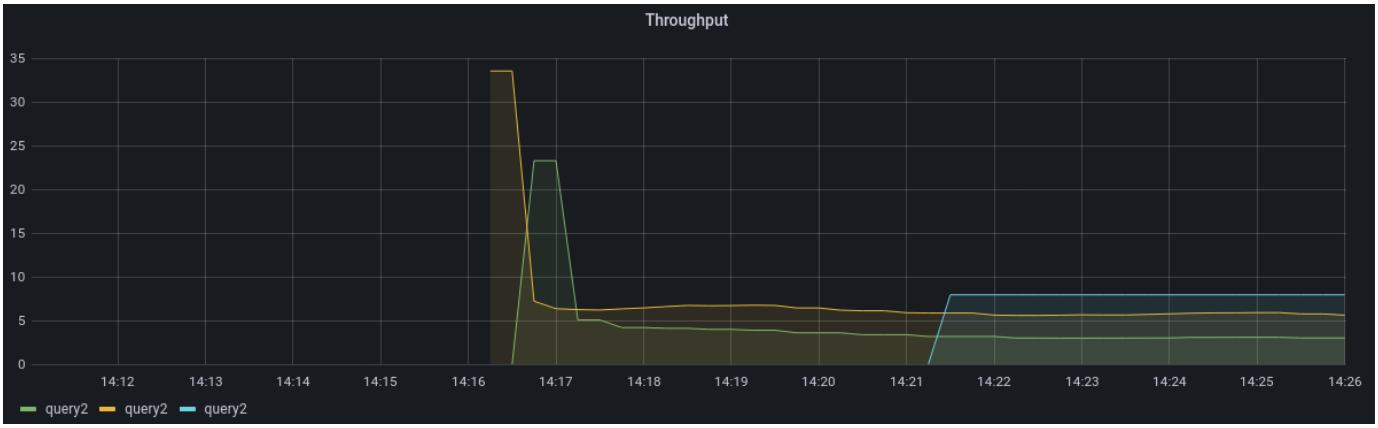

Fig. 18. throughput query 2 Nifi

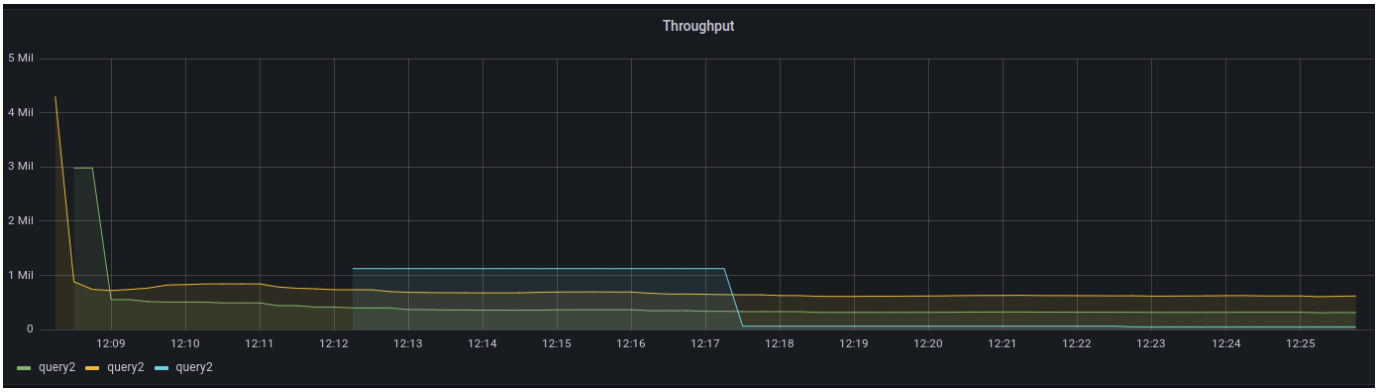

Fig. 19. Throughput query 2 senza Nifi

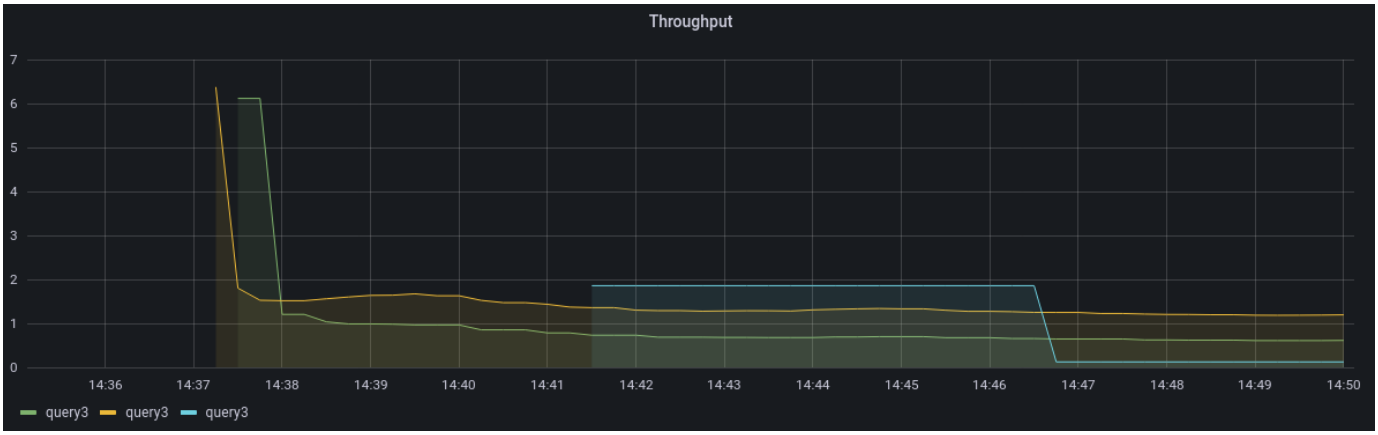

Fig. 20. throughput query 3 Nifi

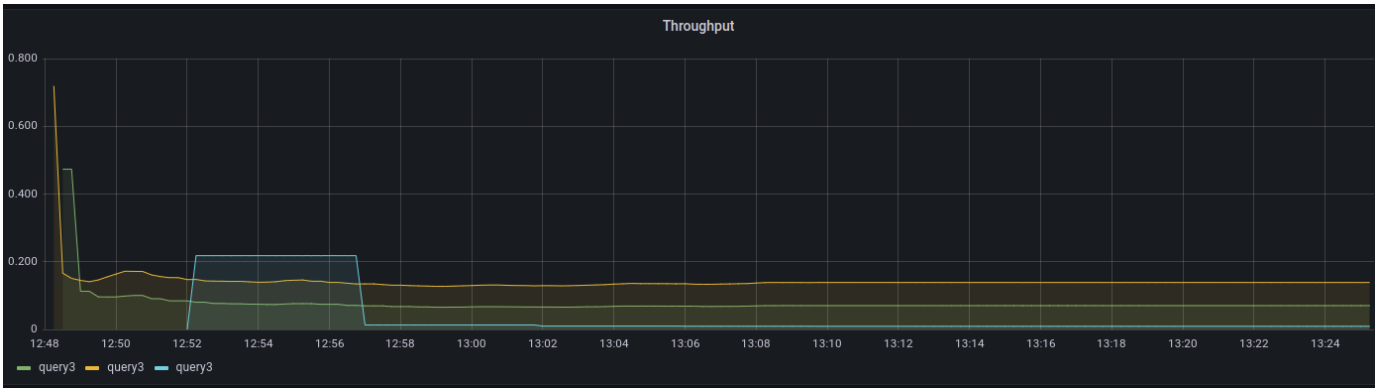

Fig. 21. Throughput query 3 senza Nifi

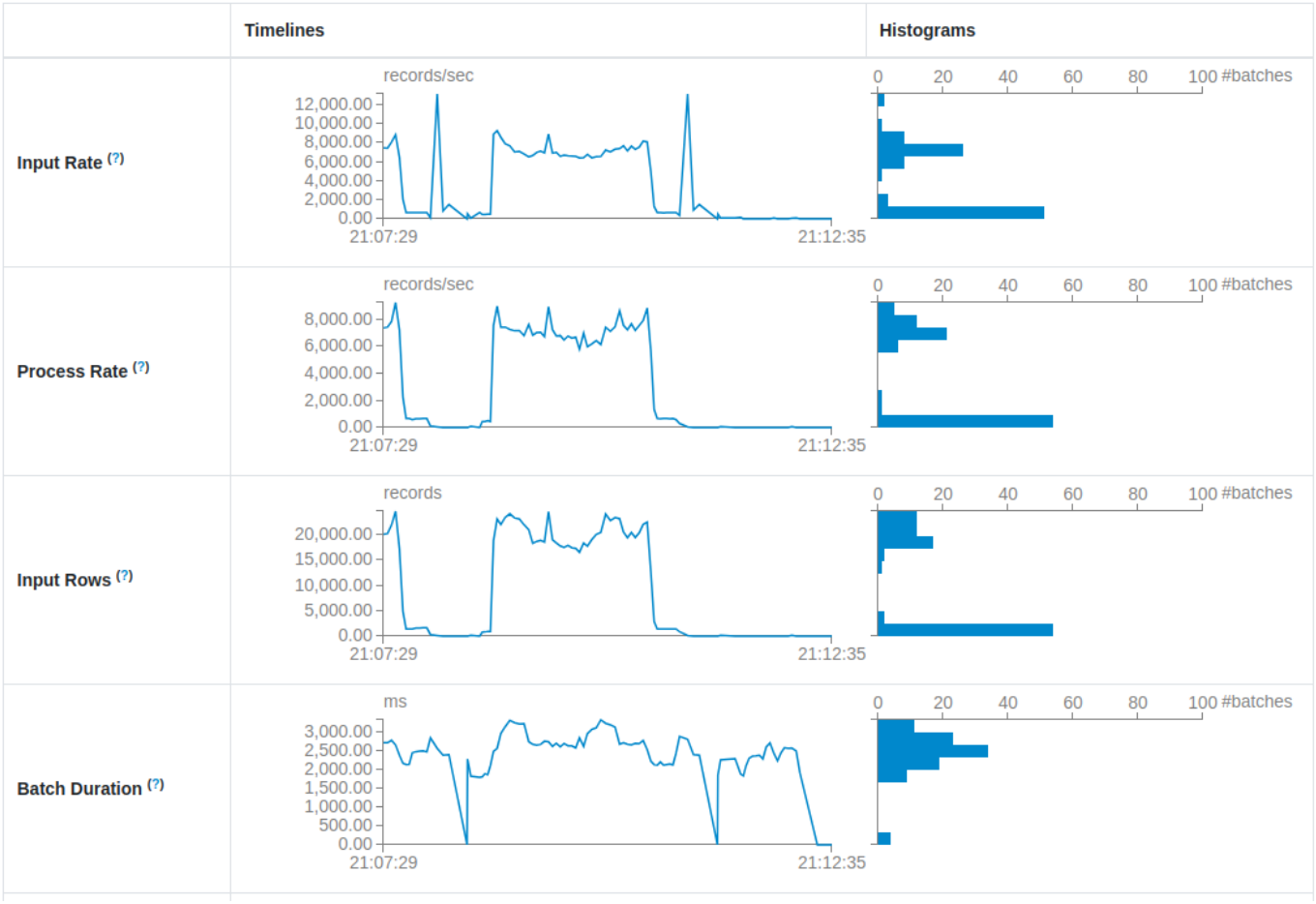

Fig. 22. Metriche Spark finestra oraria

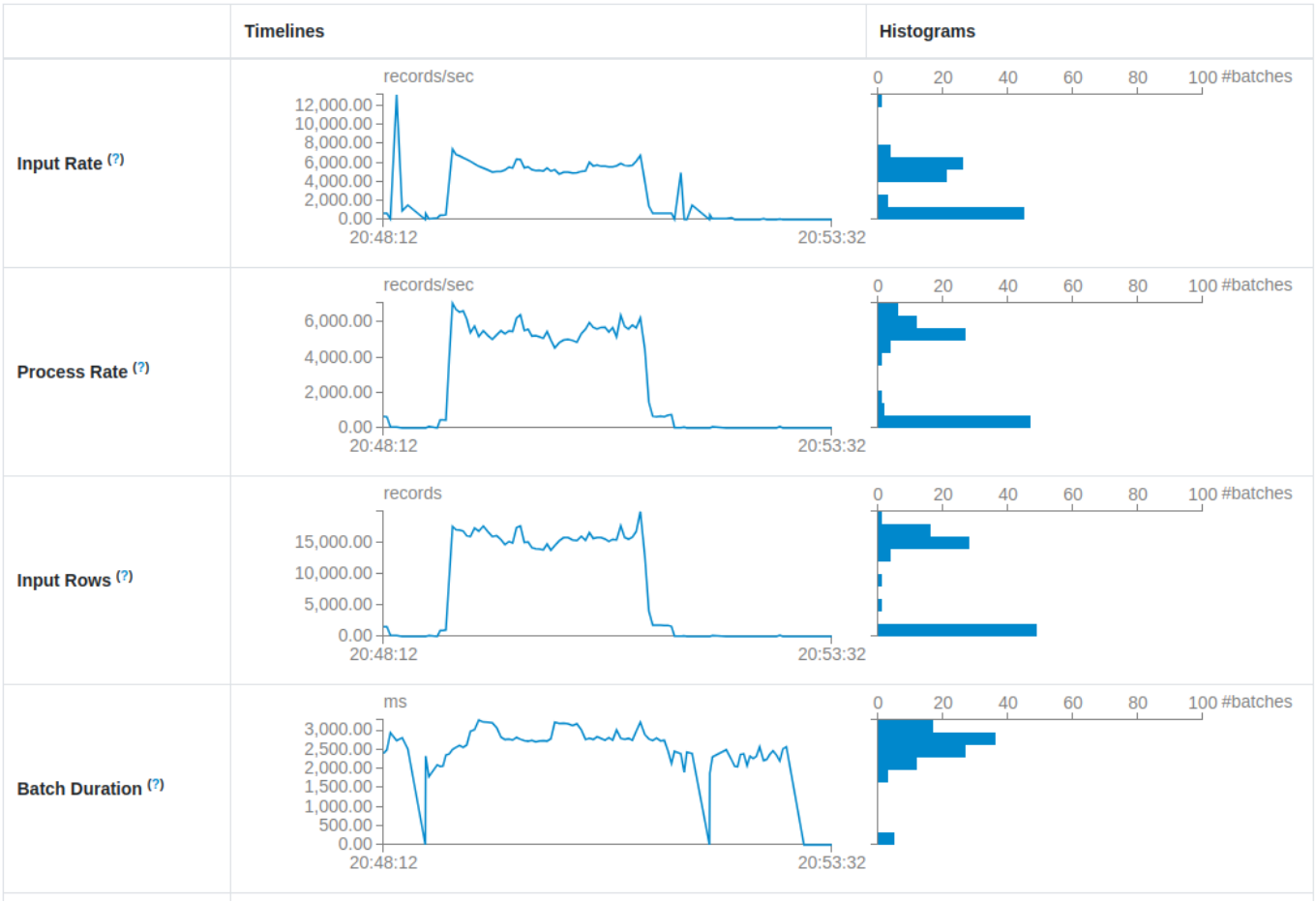

Fig. 23. Metriche Spark finestra giornaliera

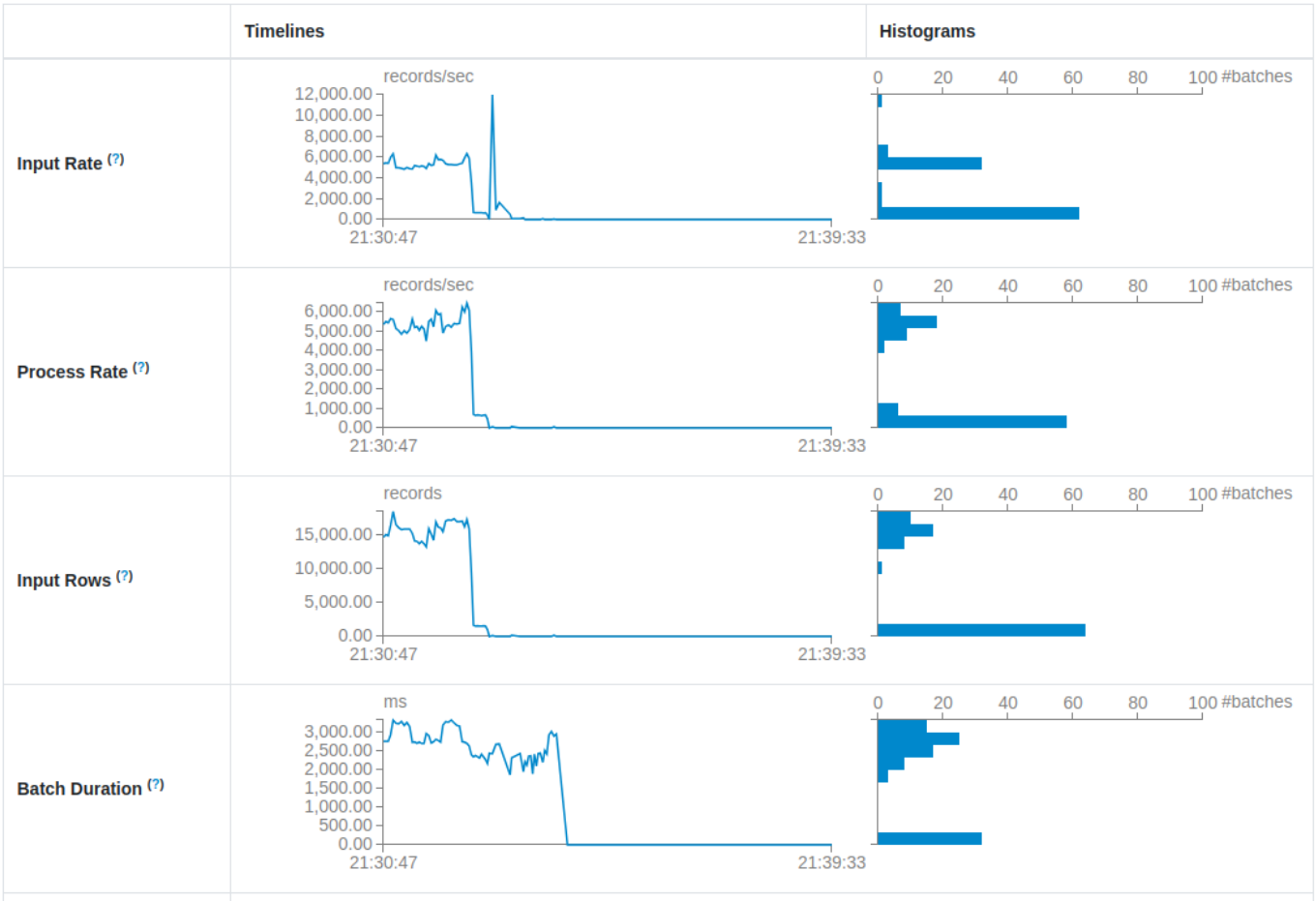

Fig. 24. Metriche Spark finestra globale