SABD – Progetto 2

Progetto valido per il corso di Sistemi e Architetture per Big Data

Di Marco Luca
0333083

Di Totto Luca
0333084

***Abstract*—Questo documento si pone lo scopo di descrivere la progettazione e la realizzazione di un'applicazione per l'elaborazione di dati riguardanti misurazioni sui guasti verificatisi su hard disk, utilizzando un'architettura basata su container Docker. L'applicazione prevede la simulazione di un sistema di invio delle metriche giornaliero in Python, Apache Kafka come framework di messaggistica e Apache Flink per recuperare i dati ed effettuare l'elaborazione streaming di questi.**

INTRODUZIONE

L'obiettivo del progetto è quello di analizzare ed eseguire query su un dataset riguardante i dati di monitoraggio S.M.A.R.T., esteso con alcuni attributi aggiuntivi. Quest'ultimo contiene eventi riguardanti circa duecento mila hard disk in un intervallo temporale di 23 giorni, per un totale di circa tre milioni di eventi.
Nell'implementazione, vengono simulati i giorni presenti nel dataset, tramite un ritardo introdotto al momento dell'invio dei dati.
Il documento ha lo scopo di descrivere nel dettaglio l'architettura utilizzata e le parti fondamentali: verranno prima descritti i componenti del sistema e i relativi dettagli implementativi; verranno poi analizzate in dettaglio le query; successivamente verranno analizzate le tempistiche e le prestazioni generali ed infine verranno discusse determinate considerazioni sul sistema in oggetto.

ARCHITETTURA

L'architettura complessiva del sistema è composta da diversi framework, ognuno con caratteristiche ed obiettivi differenti. Il sistema viene interamente containerizzato sfruttando la virtualizzazione di sistema offerta da Docker.
Ognuno dei componenti viene ospitato all'interno di uno o più container Docker e vengono interconnessi tra loro attraverso l'utilizzo di una unica rete virtuale.
Si analizzano di seguito i vari componenti del sistema in oggetto:

- Kafka (*wurstmeister/kafka:latest*)
- Flink (*flink:latest*)
- Kafdrop (*kafdrop:latest*)
- Python (*python:latest*)
- Zookeeper (*zookeeper:3.8.0*)

Si veda *Figura 1: Architettura del sistema*.

KAFKA

È un framework a livello di messaggistica distribuita, estremamente scalabile e fault-tolerant, che utilizza un pattern di comunicazione di tipo publisher-subscriber, progettato per gestire grandi quantità di dati in tempo reale. Nella configurazione, Kafka funge da broker di messaggi e distribuisce le tuple generate dalla applicazione Python, che funge da produttore. Flink, recupera le tuple e le elabora in tempo reale, svolgendo il ruolo di consumer. Per un funzionamento ottimale di Kafka, è necessaria la presenza di Zookeeper, che funge da coordinatore.

PYTHON

L'applicativo Python viene utilizzato per generare il flusso di tuple dal dataset all'interno del container *python-data-ingestion*, in modo da simulare lo streaming di dati. Dal dataset presente vengono generati 23 dataframes, corrispondenti ai 23 giorni nel dataset, sfruttando la libreria di processamento dati Pandas. Successivamente alla connessione verso Kafka e alla sottoscrizione come producer sul topic di riferimento, questi dataframes, vengono inviati in maniera continua, tupla dopo tupla, per simulare il flusso streaming. Viene ritardato invece l'invio delle tuple appartenenti ad un dataframe (quindi le registrazioni afferenti ad un unico giorno) dall'invio delle tuple del dataframe successivo, in modo da simulare lo scorrere dei giorni in un contesto reale.

APACHE FLINK

Apache Flink è un framework di processamento streaming. In questa configurazione Docker viene utilizzata l'immagine più recente di Flink per creare i due servizi principali: il JobManager e il TaskManager.

- JobManager: viene avviato un container con il ruolo di JobManager, responsabile della gestione delle risorse e del coordinamento dei lavori di streaming. Viene esposta sulla porta 8081 per l'accesso all'interfaccia web.
- TaskManager: viene avviato un container con il ruolo di TaskManager, responsabile dell'esecuzione effettiva delle operazioni di calcolo. Ogni TaskManager è configurato con 2 slot di task e 2048 MB di memoria.

Entrambi i servizi dipendono da Kafka per il recupero delle tuple come consumer

KAFDROP

Viene utilizzato Kafdrop per la visualizzazione e la gestione di Kafka. Esso è in grado di connettersi al broker Kafka e offre una interfaccia web per monitorare e gestire i topic e i messaggi.

SALVATAGGIO DEI RISULTATI

Per ogni query i risultati vengono salvati in formato CSV sul file system del container Docker del JobManager (che vengono successivamente copiati nel file system della macchina host per poter salvare i risultati in maniera persistente).

QUERY

Le query vengono scritte in Java e vengono successivamente impacchettate in formato JAR, per essere rese eseguibili all'interno dell'architettura Flink.

Nell'implementazione di tutte le query, viene realizzato un oggetto Message utilizzato per mappare le informazioni contenute in ciascuna tupla che Flink recupera da Kafka. La classe Message presenta tutti i metodi di get e set necessari all'elaborazione e un metodo create che ha il compito di convertire la stringa JSON in un oggetto.

QUERY 1

Per i vault (campo vault id) con identificativo compreso tra 1000 e 1020, calcolare il numero di eventi, il valor medio e la deviazione standard della temperatura misurata sui suoi hard disk (campo s194 temperature celsius). Si faccia attenzione alla possibile presenza di eventi che non hanno assegnato un valore per il campo relativo alla temperatura. Per il calcolo della deviazione standard, si utilizzi un algoritmo online, come ad esempio l'algoritmo di Welford. Calcolare la query sulle finestre temporali:

- 1 giorno (event time);
- 3 giorni (event time);
- dall'inizio del dataset.

IMPLEMENTAZIONE

Viene creato l'ambiente di esecuzione di Flink (ovvero, il contesto in cui viene eseguita l'applicazione di streaming). Successivamente viene configurata una sorgente Kafka per leggere messaggi dal topic "*my-topic*" e vengono deserializzati i messaggi come stringhe. I messaggi letti da Kafka vengono mappati in oggetti Message in modo da recuperare i campi utili all'elaborazione e vengono filtrati i messaggi nulli e quelli il cui valore di vault_id non è compreso tra 1000 e 1020. I messaggi filtrati vengono mappati in tuple contenenti un timestamp e il messaggio stesso. Viene successivamente assegnata una strategia di watermarks. I dati vengono raggruppati in finestre di tempo di 1, 3 e 23 giorni, utilizzando finestre tumbling. I dati nelle finestre vengono elaborati con una funzione custom che calcola le statistiche sulle temperature, filtrando e non considerando le tuple che non presentano un valore di temperatura. I risultati elaborati vengono scritti su file CSV utilizzando un formato di output personalizzato.

- La funzione *VaultStatisticsProcessAllWindowFunction* è quella che si occupa di elaborare i dati all'interno di ciascuna finestra per calcolare statistiche specifiche e raccogliere i risultati.
- Viene creata una istanza di Statistics per ogni vault_id: per ogni messaggio nella finestra, viene estratto il vault_id e la temperatura (*s194_temperature_celsius*) se presente.
- Vengono incrementati i contatori per le tuple per ogni vault_id e calcolate le statistiche richieste (valor medio e deviazione standard) tramite l'algoritmo online di *Welford*, che consente di aggiornare media e varianza in modo iterativo, senza avere la necessità di memorizzare tutti i valori precedenti. I valori calcolati e il numero di messaggi visti per vault_id, sono memorizzati in una hash map formata dal vault_id e da un'istanza dell'oggetto *statistics.*
- Vengono poi presi i tempi di inizio e fine finestra, e trasformati in un formato leggibile così da poter riportare quest'informazione in output
- Vengono infine raccolte le statistiche di quella specifica finestra temporale e i risultati ottenuti vengono preparati per essere scritti in output

QUERY 2

Calcolare la classifica aggiornata in tempo reale dei 10 vault che registrano il più alto numero di fallimenti nella stessa giornata. Per ogni vault, riportare il numero di fallimenti ed il modello e numero seriale degli hard disk guasti. Calcolare la query sulle finestre temporali:

- 1 giorno (event time);
- 3 giorni (event time);
- dall'inizio del dataset.

IMPLEMENTAZIONE

Anche in questo caso viene creato l'ambiente di esecuzione di Flink e viene poi configurata una sorgente Kafka per leggere messaggi dal topic "*my-topic*" e deserializzati i messaggi letti in stringhe. Vengono mappate le stringhe in oggetti Message, filtrando solo i messaggi che presentano nel campo failure un valore uguale a "1". Lo stream dei dati viene poi ulteriormente processato andando a creare per ogni messaggio letto, una tupla composta dal vault_id di quel messaggio, un contatore posto inizialmente a 1 e una lista di oggetti di tipo Message. Successivamente la funzione calculateTop10 calcola i primi 10 vault_id che presentano il maggior numero di failure all'interno della finestra temporale specificata.

- Il metodo *apply* della funzione *calculateTop10* si occupa quindi di analizzare le tuple presenti in una finestra temporale, contare i fallimenti verificatisi in ogni vault_id, calcolare la classifica e scrivere il risultato in output.
- Scorre tutte le tuple precedentemente create nello stream e crea una mappa in cui la chiave è il vault_id e il valore è il messaggio stesso. Se è già presente nella mappa una coppia con la stessa chiave, viene creata una nuova tupla con stessa chiave, somma dei campi failure delle due coppie e lista di messaggi che contiene la composizione delle liste delle due coppie. In questo modo viene quindi effettuato un

raggruppamento per vault_id e la lista di messaggi serve a tener traccia di quali sono i modelli e i seriali degli hard disk di quello specifico vault_id che hanno presentato una failure.
- Dopo aver eseguito la procedura per tutte le tuple presenti, vengono ordinati in maniera decrescente in base al numero di fallimenti e poi presi i primi dieci elementi.
- Vengono poi presi i tempi di inizio e fine finestra, e trasformati in un formato leggibile così da poter riportare quest'informazione in output
- Tutti i dati ottenuti vengono poi formattati in maniera opportuna per essere riportati in output.

QUERY3

Calcolare il minimo, 25-esimo, 50-esimo, 75-esimo percentile e massimo delle ore di funzionamento (campo s9 power on hours) degli hark disk per i vault con identificativo tra 1090 (compreso) e 1120 (compreso). Si presti attenzione, il campo s9 power on hours riporta un valore cumulativo; pertanto, le statistiche richieste dalla query devono far riferimento all'ultimo valore utile di rilevazione per ogni specifico hard disk (si consideri l'uso del campo serial number). I percentili devono essere calcolati in tempo reale, senza ordinare tutti i valori e possibilmente senza accumularli; si utilizzi pertanto un algoritmo approssimato che consente di calcolare i percentili riducendo la quantità di memoria occupata al prezzo di una minore accuratezza. Calcolare la query sulle finestre temporali:

- 1 giorno (event time);
- 3 giorni (event time);
- dall'inizio del dataset.

IMPLEMENTAZIONE

Viene creato l'ambiente di esecuzione di Flink (ovvero, il contesto in cui viene eseguita l'applicazione di streaming). Successivamente viene configurata una sorgente Kafka per leggere messaggi dal topic "*my-topic*" e come nei casi precedenti vengono deserializzati i messaggi come stringhe. Viene configurata una strategia di watermark per gestire l'ordine degli eventi in base al campo date degli eventi Message. Dalle tuple ricevute vengono filtrate quelle nulle o quelle che non presentano un valore nel campo vault_id. Vengono successivamente filtrati i messaggi in modo da includere solo quelli il cui valore di vaultId è compreso tra 1090 e 1120. Successivamente la funzione *calculateStats* calcola le statistiche sulle finestre temporali di diverse dimensioni (1 giorno, 3 giorni, e l'intero dataset).

- Il metodo *apply* della funzione *calculateStats* viene eseguito su ogni finestra temporale in modo da elaborare i messaggi inclusi in essa. Utilizza quattro differenti HashMap per tenere traccia del valore del TDigest necessario a calcolare i quantili, del valore minimo, di quello massimo e del numero di messaggi presenti per ogni vault_id.
- Vengono poi iterati tutti i messaggi, viene recuperato o creato nel caso in cui non ancora presente il TDigest per lo specifico vault_id letto e aggiunte le ore di funzionamento. Vengono poi aggiornati, se necessario, i valori minimi e massimi nella mappa e sommati i messaggi afferenti allo specifico vault_id.
- Dopo aver analizzato tutti i messaggi, per ogni vault_id presente nella mappa TDigest, vengono recuperati i valori minimi e massimi per poi poter calcolare i quintili.
- Vengono poi raggruppati i risultati ottenuti per ogni vault_id in una stringa in modo da essere poi riportati in output su file CSV.

È previsto quindi l'utilizzo della struttura dati probabilistica del TDigest, in modo da poter calcolare i percentili (0.25, 0.5, 0.75) delle ore di funzionamento (s9_power_on_hours) per ciascun vaultId in maniera efficiente, garantendo un basso utilizzo di memoria e tempi di calcolo rapidi. Vengono inoltre calcolati i valori di massimo, minimo e vengono conteggiate le occorrenze per il calcolo delle statistiche prima introdotte.

ANALISI DELLE PERFORMANCE

Per l'analisi delle performance, le query vengono eseguite su una macchina con un processore Intel 13th Gen i7-1355U 1.7GHz con 16 GB di RAM DDR4 4267 MHz. Le metriche raccolte durante l'esecuzione del programma sono quelle di throughput e di latenza, intesa come tempo o ritardo introdotto dal processamento dei dati real time effettuato da Flink. I valori di throughput vengono recuperati direttamente dalla dashboard offerta dall'interfaccia utente messa a disposizione da Flink. I valori di latenza vengono recuperati invece dal codice del Job durante l'esecuzione. In tutte le implementazioni spiegate precedentemente, viene inserito un punto di checkpoint in cui viene salvato l'istante di tempo subito prima che inizi il processamento dei dati in una finestra; ne viene poi inserito un secondo al termine delle operazioni effettuate in quella finestra. In questo modo, facendo la differenza tra i due, si ottiene il tempo impiegato da Flink per analizzare tutte le tuple presenti nella finestra ed effettuarne i processamenti del caso. Questo tempo è influenzato da due fattori principali:

- Durata della finestra: in base alla grandezza della finestra, i tempi di esecuzione cambiano, poiché più è grande la finestra considerata, più saranno le tuple presenti al suo interno e di conseguenza maggiori saranno le operazioni da compiere per Flink.
- Filtraggio precedente: in base al filtraggio effettuato nel momento in cui le tuple vengono lette da Flink, cambia il numero di messaggi che entrano a far parte di una finestra. Più sarà restrittivo il prefiltraggio, meno saranno i messaggi che verranno considerati buoni per la finestra e perciò meno sarà il tempo necessario a Flink per l'esecuzione.

QUERY 1

Di seguito vengono riportati i tempi di latenza media e di throughput ottenuti dall'esecuzione della query 1:

	Latenza		
	Min	**Avg**	**Max**
1 giorno	2 ms	21,43 ms	155 ms
3 giorni	21 ms	50 ms	81 ms
23 giorni	-	813 ms	-

	Throughput processamento			
		Min	**Avg**	**Max**
1 giorno	MBytes/s	15,51	193,65	344,71
	Message/s	74561,29	952269,21	1657571,4
3 giorni	MBytes/s	71,22	164,24	261,62
	Message/s	342675,32	782086,47	1258333,3
23 giorni	MBytes/s	-	67,02	-
	Message/s	-	307340,71	-

	Throughput		
	Task n.1	**Task n.2**	**Task n.3**
R_{in}/s	-	1125	180
B_{in}/s	-	1,15 MB/s	14,4 KB/s
R_{out}/s	2240	96	-
B_{out}/s	2,75 MB/s	19,2 KB/s	-

Nella tabella sopra, i task riportati sono i seguenti:

- Task n.1: recupera le tuple da Kafka
- Task n.2: effettua il filtraggio sui messaggi in ingresso, eliminando le tuple nulle e quelle che non hanno vault_id nel range desiderato
- Task n.3: effettua il processamento sulle tuple presenti nella finestra

Per quanto riguarda la latenza, analizzando i tempi finestra per finestra, troviamo il valore massimo di tempo nella prima finestra contenente le prime tuple streaming in arrivo nel sistema. Questo probabilmente dovuto al fatto che alla prima invocazione viene creato il file CSV d'output ed al setup/avvio del sistema Flink. Per i successivi valori troviamo una media di circa 11 ms nel caso di window da 1 giorno e di circa 28 ms nel caso di window da 3 giorni: questi valori mostrano che Flink riesce ad eseguire le query nelle window prefissate in tempi molto rapidi. Il tempo totale di query, di circa 25 minuti, è da ricondurre all'overhead introdotto da Kafka, sia nell'invio dei singoli record da parte del codice python, sia al ritardo volutamente introdotto per simulare la distanza tra un giorno e l'altro e all'overhead di comunicazione con Flink. Le considerazioni appena effettuate sono valide anche per le statistiche di seguito riportate.

QUERY 2

Di seguito vengono riportati i tempi di latenza media e di throughput ottenuti dall'esecuzione della query 2:

	Latenza		
	Min	**Avg**	**Max**
1 giorno	1	7,18 ms	79 ms
3 giorni	1 ms	4 ms	7 ms
23 giorni	-	20 ms	-

	Throughput processamento			
		Min	**Avg**	**Max**
1 giorno	KBytes/s	12,20	635,92	2107,03
	Message/s	75,949	3473,30	13000,0
3 giorni	KBytes/s	436,10	1102,97	2754,88
	Message/s	2714,28	6966,33	17000,0
23 giorni	KBytes/s	-	660,35	-
	Message/s	-	4100	-

	Throughput	
	Task n.1	**Task n.2**
R_{in}/s	-	< 1
B_{in}/s	-	25 B/s
R_{out}/s	< 1	-
B_{out}/s	< 1	-

Nella tabella sopra, i task riportati sono i seguenti:

- Task n.1: recupera le tuple da Kafka, trasforma queste in oggetti di tipo Message e filtra le tuple nulle e quelle che non hanno vault_id nel range desiderato
- Task n.2: effettua il processamento sulle tuple presenti nella finestra

QUERY 3

Di seguito vengono riportati i tempi di latenza media e di throughput ottenuti dall'esecuzione della query 3:

	Latenza		
	Min	**Avg**	**Max**
1 giorno	18 ms	46,40 ms	221 ms
3 giorni	44 ms	94 ms	153 ms
23 giorni	-	520 ms	-

	Throughput processamento			
		Min	**Avg**	**Max**
1 giorno	KBytes/s	13188,99	85347,65	161150,27
	Message/s	103239,82	763534,02	1261111,1
3 giorni	KBytes/s	56810.74	94797.40	131520.11
	Message/s	444483,66	726256,14	1029613,6
23 giorni	KBytes/s	-	111401,58	-
	Message/s	-	871917,30	-

	Throughput da Flink UI	
	Task n.1	**Task n.2**
R_{in}/s	-	361
B_{in}/s	-	26,7 KB/s
R_{out}/s	360	-
B_{out}/s	70,4 KB/s	-

Nella tabella sopra, i task riportati sono i seguenti:

- Task n.1: recupera le tuple da Kafka, trasforma queste in oggetti di tipo Message e filtra le tuple nulle e quelle che non hanno vault_id nel range desiderato
- Task n.2: effettua il processamento sulle tuple presenti nella finestra

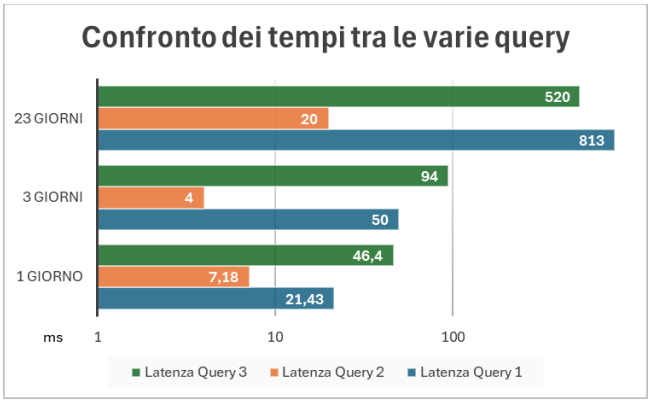
Osservazione: tutte le statistiche di throughput da Flink UI presentate, si riferiscono al momento di picco del sistema, non considerando quindi i valori di throughput inferiori che si hanno nel momento in cui si passa da un giorno fisico all'altro.

CONCLUSIONI

Nelle query presentate, analizzando i tempi di latenza, è possibile osservare che, in tutti i casi, sono presenti valori massimi molto più elevati del tempo medio di esecuzione, rappresentando quindi dei valori outliner. Analizzando in maniera più approfondita il problema, si è recuperato il tempo di esecuzione impiegato nel processare ogni singolo messaggio, e da questo studio è venuto fuori che è proprio l'elaborazione della prima finestra temporale giornaliera a riportare dei tempi ben superiori la media. A seguito di osservazioni, si è giunti alla conclusione, come anticipato, che questo è dovuto principalmente a due fattori: il primo è il delay introdotto dalla fase di setup/avvio di Flink; il secondo, probabilmente quello più impattante, è legato al fatto che l'elaborazione della prima finestra giornaliera include il tempo di creazione del file CSV per i risultati di output e perciò introduce un ritardo non indifferente.

Nella query n.2 inoltre è possibile osservare che i valori di throughput sono estremamente bassi e inferiori alle altre due query. Questo comportamento è dovuto alla presenza di operazioni di computazione all'interno della finestra molto onerose, come ad esempio aggregazione complessa dei messaggi e ordinamento dei risultati.

Confronto tempi di esecuzione

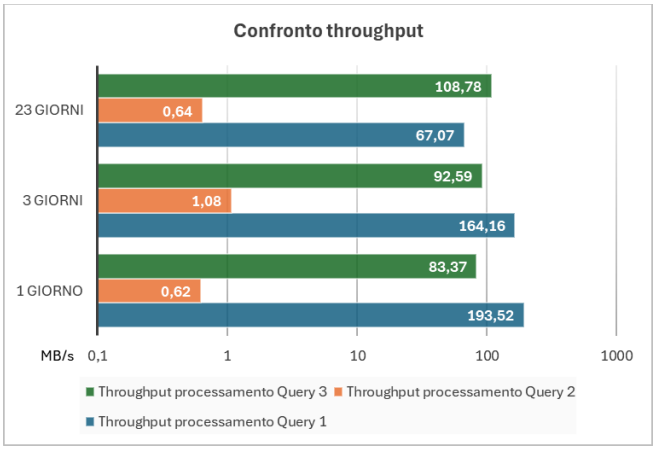


Dal grafico è possibile osservare il confronto tra i tempi di esecuzione medi delle varie query in ms. Risulta evidente come i tempi per il processamento della query n.2 sono nettamente inferiori ai tempi registrati nelle altre due query. Questo perché, come già detto, nella query in questione, è presente a monte del processamento vero e proprio, un processo di filtraggio estremamente restringente, che elimina la maggior parte dei messaggi, rendendo quindi il calcolo molto più veloce.

È inoltre possibile notare un'inversione di tendenza tra le query n.1 e n.3. In caso di utilizzo di finestra temporale di 1 o 3 giorni, il tempo di processamento della query n.1 risulta essere mediamente minore di quello richiesto dalla query n.3. In caso di utilizzo di finestra globale, basata su tutti i giorni del dataset considerato, risulta invece essere la query n.3 ad impiegare meno tempo. di quella n.1. Questo può essere dovuto al fatto che la terza query con finestra fissata a 23 giorni si trovi ad elaborare operazioni onerose come i calcoli dei percentili, su una grande quantità di dati.

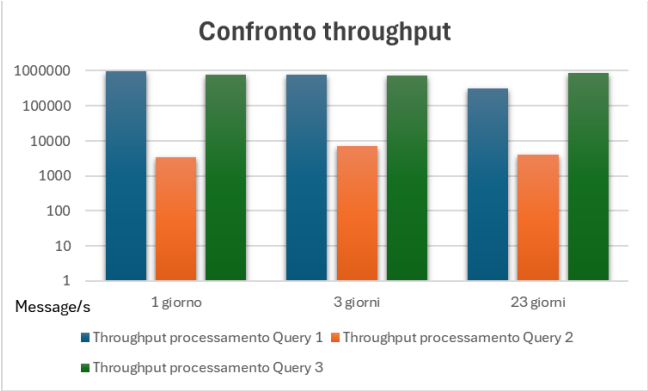
Inoltre, come era possibile immaginare, i tempi medi di esecuzione per tutte le query quando si considera una finestra di 23 giorni sono estremamente maggiori dei tempi riscontrati in caso di finestre di uno o tre giorni. Questo risultato lo si ottiene poiché maggiore è la durata della finestra, più saranno i messaggi compresi al suo interno e più saranno quindi quelli da elaborare la funzione di processamento.

Confronto throughput in MB/s



Nel secondo grafico sono invece confrontati i throughput delle tre query realizzate nelle varie finestre di esecuzione considerate in MB/s. Si nota come nella query n.2 il valore del throughput risulti notevolmente inferiore a quello ottenuto nelle altre due. Questo comportamento è dovuto, come detto precedentemente, alla presenza di operazioni di computazione all'interno della finestra molto onerose, come, ad esempio, aggregazione complessa dei messaggi e ordinamento dei risultati.

Confronto throughput in messaggi/s



Viene infine riportato il confronto tra i throughput considerati come numero di messaggi elaborati al secondo. Anche in quest'ultimo caso è possibile notare che il throughput della query n.2 è inferiore rispetto a quello ottenuto nelle altre due.

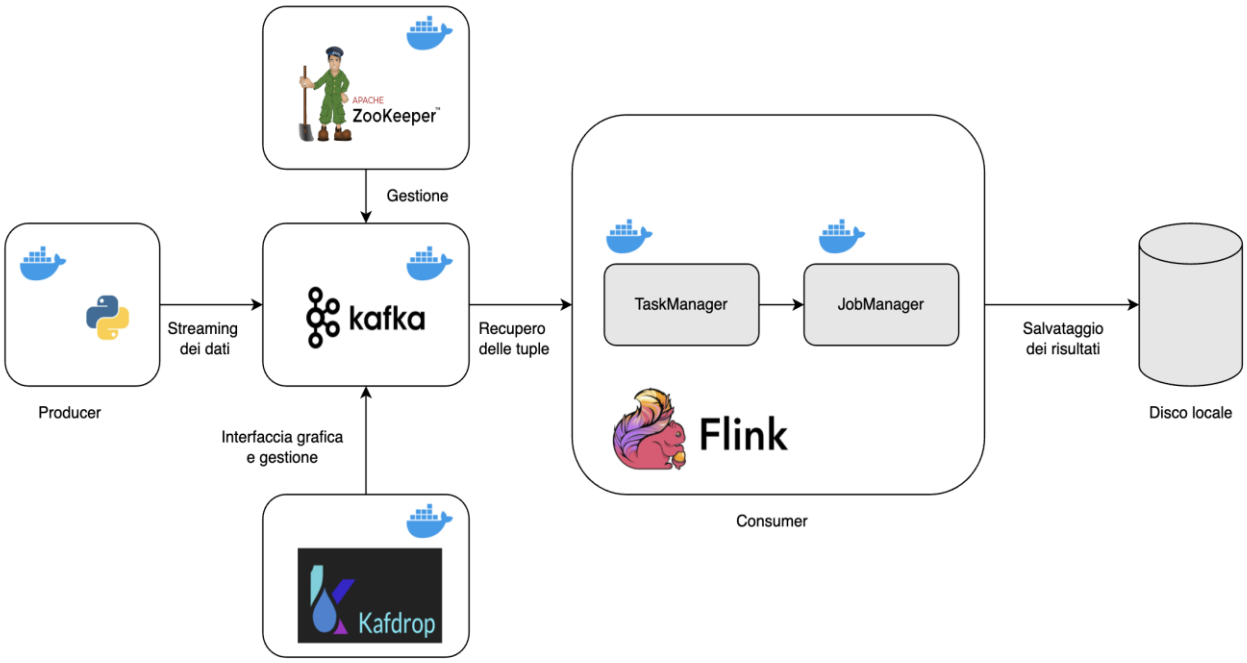


Figura 1: Architettura del sistema

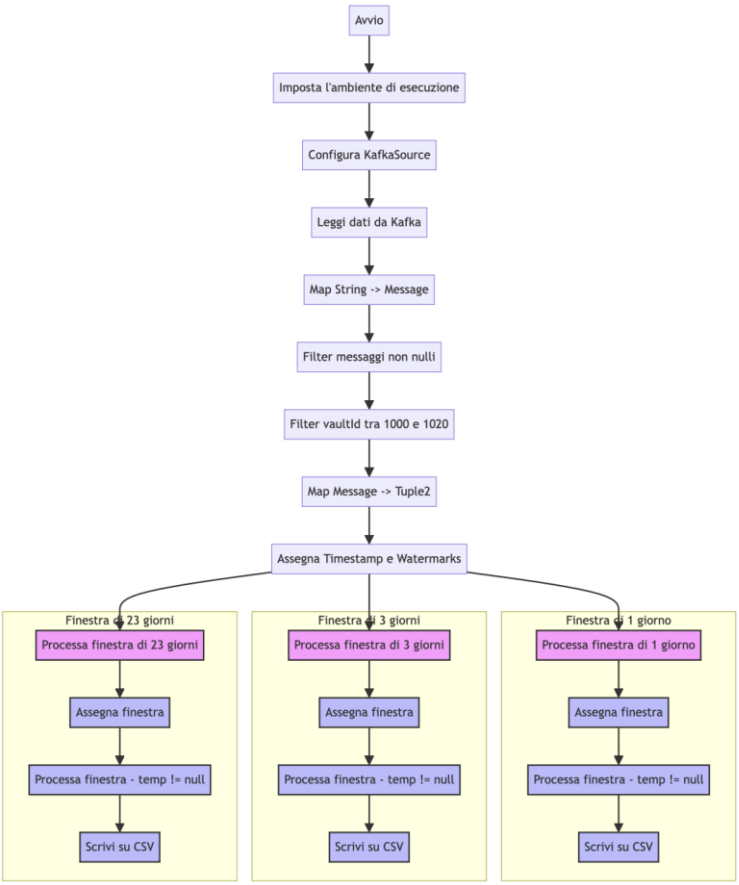


Figura 2: Query 1

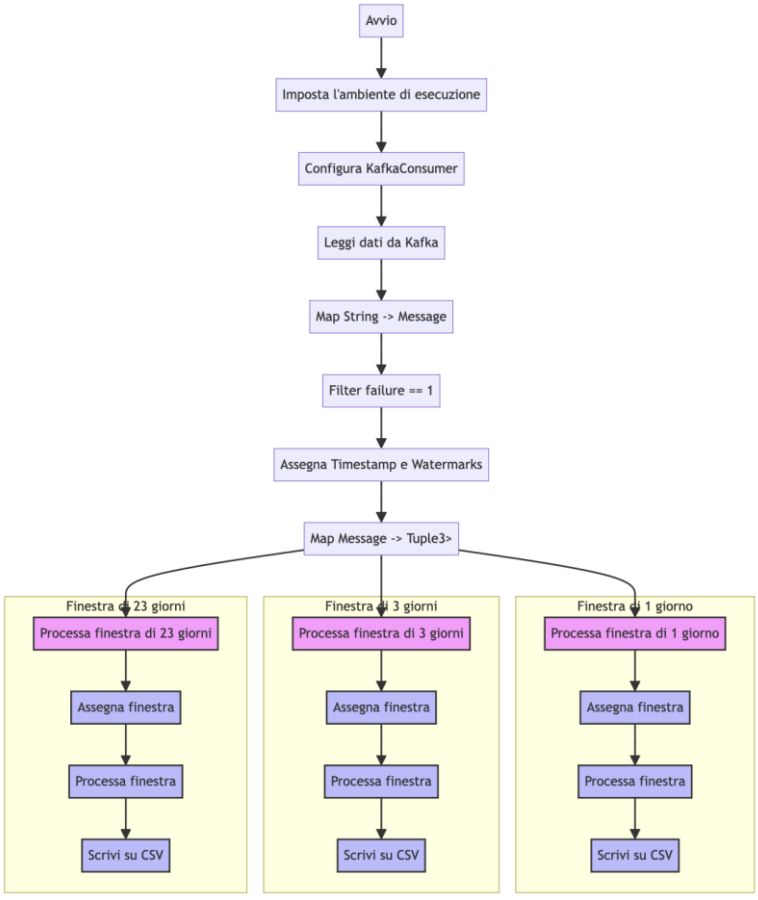


Figura 3: Query 3

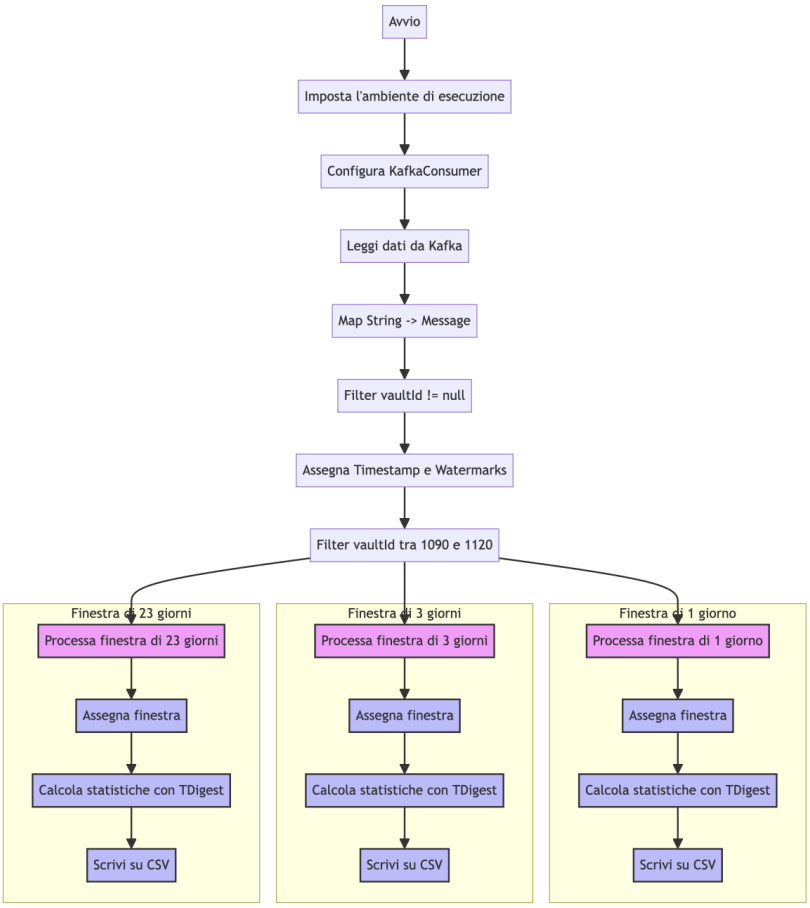


Figura 4: Query 3