SABD – Progetto 2

Esecuzione di Query Flink su dataset Blackblaze – S.M.A.R.T

Pieraldo Santurro
Facoltà di Ingegneria
Università di Roma Tor Vergata
Roma, Lazio, Italia
santurro.pieraldo99@gmail.com

ABSTRACT

L'obiettivo del progetto è quello di analizzare dei dati forniti da Blackblaze relativi alla telemetria di circa 200k hard disk in contenuti in alcuni data center. Per gli scopi del progetto si utilizza una versione ridotta del dataset indicato nel Grand Challenge della conferenza ACM DEBS 2024. Il dataset, inizialmente fornito batch in formato csv, viene reso disponibile in formato streaming mediante l'utilizzo di Kafka e processato secondo le tecniche di Stream Processing tramite Apache Flink

1. INTRODUZIONE

Preventivamente all'esecuzione del progetto è necessario effettuare alcuni step:

- Se la directory del progetto viene scaricata da GitHub, è necessario inserire nella directory *dataset,* il dataset originario in formato csv e rinominato *'dataset.csv'*. Lo script principale di esecuzione andrà ad eseguire un filtraggio delle colonne inutili ai fini del progetto mediante l'utilizzo delle API di Pandas, fornendo come risultato un file csv chiamato '*filtered_dataset.csv'.*
- Se la directory del progetto viene ottenuta con altre modalità, verificare se è presente il file *'dataset.csv'* all'interno della cartella *'dataset',* nel caso non ci sia, agire come nel punto sopra.
- All'interno della cartella *Flink Queries/* sono presenti le directory relative alle query da utilizzare per l'esecuzione del progetto, che verranno buildate mediante Maven durante l'esecuzione dello script principale del progetto
- Per il corretto funzionamento del progetto, sono necessarie le librerie python *panda*, *pandas kafka-python-ng*, *pandas pyarrow*. È possibile installarle mediante comando *pip*. In alternativa, è possibile creare un ambiente virtuale python all'interno della directory principale del progetto, ulteriori informazioni sono fornite nel README.md file del progetto.
- Il tempo di esecuzione del progetto è di circa 20 minuti, che comprendono il building iniziale delle librerie, delle queries da eseguire, e l'inserimento dei dati sul broker Kafka.

Per inizializzare il progetto aprire un terminale nella cartella principale ed eseguire il comando '*./start_project.sh*'. Questo eseguibile è responsabile del flusso totale dell'applicazione ed andrà chiuso solamente una volta terminata l'esecuzione. Durante l'esecuzione dello script principale del progetto, avverrà il building di ogni file .jar mediante Maven, e lo spostamento, tramite comando docker, dei file jar risultanti nella cartella *Flink/jobs/*. I file jar sono stati separati appositamente per permettere una facile modifica ai fini di analisi per ogni specifica Query. Per ulteriori informazioni e per consultare i comandi da utilizzare si può consultare il README.md file. Verranno aperti durante l'esecuzione dello script altri terminali in cui verranno eseguite le queries sottomettendole a Flink e l'inserimento dei dati sul Kafka Broker. Il documento è strutturato secondo punti relativi alla descrizione di componenti e frameworks utilizzati nell'applicazione, alla realizzazione delle query, all'analisi delle prestazioni ed infine alcune considerazioni finali sul sistema. Nella cartella *Scripts*, sono presenti eseguibili in linguaggio python per la graficazione dei risultati.

2. ARCHITETTURA E IMPLEMENTAZIONE

Si utilizzano vari framework, che, una volta fatti comunicare tra loro, produrranno i risultati richiesti. L'esecuzione principale è sul filesystem locale, governata e gestita dalla virtualizzazione in ambiente docker dei container che andremo a creare. Ogni framework viene quindi hostato su un container docker ed è capace di comunicare con gli altri container grazie alla rete virtuale sostenuta da docker compose. I componenti architetturali del sistema sono:

- Kafka (confluentinc/cp-kafka:latest)
- Zookeper (confluentinc/cp-zookeper:latest)
- Kafdrop (obsidiandynamics/kafdrop)[1]
- Apache Flink (flink:latest)

2.1 Kafka

Apache Kafka è una piattaforma per il data streaming distribuita che permette di pubblicare, sottoscrivere, archiviare ed elaborare flussi di record in tempo reale. È progettata per gestire interi flussi

di dati provenienti da più sorgenti distribuendoli a più consumatori. Verrà utilizzata un singolo nodo *kafka-broker* su container docker. Nel progetto Kafka legge le righe del dataset preventivamente filtrato in formato csv '*filtered_dataset.csv'* riga per riga e inserisce i dati letti in un topic '*batch-dataset'*. La scelta di questo tipo di inserimento, che richiede tra i 18 ed i 19 minuti per l'inserimento totale del dataset nel topic, è stata puramente personale e basata sulla voglia di immaginare una situazione progettuale il più reale possibile: si sta parlando di un dataset che effettua analisi giornaliere e cattura metriche relativamente ad Hard Disks, quindi, immaginando di avere uno streaming continuo di molte misurazioni nell'arco di molti giorni (nel nostro dataset ridotto abbiamo un arco di analisi di 23 giorni), è stato ritenuto opportuno modellizzare il dataset fornito in batch come un flusso continuo di dati. Sono state effettuate anche prove per inserimento in Kafka più veloci basate su batch di dati più o meno elevati, ma la trattazione ed il progetto si sarebbero ridotti ad un'analisi in stile batch, seppur con batch continui e ridotti. Date le tempistiche di inserimento, sono state fatte le seguenti considerazioni relative ad un ipotetico event time (utilizzato per la terza Query):

- Tempo simulato totale: 23 giorni
  - 23 giorni corrispondono a 33120 minuti
- Tempo reale totale: 18 minuti
- Fattore di accelerazione:
  - $\frac{Tempo\ simulato\ totale}{Tempo\ reale\ totale} = 1840$
- Un giorno nel tempo reale:
  - $\frac{Minuti\ in\ un\ giorno=1440}{1840} = 49.96\ secondi$
- Un'ora nel tempo reale:
  - $\frac{Minuti\ in\ un'ora=60}{1840} = 1.96\ secondi$
- Mezz'ora nel tempo reale:
  - $\frac{30}{1840} = 0.98\ secondi$

Queste metriche saranno utili per la Query 3 e nel caso si voglia modificare la finestra utilizzata per le altre query, che di base sarà di tipo Tumbling Window di 5 secondi.

2.2 Zookeper

Apache ZooKeeper è un framework utilizzato da diversi progetti open-source per garantire una maggiore affidabilità delle risorse del cluster e ulteriori servizi di configurazione del cluster stesso, svolgendo servizi centralizzati per il mantenimento dei metadati delle varie applicazioni in esecuzione nella sua rete. Le principali funzionalità che ZooKeeper svolge nel cluster proposto sono:

- Elezione del Kafka Broker leader, assicurando che ulteriori broker prendano il ruolo di leader nel caso in cui il corrente leader subisse dei fallimenti
- Topic Configuration: mantiene la lista di tutti i topic presenti incluse le informazioni relative al numero di partizioni per ogni topic, locazione delle repliche e ulteriori dettagli.

2.3 Kafdrop

Kafdrop è una web UI che permette all'utente la visualizzazione dei vari Kafka Topic presenti nel cluster e le loro principali informazioni come il numero di broker, il numero di partizioni, i consumer attivi su un determinato topic. Infine, kafdrop permette di visualizzare i messaggi scritti sul topic. È stata utilizzata per agevolare lo sviluppo dell'applicazione e contemporaneamente per offrire una rapida via di accesso e una rapida visualizzazione dello stato dei topic kafka.

2.4 Flink

Apache Flink è un sistema di processamento distribuito per computazione stateful su streams di dati bounded e unbounded, quindi progettato sia per batch che stream processing. Flink è stato progettato per essere un sistema in grado di offrire un alto throughput, ed una bassa latenza. Si utilizza un'immagine docker per il jobmanager, ed un'altra immagine docker per il taskmanager. Si è deciso di utilizzare un unico taskmanager, ognuno con massimo 12 task slot assegnabili, in quanto si è visto sperimentalmente che il jobmanager tende a schedulare i task su un unico taskmanager; considerando il grado di parallelismo richiesto dall'applicazione non è stato quindi necessario l'utilizzo di più taskmanager. Per questo progetto, si è deciso di utilizza uno stream processing basato su Tumbling Window fisse di 5 secondi, liberamente riconfigurabili, per la Query 1 e la Query 2, e Tumbling Window basate sui risultati ottenuti nel Paragrafo 2.1 per la Query 3.

3. QUERY

Vediamo di seguito le scelte effettuate durante la progettazione delle query. Per la realizzazione delle tre Queries, si è deciso di creare un file .jar per ogni tipologia di grandezza della finestra considerata: quindi per la Query 1 e la Query 2 ci saranno tre diverse directory ciascuna, per separare l'esecuzione in base all'event time considerato; per la Query 1 e la Query 2 avremo le versioni relative ad un giorno, a tre giorni e per l'intero dataset. Nel caso specifico della Query 1, se i dati relativi alla temperatura non sono disponibili nel messaggio letto, quel messaggio viene direttamente scartato. Nei paragrafi successivi si vedranno le differenze di implementazione. In linea generale, si utilizza una lettura da topic impostata con Tumbling Window fisse di cinque secondi. Per la Query 3 invece, si fornisce una sola versione, utilizzando una configurazione che legge l'intero dataset con finestra sempre di tipo Tumbling, ma liberamente configurabile prima dell'esecuzione; per le misurazioni sulla Query 3 sono state utilizzate le finestre ricavate nel Paragrafo 2.1. Un'implementazione della terza Query che consente un filtraggio giornaliero può essere semplicemente ottenuta agendo come nella Query 1, che come vedremo, effettua un filtraggio su base giornaliera ed esegue un calcolo delle statistiche necessarie. Ogni filtraggio e ogni finestra per ogni Query può essere cambiato per

scopi di analisi diversi, nell'esecuzione standard per gli event time giornalieri si è scelto il giorno 2023-04-01, e nell'implementazione per tre giorni 2023-04-01, 2023-04-02 e 2023-04-03, nella Query 3, i risultati forniti sono per una finestra di due secondi, quindi relative a un event time di un giorno (relativamente ai tempi del progetto). Se si utilizza lo script principale di esecuzione per sottomettere i file .jar a Flink, ogni file .jar verrà eseguito a cinque secondi di distanza l'uno dall'altro mediante comando *sleep* da terminale, per un totale di sette jobs in esecuzione contemporaneamente, sia nel caso in cui il topic Kafka da cui vengono letti i dati sia completamente popolato, sia nel caso in cui sia in atto l'operazione di popolamento del suddetto topic. Le funzioni realizzate per ogni Query, implementano l'interfaccia *AllWindowFunction*[2] di Flink, in particolare facendo l'override del metodo *apply*, che accetta come parametri in ingresso la finestra temporale su cui calcolare le statistiche, una collezione di record su cui effettuare le operazioni ed un collettore su cui restituire i risultati; nel proseguo della trattazione vedremo come sostanzialmente verrà cambiato il modo di implementare le entità passate in ingresso e in uscita all'interfaccia realizzata, oltre che le funzionalità interne, cercando di mantenere la stessa logica per ogni Query. I risultati sono scritti su relativi file csv secondo tecniche diverse spiegate nei paragrafi successivi per ogni Query. Il calcolo di latenza e throughput rimane sostanzialmente lo stesso per ogni versione delle Query, intendendo la latenza come '*tempo che intercorre tra quando i dati sono disponibili e quando vengono processati* e il throughput come '*quantità di dati processati nella finestra temporale'*.

3.1 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.

3.1.1 Implementazione per un giorno

Si inizializza l'ambiente di esecuzione Flink e il consumer Kafka, in modo che venga letto dal topic '*batch-dataset*' a partire dall'inizio. Il flusso di dati in ingresso, cioè i messaggi in formato Json, sono filtrati per includere solamente quelli con *vault_id* compreso tra 1000 e 1020, date pari a 2023-04-01, e s194_temperature_celsius diverso da zero. Viene applicata una finestra di elaborazione di cinque secondi ai dati filtrati. La finestra è di tipo "tumbling", quindi non si sovrappone. Per ogni finestra di cinque secondi, il codice raccoglie i valori di s194_temperature_celsius per ciascun vault_id, calcola la media, la deviazione standard, la latenza e il throughput, e scrive questi valori su un file CSV. Il throughput è calcolato per ogni finestra temporale di cinque secondi, dividendo il numero di record processati (*count*) per la durata della finestra in secondi. La latenza viene calcolata come la differenza tra il timestamp di inizio della finestra e il timestamp corrente al momento dell'elaborazione dei dati. Di seguito il calcolo di media e deviazione standard:

- Media: calcolata sommando tutti i valori di s194_temperature_celsius e dividendo per il numero di valori.
  - $\mathbf{mean} = \frac{1}{N}\sum_{i=1}^{N} x_i$
- Deviazione Standard:
  - $\mathbf{stDev} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - mean)^2}$

3.1.2 Implementazione per tre giorni

Le operazioni effettuate sono le medesime della Query 1 per un solo giorno, il cambiamento sostanziale risiede nel filtraggio dei messaggi Json dal topic '*batch-dataset*', in modo che includa un range di tre giorni (riconfigurabili prima della ricompilazione del progetto). Anche qui si utilizza una finestra di tipo Tumbling di 5 secondi, e come nella Query per un solo giorno, si fanno le stesse operazioni di calcolo di media, varianza, latenza e throughput. Anche qui i risultati vengono forniti in un file csv.

3.1.3 Implementazione per l'intero dataset

Il calcolo di media, varianza, latenza e throughput avviene nello stesso modo dei due paragrafi precedenti. Il vero cambiamento risiede nella mancanza di filtraggio giornaliero, cioè, vengono accettati tutti i giorni letti dal topic kafka di ingresso. Di base la finestra utilizzata rimane di cinque secondi, ma è possibile cambiarla con i valori menzionati nel paragrafo 2.1 per un'analisi basata su un giorno, e tre giorni relativamente al tempo di esecuzione del nostro dataset.

3.2 Query 2

Calcolare la classifica aggiornata in tempo reale dei 10 vault che registrano il piu alto numero di falli- ` menti nella stessa giornata. Per ogni vault, riportare il numero di fallimenti ed il modello e numero seriale degli hard disk guasti.

3.2.1 Implementazione per un giorno

Si inizializza l'ambiente di esecuzione Flink e il consumer Kafka, in modo che venga letto dal topic '*batch-dataset*' a partire dall'inizio. Il flusso di dati viene filtrato per includere solo i record con failure pari a 1 e date pari a 2023-04-01. Viene applicata, come nella Query 1 una tumbling window di cinque secondi. Per ogni finestra quindi, il codice raccoglie i valori di failure per ciascun vault_id, calcola il numero di fallimenti per ogni vault, ordina i vault in base al numero di fallimenti e seleziona i primi 10 vault con il maggior numero di fallimenti. Inoltre, calcola la latenza e il throughput per ogni finestra. L'aggregazione dei dati in ingresso avviene per mezzo di *globalVaultData*, una struttura dati di tipo Hash Map[3], che utilizza *vault_id* come chiave e *vault_info* come valore, dove *vault_info* non è altro che una classe helper che

contiene informazioni sul numero di fallimenti e una lista di modelli e numeri di serie degli hard disk guasti per ciascun vault. L'entità *globalVaultData* viene utilizzata per tenere traccia delle informazioni sui fallimenti (failures) di ciascun vault durante l'esecuzione del job Flink. Questa mappa globale viene aggiornata continuamente con i dati raccolti da ogni finestra temporale e viene utilizzata per mantenere uno stato aggregato dei fallimenti per ciascun vault. Il calcolo della latenza e del throughput segue la stessa logica della Query 1. I risultati saranno scritti in un file csv contenente la classifica aggiornata e giornaliera dei vault_id con il maggior numero di fallimenti.

3.2.2 Implementazione per tre giorni

Come nella versione per un giorno si inizializza l'ambiente di esecuzione Flink e il consumer Kafka. La differenza fondamentale risiede nel filtraggio che adesso accetta tre giorni (configurabili) in ingresso e nell'utilizzo più intensivo dell'aggregazione dei dati catturati in ingresso. La Hash Map utilizzata nell'implementazione per un solo giorno in questa versione viene estesa, utilizzando una mappa annidata[4] per ciascun giorno, quindi avremo una mappa generale, con chiavi relative al giorno, e per ogni giorno una mappa interna che avrà, come nel paragrafo precedente, come chiave *vault_id* e *vault_info* come valore. A questo punto il calcolo dei risultati avviene su base giornaliera, e sempre su base giornaliera, avverrà la scrittura dei risultati: verrà creato un file *output_yyyy-mm-dd.csv* per ogni giorno. Il calcolo di latenza e throughput avviene nel medesimo modo della versione per un solo giorno.

3.2.3 Implementazione per l'intero dataset

Come nelle versioni precedenti avviene l'inizializzazione dell'ambiente Flink e del consumer Kafka. La differenza fondamentale rispetto alla versione per tre giorni risiede nell'accettare tutti i giorni del dataset in ingresso; quindi, non avendo un filtraggio giornaliero ma catturando comunque i valori relativi al campo *date*. Per quanto riguarda l'aggregazione dei dati rimane sostanzialmente identica alla versione della Query 2 per tre giorni, è stata tuttavia effettuata una forzatura per consentire l'aggregazione dei valori e la stampa su file di output dell'ultimo giorno del dataset, il 2024-04-23. Questa operazione è stata resa necessaria, inizialmente, perché sembrava che il codice non catturasse correttamente l'ultimo giorno disponibile; dopo diversi tentativi invece, il codice non stampava un risultato per l'ultimo giorno semplicemente perché non sono presenti fallimenti, di conseguenza i dati letti banalmente non passano il filtraggio iniziale. Per chiarezza si è preferito lasciare l'implementazione corrente per sottolineare questo risultato, ottenuto anche mediante uno script python con utilizzo delle API Pandas, per avere una doppia conferma. Come nella versione precedente, le operazioni vengono effettuate su base giornaliera e i risultati sono ottenuti nello stesso identico modo della versione per tre giorni, così come la scrittura su file csv. Il calcolo di latenza e throughput rimane il medesimo.

3.3 Query 3

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 quantita di memoria occupata al prezzo di una minore accuratezza, e.g., [4, 6, 7, 3, 5].

3.3.1 Implementazione

Nella versione base, il codice utilizza finestre di 2 secondi, cioè corrispondenti ad un'ora di event time; tale parametro, come già accennato, è liberamente configurabile prima dell'esecuzione, e sarà necessario rieseguire le operazioni di build della Query. Si crea un'istanza dell'ambiente di esecuzione di Flink e vengono impostate le proprietà di connessione per Kafka. I messaggi vengono letti dal topic '*batch*-dataset' e viene effettuato un filtraggio per accettare in ingresso solamente i record con vault_id compreso tra 1090 e 1120. Per le richieste della Query, viene creata una classe *CalculateStatistics*, responsabile del calcolo di massimo, del minimo, e i relativi percentili, delle ore di funzionamento degli hard disks. Per il calcolo, vengono create due Hash Map, *lastValues* e *vaultData* per memorizzare rispettivamente gli ultimi valori di *s9_power_on_hours* per ciascun hard disk (identificato da *serial_number*) e le liste di *s9_power_on_hours* per ciascun *vault_id*. Per ogni record che passa la fase iniziale di filtraggio, quindi, viene effettuato un parsing del Json, viene aggiornato un contatore, si estraggono *serial_number*, *s9_power_on_hours* e *vault_id* dai record, viene aggiornata la mappa di *vaultData* per aggiungere il valore di *s9_power_on_hours* alla lista corrispondente al *vault_id*. Adesso, per ogni *vault_id* nella mappa *vaultData* vengono calcolati i valori del minimo, del massimo e dei relativi percentili utilizzando *MergingDigest*[5]. I risultati vengono inseriti in un *csvBatch* e raccolti tramite il collettore *out*. A questo punto viene eseguita la scrittura sul file di output, che cambierà nome a seconda della grandezza della finestra. Il calcolo della latenza e del throughput segue la stessa logica di tutte le versioni della Query 1 e della Query 2.

3.4 Tempi di esecuzione

Dato che si tratta di un progetto scritto su base Docker Compose con container locali ed utilizzando Flink, che è estremamente veloce, i tempi di latenza sono sostanzialmente nulli; nei file risultanti viene indicato infatti il tempo di *5000ms* per quasi tutte le finestre, che sarebbe il valore della Tumbling Window utilizzata. Questo risultato è in linea con quanto progettato, dato che le operazioni vengono eseguite quando la finestra di interesse scorre nel tempo. Per quanto riguarda il Throughput, è possibile che in alcuni file di output venga indicato uno spike iniziale di dati processati, questo perché, dato che i jobs vengono eseguiti in sequenza dopo l'invio iniziale dei messaggi dal local file system al

topic Kaka, per le prime finestre è normale avere più dati da processare rispetto alle finestre successive. Di base, Flink è un framework estremamente veloce ed efficiente, per la completa esecuzione dei jobs si sono impiegati tra i 18 ed i 19 minuti, che è il tempo che impiega Kafka per accettare le righe del dataset in ingresso, quindi il bottleneck dell'applicazione, risulta essere Kafka e le strategie di inserimento nei topic. Nel capitolo 6 sono indicati alcuni grafi di esempio relativi alle tempistiche delle query.

3.5 Risultati

Come già accennato nei paragrafi precedenti, ogni Query ha il suo modo specifico di salvare i risultati. In ogni caso tutti i risultati vengono salvati durante l'esecuzione in formato csv nella cartella *Flink/Results/*. Nell'appendice sono indicati alcuni esempi di output per ogni versione di ogni query. I grafici visualizzati nell'appendice sono relativi alle versioni delle Query per tutto il dataset.

4. CONCLUSIONI

Il progetto è stato eseguito tramite macchina virtuale WSL2 su un calcolatore che possiede le seguenti caratteristiche:

- Processore: 12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz
- RAM: 16,0 GB (15,7 GB utilizzabile)
- OS: Windows 11 Home – 23H2
- WSL2: Ubuntu 22.04.3 LTS

4.1 Difficoltà riscontrate

Le maggiori difficoltà riscontrate sono principalmente relative all'implementazione delle Queries utilizzando il codice Java. È stata necessaria, infatti, una buona ricerca sui vari utilizzi che possono essere fatti delle librerie utilizzate nel progetto, sia per le librerie riguardanti Flink, sia per le librerie utilizzate nell'aggregazione dei dati, in particolare per le Hash Maps e per la modifica dei metodi messi a disposizione da Flink per utilizzare le funzionalità del framework. L'architettura non è stata particolarmente difficile da implementare, grazie ovviamente a Docker Compose. L'ingegnerizzazione del progetto è stata complessivamente abbastanza sfidante, in particolare nella scelta di quale tattica utilizzare per il trattamento delle finestre di cattura dei dati in Flink, e nella scelta delle librerie da utilizzare e la loro versione.

4.2 Miglioramenti

Esecuzione delle Query su altri framework di Data Processing come Kafka-Streams e Spark Stream e confronto dei risultati e dell'analisi di latenza e throughput. Un'altra interessante variante potrebbe essere quella di utilizzare PyFlink, un set di librerie in linguaggio Python per Flink, sulla stessa impronta di PySpark. L'immagine ufficiale Docker utilizzata per questo progetto non possiede PyFlink installato al suo interno, di conseguenza potrebbe essere necessario l'utilizzo di un'altra immagine Docker o del building di una propria immagine personale. Al fine di questo possibile futuro miglioramento è stata realizzata da me un'immagine Docker che estende *flink:latest* installando una versione Python compatibile e le relative librerie utilizzabili; tale immagine è liberamente scaricabile al link: https://github.com/Pi3raldoSanturro/PyFlink

5. RIFERIMENTI

[1] https://medium.com/@jeyagan/kafdrop-kafka-web-ui-quick-start-example-42beb5efd7b7

[2] https://nightlies.apache.org/flink/flink-docs-release-1.13/api/java/org/apache/flink/streaming/api/functions/windowing/AllWindowFunction.html

[3] https://www.w3schools.com/java/java_hashmap.asp
https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/
https://www.geeksforgeeks.org/difference-between-arraylist-and-hashmap-in-java/

[4] https://www.baeldung.com/java-nested-hashmaps

[5] https://stackoverflow.com/questions/2837311/incremental-way-of-counting-quantiles-for-large-set-of-data

6. APPENDICE: Grafici e Risultati

6.1 Risultati Query 1

window,vault_id,count,mean_s194,stddev_s194,latency,throughput
1719670290000 - 1719670295000,1000,54,22.27777777777778,2.6695586170519916,5001,243.4
1719670290000 - 1719670295000,1001,59,17.93220338983051,1.7550354362124403,5001,243.4
1719670290000 - 1719670295000,1002,59,22.47457627118644,2.181244647050028,5001,243.4
1719670290000 - 1719670295000,1003,60,17.983333333333334,1.7841119795448814,5001,243.4
1719670290000 - 1719670295000,1004,64,19.9375,3.020114857087392,5001,243.4
1719670290000 - 1719670295000,1005,62,21.467741935483872,2.4997918747915144,5001,243.4
1719670290000 - 1719670295000,1006,57,23.140350877192983,3.6822885546480753,5001,243.4
1719670290000 - 1719670295000,1007,65,26.123076923076923,4.619382883551879,5001,243.4
1719670290000 - 1719670295000,1008,49,29.081632653061224,3.5504339182575095,5001,243.4
1719670290000 - 1719670295000,1009,68,28.264705882352942,3.1744274474898875,5001,243.4
1719670290000 - 1719670295000,1010,62,22.661290322580644,2.545987028240887,5001,243.4
1719670290000 - 1719670295000,1011,48,26.666666666666668,6.752571526627632,5001,243.4
1719670290000 - 1719670295000,1012,58,30.482758620689655,7.495381058865393,5001,243.4
1719670290000 - 1719670295000,1013,62,26.516129032258064,4.015060098710093,5001,243.4
1719670290000 - 1719670295000,1014,55,26.10909090909091,5.8236905739253055,5001,243.4
1719670290000 - 1719670295000,1015,60,26.45,3.81411850890871,5001,243.4
1719670290000 - 1719670295000,1016,54,26.925925925925927,4.829890926810191,5001,243.4
1719670290000 - 1719670295000,1017,58,28.379310344827587,5.722995315955161,5001,243.4
1719670290000 - 1719670295000,1018,57,27.087719298245613,3.565333584719575,5001,243.4
1719670290000 - 1719670295000,1019,55,28.672727272727272,6.09337806133405,5001,243.4
1719670290000 - 1719670295000,1020,51,26.15686274509804,5.828841572662642,5001,243.4
1719670395000 - 1719670400000,1000,23,22.347826086956523,3.0447202434802776,5001,91.4
1719670395000 - 1719670400000,1001,21,17.952380952380953,1.6754868653103983,5001,91.4
1719670395000 - 1719670400000,1002,30,22.533333333333335,2.0612833111653743,5001,91.4
1719670395000 - 1719670400000,1003,21,18.095238095238095,1.8232561637215854,5001,91.4

Figura 1-Risultati Query1_1_day

window,vault_id,count,mean_s194,stddev_s194,latency,throughput
1719670420000 - 1719670425000,1000,260,22.203846153846154,2.3189819996698766,5000,1096.0
1719670420000 - 1719670425000,1001,264,18.265151515151516,1.8105743625981519,5000,1096.0
1719670420000 - 1719670425000,1002,291,22.649484536082475,2.448260075548999,5000,1096.0
1719670420000 - 1719670425000,1003,267,18.149812734082396,2.192947062485563,5000,1096.0
1719670420000 - 1719670425000,1004,262,19.713740458015266,2.837541763724762,5000,1096.0
1719670420000 - 1719670425000,1005,262,21.31297709923664,2.732568541050091,5000,1096.0
1719670420000 - 1719670425000,1006,250,22.728,3.897437106612498,5000,1096.0
1719670420000 - 1719670425000,1007,286,25.874125874125873,4.722905252714027,5000,1096.0
1719670420000 - 1719670425000,1008,268,29.45149253731343,4.100586690361294,5000,1096.0
1719670420000 - 1719670425000,1009,260,27.726923076923075,3.3337502204790894,5000,1096.0
1719670420000 - 1719670425000,1010,265,22.09811320754717,2.694122923477452,5000,1096.0
1719670420000 - 1719670425000,1011,246,26.378048780487806,6.670479030900667,5000,1096.0
1719670420000 - 1719670425000,1012,263,30.315589353612168,7.465206005080483,5000,1096.0
1719670420000 - 1719670425000,1013,234,26.641025641025642,3.9698899605011517,5000,1096.0
1719670420000 - 1719670425000,1014,268,26.003731343283583,5.343079031772211,5000,1096.0
1719670420000 - 1719670425000,1015,255,26.298039215686273,4.046469785374941,5000,1096.0
1719670420000 - 1719670425000,1016,250,26.824,4.853969921620858,5000,1096.0
1719670420000 - 1719670425000,1017,259,28.08880308880309,5.370790990599139,5000,1096.0
1719670420000 - 1719670425000,1018,256,27.1484375,3.930231075724906,5000,1096.0
1719670420000 - 1719670425000,1019,246,27.79268292682927,4.941292262624005,5000,1096.0
1719670420000 - 1719670425000,1020,268,26.496268656716417,5.543247966434102,5000,1096.0
1719670425000 - 1719670430000,1000,57,22.07017543859649,2.3310237877477484,5001,199.6
1719670425000 - 1719670430000,1001,52,18.423076923076923,1.9840339040813741,5001,199.6
1719670425000 - 1719670430000,1002,41,22.024390243902438,2.300453877571043,5001,199.6
1719670425000 - 1719670430000,1003,38,18.842105263157894,2.367836185095156,5001,199.6

Figura 2-Risultati Query1_3_days

```
window,vault_id,count,mean_s194,stddev_s194,latency,throughput
1719670425000 - 1719670430000,1000,317,22.179810725552052,2.3217194138677457,5001,1296.8
1719670425000 - 1719670430000,1001,316,18.29113924050633,1.8411739426086813,5001,1296.8
1719670425000 - 1719670430000,1002,332,22.572289156626507,2.439178868993142,5001,1296.8
1719670425000 - 1719670430000,1003,305,18.236065573770492,2.2272553709674514,5001,1296.8
1719670425000 - 1719670430000,1004,308,19.7012987012987,2.8137645673808005,5001,1296.8
1719670425000 - 1719670430000,1005,307,21.257328990228014,2.7327611943207963,5001,1296.8
1719670425000 - 1719670430000,1006,309,22.57281553398058,3.8447255723250375,5001,1296.8
1719670425000 - 1719670430000,1007,321,25.822429906542055,4.738283962103057,5001,1296.8
1719670425000 - 1719670430000,1008,317,29.533123028391167,4.122303040808419,5001,1296.8
1719670425000 - 1719670430000,1009,297,27.64983164983165,3.340186133436527,5001,1296.8
1719670425000 - 1719670430000,1010,317,22.082018927444796,2.738824443236922,5001,1296.8
1719670425000 - 1719670430000,1011,283,26.374558303886925,6.562635647539818,5001,1296.8
1719670425000 - 1719670430000,1012,315,30.16190476190476,7.532099110261543,5001,1296.8
1719670425000 - 1719670430000,1013,289,26.737024221453286,4.089827806981952,5001,1296.8
1719670425000 - 1719670430000,1014,318,25.965408805031448,5.14830969905723,5001,1296.8
1719670425000 - 1719670430000,1015,294,26.156462585034014,4.064447951021061,5001,1296.8
1719670425000 - 1719670430000,1016,301,26.627906976744185,4.774073663045416,5001,1296.8
1719670425000 - 1719670430000,1017,304,28.14144736842105,5.4231829855351865,5001,1296.8
1719670425000 - 1719670430000,1018,310,27.10322580645161,3.887405957453338,5001,1296.8
1719670425000 - 1719670430000,1019,306,27.76143790849673,5.016829509219544,5001,1296.8
1719670425000 - 1719670430000,1020,318,26.569182389937108,5.609958011541817,5001,1296.8
1719670430000 - 1719670435000,1000,49,22.244897959183675,2.5114481735371688,5000,189.2
1719670430000 - 1719670435000,1001,41,18.024390243902438,1.4564876709105967,5000,189.2
1719670430000 - 1719670435000,1002,40,22.775,2.6219029348928995,5000,189.2
1719670430000 - 1719670435000,1003,47,18.93617021276596,2.823942131580776,5000,189.2
```

Figura 3-Risultati Query1_all_days

6.2 Risultati Query 2

```
window,latency,throughput,vault_id1,failures1,model_serial_list1,...,vault_id10,failures10,model_serial_list10
1719670430000,5001,0.8,1047,1,[ST4000DM000, Z304NT6A],1145,1,[HGST HUH721212ALE604, 8HGXNP2H],1099,1,[ST12000NM001G,
ZTN0AGRW],1055,1,[ST8000NM0055, ZA1814BV], , , , , , , , , , , , , , , , , , , , , , , ,
1719670440000,5000,0.2,1106,1,[TOSHIBA MG07ACA14TA, 70K0A08ZF97G],1047,1,[ST4000DM000, Z304NT6A],1145,1,[HGST HUH721212ALE604,
8HGXNP2H],1099,1,[ST12000NM001G, ZTN0AGRW],1055,1,[ST8000NM0055, ZA1814BV], , , , , , , , , , , , , , , , , , , ,
1719670445000,5000,0.2,1106,1,[TOSHIBA MG07ACA14TA, 70K0A08ZF97G],1047,1,[ST4000DM000, Z304NT6A],1145,1,[HGST HUH721212ALE604,
8HGXNP2H],1099,1,[ST12000NM001G, ZTN0AGRW],1118,1,[HGST HUH721212ALN604, AAG68EWH],1055,1,[ST8000NM0055,
ZA1814BV], , , , , , , , , , , , , , , ,
```

Figura 4-Risultati Query2_1_day

```
window,latency,throughput,vault_id1,failures1,model_serial_list1,...,vault_id10,failures10,model_serial_list10
1719670520000,5000,0.4,1131,1,[ST12000NM0008, ZHZ2JJMP],1103,1,[ST12000NM001G,
ZTN0AKND], , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,
1719670530000,5000,0.4,1042,1,[ST8000NM0055, ZA132DWE],1131,1,[ST12000NM0008, ZHZ2JJMP],1103,1,[ST12000NM001G,
ZTN0AKND],1055,1,[ST8000NM0055, ZA16K94X], , , , , , , , , , , , , , , , , , , , , , , ,
```

Figura 5-Risultati Query2_3_days per 2024-04-03

```
1719671590000,5000,0.2,1093,3,[ST10000NM0086, ZA21HD3Y], [ST10000NM0086, ZA21E1Z8], [ST10000NM0086, ZA20ZBAR],1032,3,[ST8000DM002,
ZA136191], [ST8000DM002, ZA12QCLA], [ST8000DM002, ZA12YMZ6],1041,2,[ST8000NM0055, ZA16V744], [ST8000NM0055, ZA171VP2],1010,2,[ST4000DM000,
Z304KA0X], [ST4000DM000, Z304JW59],1044,2,[ST8000NM0055, ZA17HPSZ], [ST8000NM0055, ZA18BLRJ],1014,2,[ST4000DM000, Z304LAP7], [ST4000DM000,
Z304KCMR],1152,1,[WDC WUH721414ALE6L4, QGK8T2WT],1120,1,[HGST HUH721212ALN604, AAG63XJH],1089,1,[TOSHIBA MG07ACA14TA,
61A0A0MUF97G],1034,1,[ST8000DM002, ZA11R9DT]
1719671715754,0,0.0,1093,3,[ST10000NM0086, ZA21HD3Y], [ST10000NM0086, ZA21E1Z8], [ST10000NM0086, ZA20ZBAR],1032,3,[ST8000DM002, ZA136191],
[ST8000DM002, ZA12QCLA], [ST8000DM002, ZA12YMZ6],1041,2,[ST8000NM0055, ZA16V744], [ST8000NM0055, ZA171VP2],1010,2,[ST4000DM000, Z304KA0X],
[ST4000DM000, Z304JW59],1044,2,[ST8000NM0055, ZA17HPSZ], [ST8000NM0055, ZA18BLRJ],1014,2,[ST4000DM000, Z304LAP7], [ST4000DM000,
Z304KCMR],1152,1,[WDC WUH721414ALE6L4, QGK8T2WT],1120,1,[HGST HUH721212ALN604, AAG63XJH],1089,1,[TOSHIBA MG07ACA14TA,
61A0A0MUF97G],1034,1,[ST8000DM002, ZA11R9DT]
```

Figura 5-Risultati Query2_all_days per 2024-04-05

6.3 Risultati Query 3

window,vault_id,min,percentile25,percentile50,percentile75,max,count,latency,throughput
1719670446000,1090,333.000000,49421.953125,49447.000000,49455.140625,74326.000000,367,2000,5954.000000
1719670446000,1091,314.000000,49142.000000,49146.000000,49150.700000,61178.000000,369,2000,5954.000000
1719670446000,1092,502.000000,48858.160000,48864.000000,48868.160000,60002.000000,398,2000,5954.000000
1719670446000,1093,215.000000,48494.870000,48500.347222,48505.560000,48514.000000,409,2000,5954.000000
1719670446000,1094,8207.000000,16598.810000,17781.458333,21518.110000,44624.000000,395,2000,5954.000000
1719670446000,1095,3876.000000,16507.880000,18136.666667,22355.800000,45442.000000,395,2000,5954.000000
1719670446000,1096,8064.000000,47051.680000,47056.720000,47063.520000,69680.000000,382,2000,5954.000000
1719670446000,1097,78.000000,18312.156250,21198.200000,23043.093750,77804.000000,358,2000,5954.000000
1719670446000,1098,4021.000000,17695.391667,20631.680000,23011.250000,47505.000000,373,2000,5954.000000
1719670446000,1099,725.000000,15037.000000,17981.480000,22721.312500,69102.000000,364,2000,5954.000000
1719670446000,1100,5187.000000,18716.250000,20498.500000,23663.875000,72014.000000,352,2000,5954.000000
1719670446000,1101,356.000000,15621.980000,18201.500000,22277.100000,45754.000000,376,2000,5954.000000
1719670446000,1102,2773.000000,17953.340000,21760.472222,23585.660000,71085.000000,404,2000,5954.000000
1719670446000,1103,1283.000000,14362.600000,18042.388889,21092.080000,45518.000000,390,2000,5954.000000
1719670446000,1104,111.000000,14788.560000,17490.600000,21584.360000,45042.000000,382,2000,5954.000000
1719670446000,1105,3061.000000,17395.100000,19511.430556,22287.960000,43243.000000,399,2000,5954.000000
1719670446000,1106,6560.000000,16489.533333,18357.060000,21020.960000,76511.000000,372,2000,5954.000000
1719670446000,1107,558.000000,16268.640000,17680.416667,22266.060000,42896.000000,401,2000,5954.000000
1719670446000,1108,696.000000,16796.800000,19218.666667,22888.360000,41573.000000,384,2000,5954.000000
1719670446000,1109,95.000000,16910.180000,19382.333333,21713.500000,84375.000000,392,2000,5954.000000
1719670446000,1110,267.000000,17223.970000,18659.569444,22581.590000,69759.000000,403,2000,5954.000000
1719670446000,1111,570.000000,39380.820000,39398.750000,39406.540000,39417.000000,388,2000,5954.000000
1719670446000,1112,142.000000,17253.410000,18663.319444,22620.090000,39065.000000,403,2000,5954.000000
1719670446000,1113,80.000000,38440.000000,38448.680000,38460.208333,42718.000000,371,2000,5954.000000
1719670446000,1114,1660.000000,17534.125000,18721.460000,21263.281250,81516.000000,362,2000,5954.000000
1719670446000,1115,602.000000,14966.670000,17036.160000,21351.760000,38060.000000,379,2000,5954.000000

Figura 6-Risultati Query3 per 2 secondi

6.4 Grafici Query 1

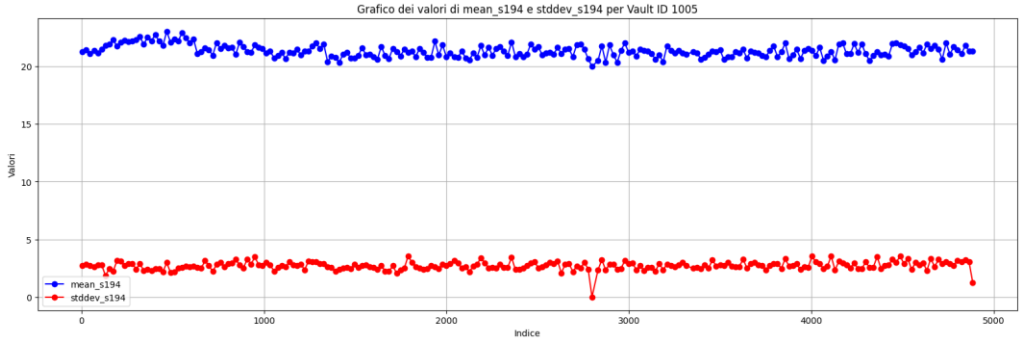

Figura 7-Media e varianza misurati per vault_id 1005

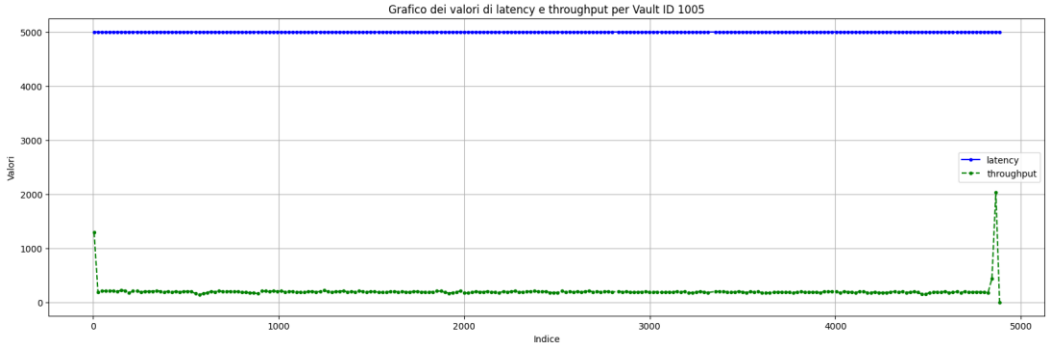

Figura 7-Latenza e Throughput misurati per vault_id 1005

6.5 Grafici Query 2

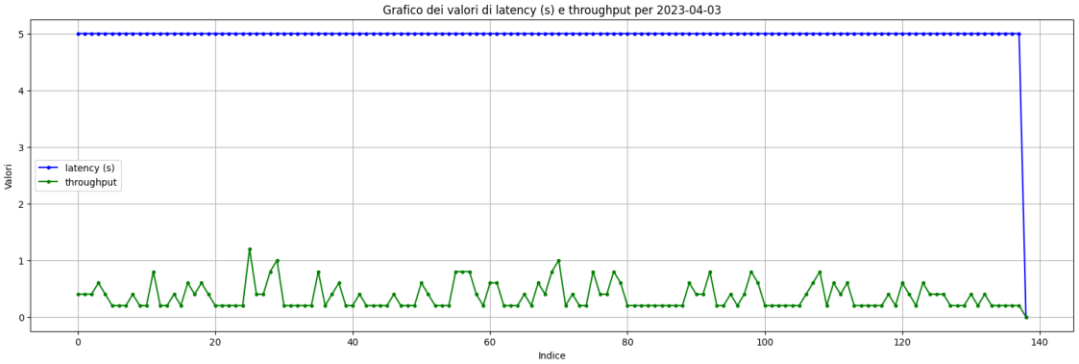

Figura 8-Latenza e Throughput misurati per 2024-04-03

6.5 Grafici Query 3

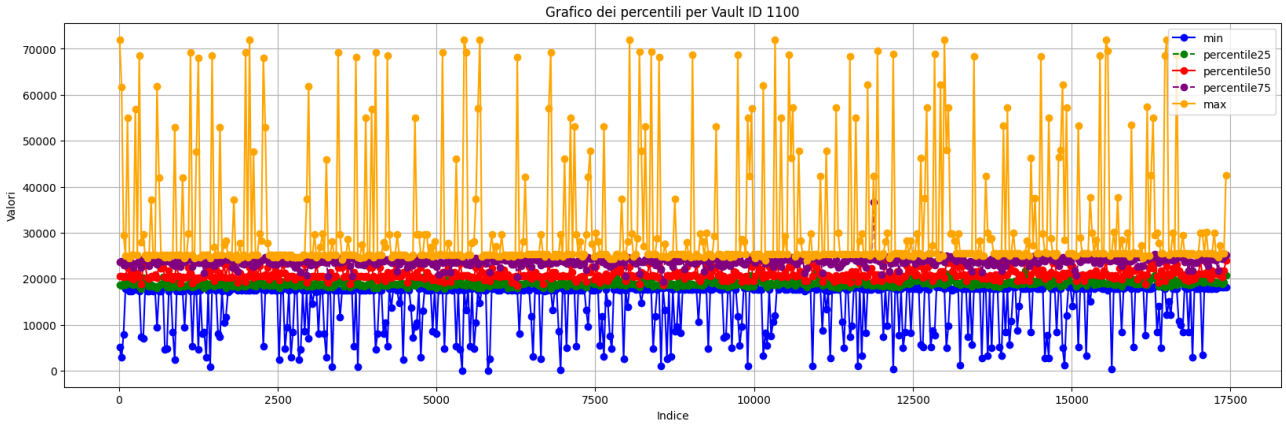

Figura 9-Percentili per vault_id 1100

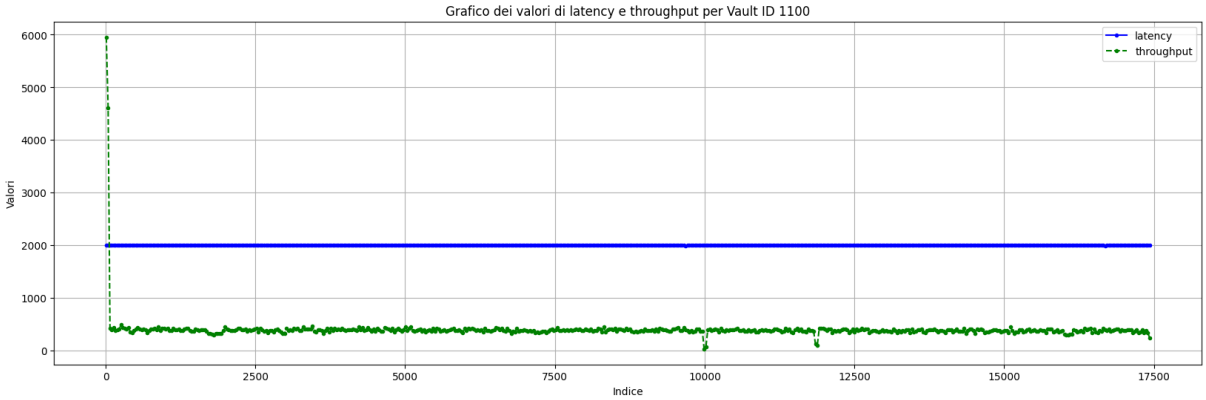

Figura 10-Latenza e Throughput per vault_id 1100