SABD-PROJECT2

Introduzione

L'obbiettivo di questo report è quello di presentare l'implementazione ed i risultati ottenuti nella realizzazione del progetto di Sistemi e Architetture per i Big Data dell'anno 2020-2021.

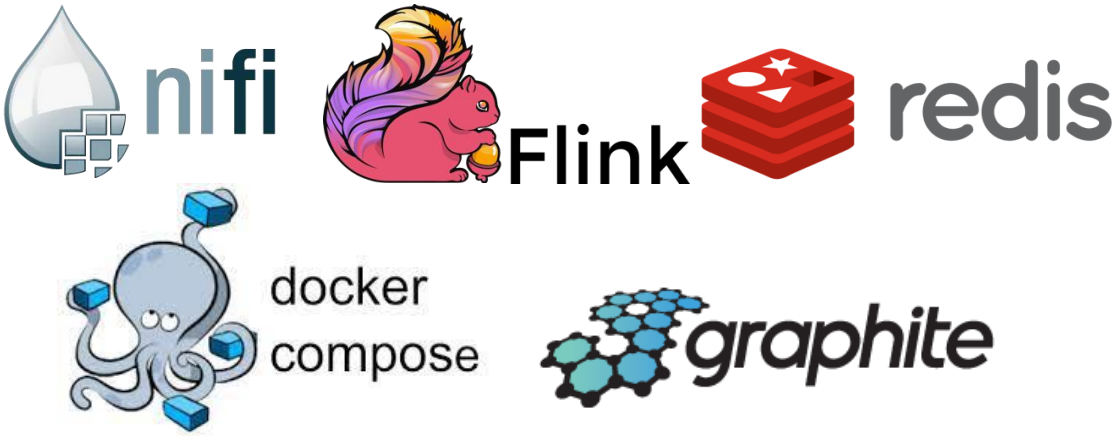
Per lo sviluppo di questo progetto è stato utilizzato il framework Apache Flink per analizzare i dati provenienti da dispositivi Automatic Identification System (AIS), utilizzati per garantire la sicurezza delle navi in mare e nei porti, in modo tale da rispondere alle seguenti domande:

- Query1: calcolare, per ogni cella del Mar Mediterraneo Occidentale, il numero medio di navi militari, navi per trasporto passeggeri, navi cargo e others transitate negli ultimi 7 giorni e nell'ultimo mese

- Query2: computare, per il Mar Mediterraneo Occidentale ed Orientale, la classifica delle tre celle più frequentate nelle fasce orarie di servizio (00:00-11:59, 12:00-23:59), considerando una finestra temporale di 7 giorni e una di un mese

- Query3: computare la classifica in tempo reale dei cinque viaggi che hanno il punteggio di percorrenza più alto, considerando una finestra temporale di un'ora e una di due ore

Architettura del sistema



Docker Compose

Docker Compose è uno strumento per la definizione e l'esecuzione di applicazioni Docker multi-containers. Si utilizza un file "docker-compose.yml" per configurare i servizi necessari all'esecuzione del progetto in modo tale che questi possano eseguire contemporaneamente e in maniera isolata. Con un solo comando si è in grado di creare e avviare tutti i servizi specificati all'interno della configurazione, avendo in più la possibilità di decidere il numero di taskmanager che si vuole andare ad utilizzare.

In questo caso particolare, i containers che vengono istanziati sono:

- Flink: jobmanager/taskmanager
- Nifi
- Graphite
- Redis
- Grafana

Apache Flink

Apache Flink è un framework e un engine di processamento distribuito per la computazione di flussi di dati illimitati e limitati. Flink è progettato per eseguire applicazioni di streaming stateful su qualsiasi scala. Le applicazioni vengono parallelizzate in forse migliaia di attività che vengono distribuite ed eseguite contemporaneamente in un cluster. Pertanto, un'applicazione può sfruttare quantità virtualmente illimitate di CPU, memoria principale, disco e IO di rete.

La scelta di utilizzare tale framework rispetto Apache Storm è stata dettata dalla possibilità di avere semantica exactly once senza la necessità dell'utilizzo di Trident, dalla buona documentazione reperibile e dalla semplicità.

Apache Nifi

Apache Nifi è un tool affidabile e potente progettato per automatizzare il flusso di dati tra i sistemi software. Per tali ragioni, in questo progetto è stato utilizzato per andare a gestire il routing dei dati fra i conteinars istanziati.

La scelta di tale sistema è stata dettata dall'interfaccia grafica semplice e intuitiva e dalla buona documentazione reperibile online.

Redis

Redis è un key-value store open source residente in memoria, adatto per la memorizzazione veloce dei dati ed è stato utilizzato per esportare tutti i risultati ottenuti dalle query.

Graphite

Graphite è uno strumento di monitoraggio utilizzato per archiviare, recuperare, condividere e visualizzare i dati riguardanti le serie temporali.

Implementazione

Per la realizzazione delle tre queries è stata eseguita una prima fase di acquisizione dei dati provenienti dal framework Nifi eseguendo operazioni di selezione e filtraggio: vengono presi in considerazione solamente i dati in cui la latitudine e la longitudine risultano comprese rispettivamente tra i valori 32, 45 e -6, 37.

Query1

Nella Query 1, per ogni cella del Mar Mediterraneo Occidentale e per ciascuna tipologia di navi, è stato calcolato il numero medio di navi transitate negli ultimi sette giorni e nell'ultimo mese.

I passi che si eseguono sono i seguenti:

- Si realizza un KeyedStream composto dai record filtrati in modo tale da considerare solamente le celle appartenenti al Mar Mediterraneo Occidentale e avente come chiave il valore della cella
- Si applica una Tumbling window
- Si realizza una funzione di aggregate, *QueryAggregateFunction,* che permette di andare a calcolare la media per ogni categoria di navi presenti nel dataset
- Si aggiunge il file a Redis

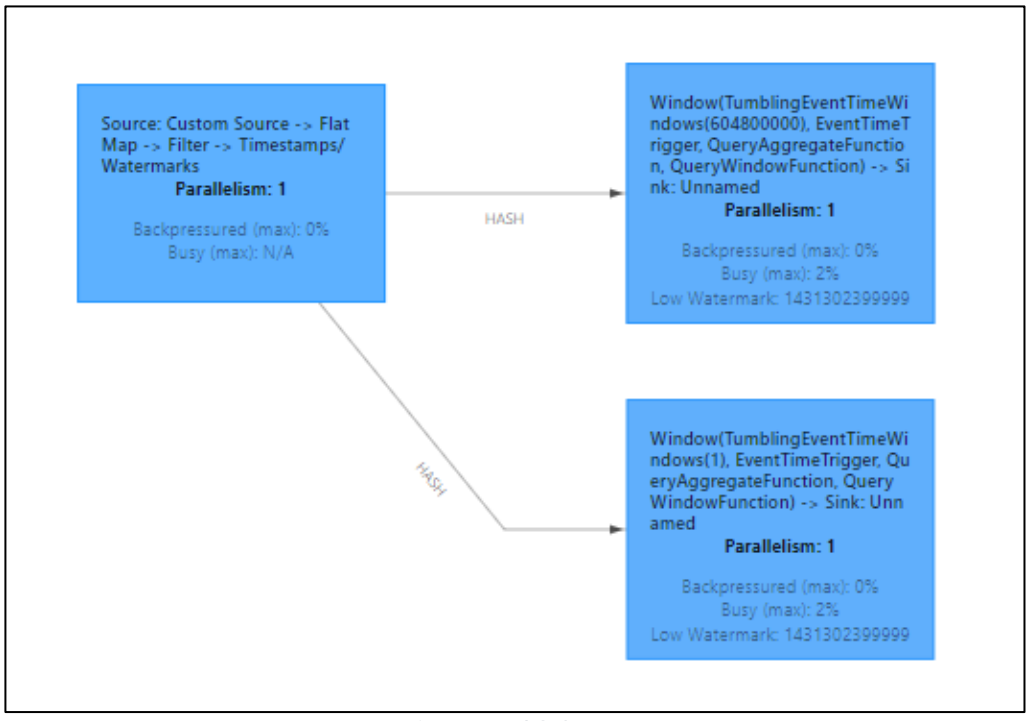


Figura 1: DAG Query1

Query2

Nella Query 2, per il Mar Mediterraneo Occidentale ed Orientale, è stata calcolata la classifica delle tre celle più frequentate nelle fasce orari di servizio (00:00-11:59, 12:00-23:59), considerando una finestra temporale di 7 giorni e una di un mese.

I passi che vengono realizzati sono i seguenti:

- Si realizza un KeyedStream avente come chiave il valore della cella
- Si applica una Tumbling window
- Si realizza una funzione di aggregate, *QueryAggregateFunction,* per calcolare la frequenza di navi per ciascuna cella e ciascuna fascia oraria
- Si esegue una keyBy per raggruppare i risultati in base al tipo di mare
- Si applica nuovamente una Tumbling window
- Si esegue una funzione di process per andare a stampare le classifiche ottenute
- Si aggiunge il file a Redis.

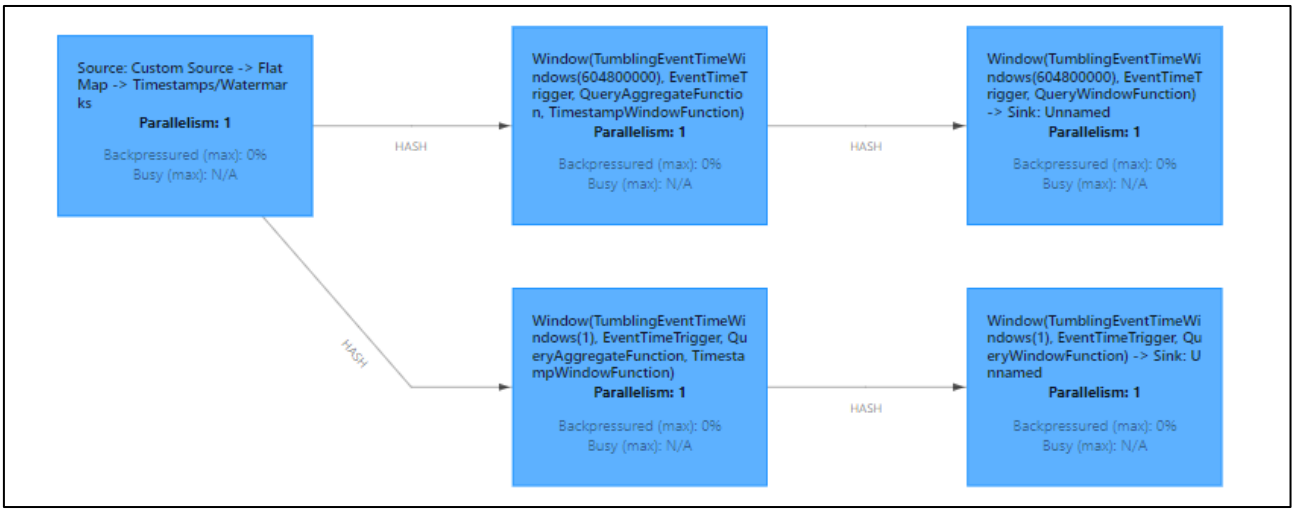


Figura 2: DAG Query2

Query3

Nella Query 3, si calcola in tempo reale la classifica dei cinque viaggi che hanno il punteggio di percorrenza più alto.

I passi che sono stati realizzati sono i seguenti:

- Si realizza un KeyedStream avente come chiave l'id del viaggio
- Si applica una Tumbling window
- Si realizza una funzione di aggregate, *QueryAggregateFunction,* per calcolare la classifica
- Si applica nuovamente una Tumbling window
- Si esegue una funzione di process per andare a stampare la classifica ottenuta
- Si salvano i risultati ottenuti su Redis.

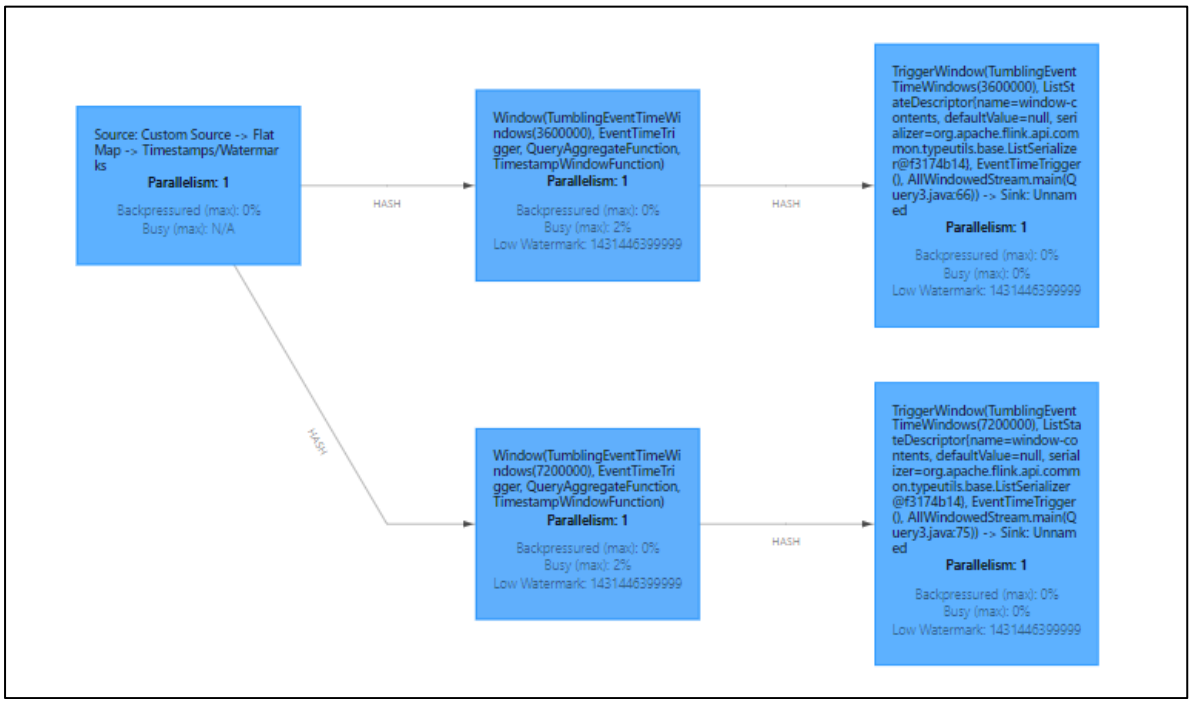


Figura 3: DAG Query3

MonthAssigner

MonthAssigner è una classe che è stata realizzata per riuscire ad effettuare la finestra di un mese necessaria per la computazione di Query 1 e Query 2. Ciò che viene calcolato è una finestra della lunghezza del mese in millisecondi, considerando l'inizio del mese stesso e la sua fine. Per fare ciò viene sfruttata la libreria di Calendar messa a disposizione da Java.

MetricsMapper

È stata utilizzata una classe *MetricsMapper,* ovvero una RichMapFunction che si occupa di calcolare le metriche richieste:

- Throughput: computato in secondi
- Latenza: computata in millisecondi

Per andare a valutare i valori ottenuti, viene implementato un contatore, che permette di stimare il numero di tuple in arrivo nel sistema, e l'intervallo temporale trascorso tra il momento attuale e lo start-time.

Nifi

input.xml

Questo template si occupa di prendere i dati dei dispositivi Automatic Identification System che vengono pre-processati dai diversi process group, i quali eseguono i seguenti compiti:

- GetFile: prende il file del dataset dal folder /nifi/dataset
- UpdateRecord: esegue un'operazione di formattazione del file, in quanto le date riportate all'interno del dataset hanno formati differenti. Si è quindi scelto tramite l'utilizzo di questo process group di andare a modificare la colonna introducendo un unico formato: yy-MM-dd HH:mm:ss.

- QueryRecord: ordina il file in base alla data
- SplitRecord: esegue uno split del dataset in più file, ciascuno composto da 5000 righe, in modo tale da avere uno streaming continuo di dati

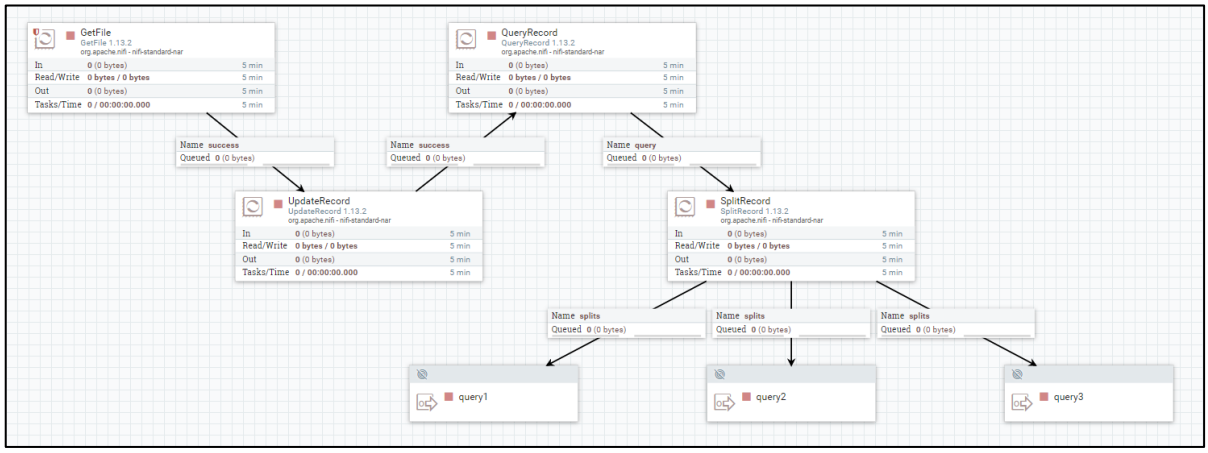


Figura 4: Template input.xml

Deployment

Come specificato precedentemente per sviluppare il progetto è stato utilizzato Docker-compose, è possibile quindi avviare l'applicazione andando ad eseguire contemporaneamente tutti i servizi, in maniera isolata e specificando il numero di taskmanager che si vuole instanziare, rendendo così il sistema più scalabile. Per fare ciò viene utilizzato il comando:

docker-compose up –scale taskmanager=2

e.g., cluster con 2 taskmanager.

Al primo deployment del cluster è, inoltre, necessario importare il template, memorizzato nel folder /nifi/templates, per poter andare ad utilizzare nifi.

Submit query

Per effettuare il submit della query è stato realizzato uno script che prende come parametro il numero della query che si intende andare ad eseguire.

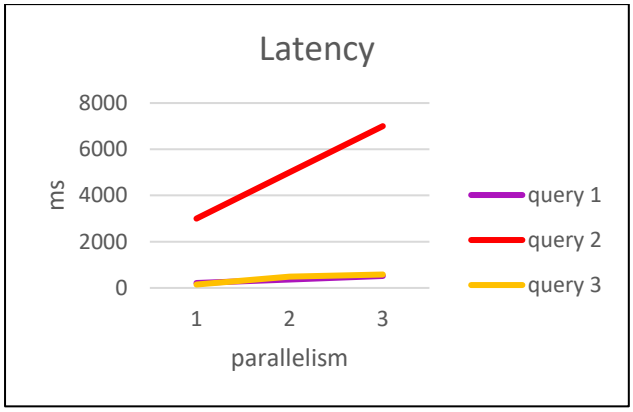
Per tanto è sufficiente andare ad eseguire i seguenti comandi:

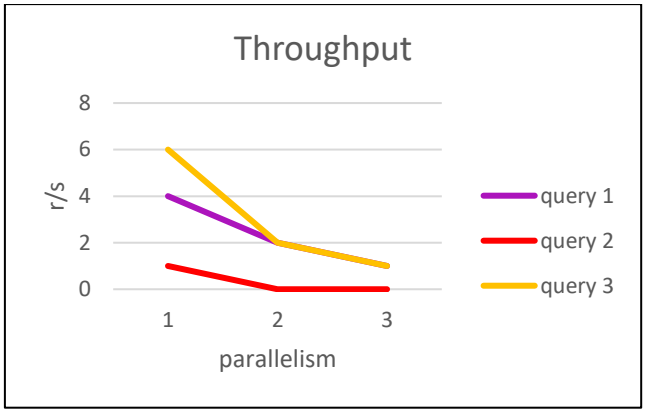
- *mvn compile assembly:single* : crea il file jar con tutte le dipendenze
- *sh submit-job.sh <numero della query>*

Visualizzazione dei risultati

Le metriche sono state misurate utilizzando un processore intel core i-7 3770. È stata presa in considerazione un'architettura composta da due taskmanager, ciascuno dei quali aveva instanziato due task. La stima delle metriche è stata, inoltre, calcolata aumentando il livello di parallelismo, facendolo variare tra 1 e 3.

Di seguito vengono riportati i risultati ottenuti:





Conclusioni

Si può quindi concludere che le performance migliori si riscontrano nella query 1 e 3 sia per quanto riguarda la latenza e per il throughput. Inoltre, all'aumentare del grado di parallelismo degli operatori vi è un deterioramento delle prestazioni.