Michele Mariotti
Martin Arsoski

Monitoraggio Traffico Stradale

Documento di progetto del corso DATA INTENSIVE APPLICATION AND BIG DATA

OBIETTIVO

L'obiettivo del progetto è:

- raccogliere i dati relativi al traffico, simulando la trasmissione da parte di sensori situati in diverse postazioni
- elaborare tali dati per calcolare statistiche relative a ciascuna postazione (transiti medi giornalieri, transiti medi per ogni giorno della settimana, transiti totali settimanali)
- memorizzare i dati in un database distribuito
- visualizzare i dati attraverso una semplice interfaccia grafica, permettendo di analizzarli più agevolmente

DATASET

I dati utilizzati provengono dal Sistema di Monitoraggio regionale dei flussi di Traffico Stradali (MTS) dell'Emilia-Romagna e sono accessibili al seguente link: https://serviziambiente.regione.emilia-romagna.it/portaleviabilita/flussi.

In particolare, i dati selezionati rappresentano il numero di transiti giornalieri per ciascuna postazione, nel periodo compreso tra il 1° gennaio 2025 e il 31 marzo 2025.

È stata fatta una pulizia dei dati prima del loro utilizzo attraverso uno script python.

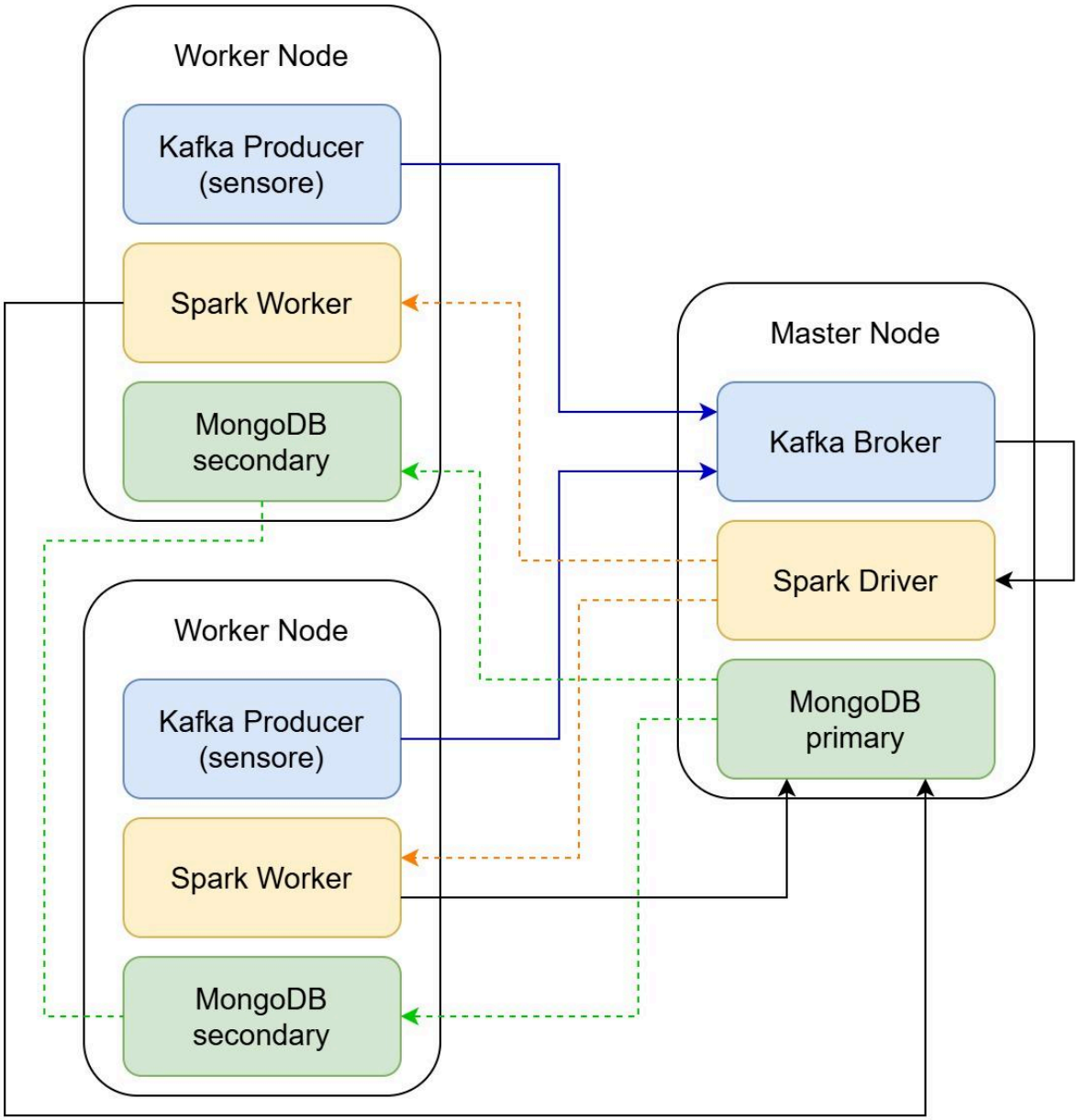
L'invio dei dati viene simulato come se avvenisse attraverso operazioni batch a fine giornata da parte dei sensori.

SPECIFICHE E ARCHITETTURA

Per eseguire correttamente il progetto è necessario installare le seguenti tecnologie sulle macchine virtuali:

- **Java**: versione 21.0.6
- **Apache Kafka**: versione 2.8.1
- **Apache Spark**: versione 3.5.4
- **MongoDB**: versione 8.0.4

Inoltre sono stati utilizzati anche Python e Maven durante lo sviluppo del sistema.



Per semplicità nella nostra simulazione i sensori sono implementati nei nodi worker, in modo da limitare il numero complessivo di VM.

CONFIGURAZIONE

Connessione delle Macchine Virtuali

Sono state utilizzate tre macchine virtuali (VM), connesse tra loro tramite rete interna per semplificare la comunicazione tra i nodi del sistema.

È necessario modificare il file `/etc/hosts` su ciascuna VM aggiungendo le seguenti righe:

```
192.168.56.101   master
192.168.56.102   worker1
192.168.56.103   worker2
```

Configurazione di Kafka

Modificare il file `~/kafka/config/server.properties` sul nodo master:

```
broker.id=0
listeners=PLAINTEXT://192.168.56.101:9092
advertised.listeners=PLAINTEXT://192.168.56.101:9092
zookeeper.connect=localhost:2181
```

Eseguire il seguente comando sul nodo master per creare un topic chiamato `traffic`:

```
kafka-topics.sh --create --topic traffic --bootstrap-server
192.168.56.101:9092 --partitions 1 --replication-factor 1
```

Configurazione MongoDB

Modificare il file `/etc/mongod.conf` per ogni macchina:

- Master (`192.168.56.101`)

```
net:
  bindIp: 192.168.56.101
```

- Worker1 (`192.168.56.102`)

```
net:
  bindIp: 192.168.56.102
```

- Worker2 (`192.168.56.103`)

```
net:
  bindIp: 192.168.56.103
```

Su tutte e 3 le macchine aggiungere questa sezione:

```
replication:
  replSetName: "rstraffic"
```

Sulla macchina master (`192.168.56.101`), aprire mongosh e inizializzare il replica set:

```
rs.initiate({
  _id: "rstraffic",
  members: [
    { _id: 0, host: "192.168.56.101:27017" },
    { _id: 1, host: "192.168.56.102:27017" },
    { _id: 2, host: "192.168.56.103:27017" }
  ]
})
```

Configurazione di Spark

Per il corretto funzionamento del cluster Spark, è necessario modificare il file `conf/spark-env.sh` su tutte le VM.

- Nodo Master:

```
export SPARK_MASTER_HOST = 192.168.56.101
```

- Nodi Worker:

```
export SPARK_WORKER_CORES=1
export SPARK_WORKER_MEMORY=2g
```

SVILUPPO

Trasmissione dati tramite Kafka Producer

I dati vengono pubblicati nel topic traffic simulando operazioni batch a fine giornata da parte di 2 sensori (interpretati dai nodi worker).

Ogni nodo worker si occupa di una partizione del dataset. La partizione è stata fatta dividendo le postazioni in due sottoinsiemi in modo equo.

Nella nostra simulazione si svolge una giornata ogni 20 secondi per poterlo testare velocemente.

Elaborazione Batch con Apache Spark (Kafka Consumer)

Nel progetto è stato implementato un componente in Apache Spark con il ruolo di consumer dei dati trasmessi su Kafka, sottoscrivendosi al topic traffic.

Al fine di simulare l'elaborazione giornaliera dei dati in un contesto accelerato, è stata introdotta una variabile che rappresenta la data corrente simulata, inizialmente impostata al 01/01/25.

Il batch processing segue questa logica:

1. Verifica la presenza di un messaggio di fine giornata (formato: `END_OF_DAY:<data>`) da parte di entrambi i sensori per la giornata corrente
2. In caso affermativo, elabora tutti i dati raccolti per quella giornata e calcola le relative statistiche
3. I risultati aggregati vengono salvati all'interno del database MongoDB
4. La variabile di data corrente viene aggiornata al giorno successivo.

Questa logica si è resa necessaria per gestire correttamente la simulazione di tempo accelerato: in uno scenario reale, l'elaborazione avverrebbe naturalmente a fine giornata (es. alle 23:59), ma nella simulazione un giorno corrisponde a 20 secondi, e il sistema deve poter riconoscere e reagire dinamicamente alla conclusione della giornata simulata.

Memorizzazione dati in MongoDB

Struttura documenti transiti giornalieri

```
{
  "station_id": 12,
  "station_name": "SS 16 tra A 14 (casello Cattolica) e confine regionale
Marche",
  "date": "2025-01-01",
  "total_transits": 10584,
  "day_of_week": 3,
  "week": 1
}
```

Questi documenti vengono memorizzati nella collezione `daily_transits`, che cresce quotidianamente.

È stato scelto di integrare direttamente il nome della postazione all'interno di ciascun documento, anziché creare una collezione separata, poiché il nome non cambia mai e ciò consente query più rapide e semplici nel contesto della nostra applicazione.

Struttura documenti statistiche

```
{
  "station_id": 12,
  "daily_avg": 8375,
  "day_of_week_avg": {
    "1": 8231,
    "2": 8120,
    ...
    "7": 7800
  },
  "weekly_total": {
    "1": 54100,
    "2": 56780,
    ...
  },
  "updated_to": "2025-03-31"
}
```

Questi documenti sono salvati nella collezione `station_stats`, che non cresce nel tempo, ma viene semplicemente aggiornata quotidianamente con statistiche aggiornate per ciascuna postazione.

Le statistiche sono state aggregate per postazione, con l'obiettivo di rendere più efficiente e immediato l'accesso ai dati nella nostra applicazione.

Preferenze di lettura

readPreference: "primaryPreferred"

readConcern: "local"

Abbiamo scelto di privilegiare la disponibilità rispetto alla consistenza nella lettura dei dati. Questa decisione è motivata dal fatto che, nel contesto della nostra applicazione, l'unico potenziale problema di consistenza riguarda la possibilità che il nodo master disponga di dati più aggiornati rispetto ad un nodo secondario. Tuttavia, anche in tal caso, i dati presenti sul secondario non sarebbero errati, ma semplicemente riferiti a un momento precedente e non aggiornati all'ultimo giorno disponibile.

Inoltre, la nostra interfaccia ha uno scopo esclusivamente consultivo: viene utilizzata solo per la visualizzazione dei dati, senza possibilità di modificarli. Questo rende accettabile una minore consistenza in favore di una maggiore disponibilità.

Preferenza di scrittura

writeConcern: "majority"

Evita perdita di dati in caso di crash del PRIMARY subito dopo la scrittura.

Visualizzazione dati

È stato realizzato un server Node.js che fornisce API per leggere dati tramite operazioni GET da MongoDB.

Per quanto riguarda il front-end è stata utilizzata la tecnologia HTML + CSS + JS.

Una volta selezionata la postazione che si intende visionare, vengono mostrati 3 grafici:

- line chart con transiti giornalieri e relativa media
- bar chart con transiti medi per ogni giorno della settimana
- bar chart per transiti totali settimanali.

AVVIO E ARRESTO DEL SISTEMA

Avvio del Sistema

Per avviare correttamente l'intero sistema, seguire la seguente sequenza di comandi a partire dal nodo Master:

1. `startMaster.sh` – Avvia i servizi sul nodo Master (Spark, Kafka, Zookeeper, MongoDB).
2. `recreateTopic.sh` – Ricrea il topic `traffic`, eliminando eventuali dati residui.
3. `logMessages.sh` – Visualizza in tempo reale tutti i messaggi ricevuti dal topic `traffic`
4. `startWorkers.sh` – Avvia i servizi su entrambi i nodi Worker (Spark, Kafka Producer, MongoDB) .
5. `runSparkProcessor.sh` – Avvia il componente Spark che consuma i dati da Kafka, esegue l'elaborazione giornaliera e scrive i risultati su MongoDB.
6. `node server.js` – Avvia il backend Node.js che espone le API REST per l'accesso ai risultati.
7. `python3 -m http.server 8000` – Avvia un server locale per servire il frontend.

Arresto del Sistema

Per arrestare correttamente l'esecuzione del sistema, seguire la seguente sequenza di comandi a partire dal nodo Master:

1. `stopAllWorkers.sh` – Ferma i servizi sui nodi Worker.
2. `stopMaster.sh` – Ferma i servizi sul nodo Master.

Reset del Database

Per eliminare tutti i dati dal database `traffic` di MongoDB, eseguire il seguente comando a partire dal nodo Master:

- `deleteDB.sh`