UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

Analisi di un dataset tramite Apache Druid

Studente:
Riccardo Schiavoni (1108426)

Professore:
Domenico Potena

Anno Accademico 2022-2023

Indice

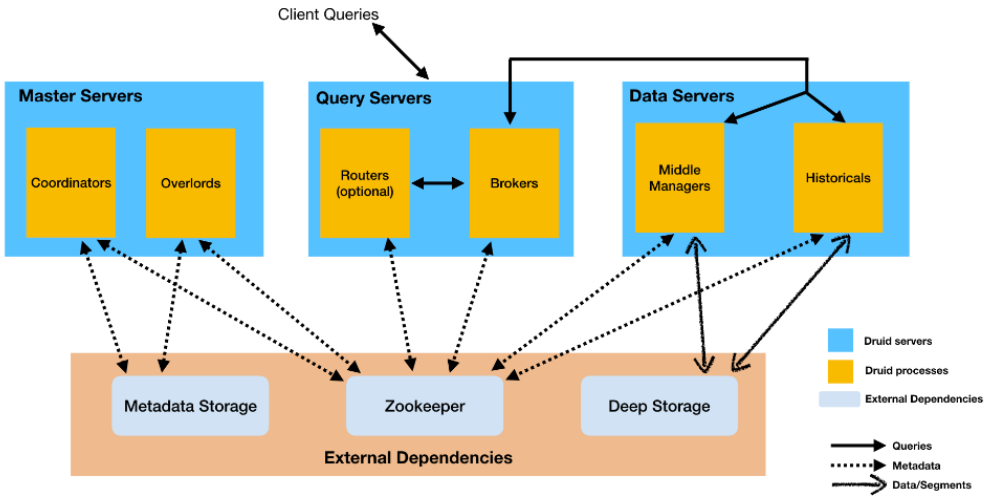
Capitolo 1

Panoramica Apache Druid

1.1 Architettura

Apache druid è un database distribuito open-source progettato per supportare l'analisi in tempo reale di grandi volumi di dati, esso permette di gestire i dati ricevuti attraverso due tipologie di fonti: batch e streaming.

Druid è composto da diversi servizi interni che interagiscono con altri esterni per realizzare i vari processi di gestione dei dati. Nella seguente figura vengono mostrati i componenti che formano l'architettura e quelle che sono le relazioni tra di essi.



Generalmente, i processi sono sviluppati su tre difersi tipi di server nodes:

- **Master Nodes**: realizzano i processi che gestiscono la disponibilità dei dati nel cluster e coordinano le operazioni di ingestione.
  - **Overlords**: bilanciano i carichi di lavoro relativi alle ingestioni dei dati sui MiddleManagers.

  - **Coordinators**: componenti che bilanciano il carico i dati tra gli Historicals, notificando loro quando i segmenti devono essere caricati, rimuovendo i segmenti obsoleti e gestendo le repiliche dei segmenti.

- **Query Nodes**: si occupano dell' insieme dei processi che ricevono le query da clienti esterni e forniscono i risultati
  - **Brokers**: ricevono le query e identificano quali sono i nodi Historicals e MiddleMangers che gestiscono i segmenti necessari, organizzano la query in subquery da inviare ai processi individuati e infine uniscono i risultati parziali per fornire la risposta al client.
  - **Router**: servizi opzionali che instradano le richieste ai Brokers, ai Coordinators e agli Overlords.
- **Data Nodes**: comprendono i processi che si occupano dell'ingestione, della memorizzazione e del recupero dei dati.
  - **MiddleManagers**: gestiscono l'ingestione dei dati e creano nuovi segmenti. Sono responsabili di come i dati vengono indicizzati, pre-aggregati, suddivisi in segmenti e pubblicati nel Deep Storage.
  - **Historicals**: sono responsabili della gestione dei dati utilizzati dalle query: recuperano i segmenti dal Deep Storage in modo per renderli disponibili piu velocemente quando vengono richiesti dalle query.

A seconda del tipo di configurazione i nodi possono essere eseguiti su uno o più server. Nel caso di ambienti di sviluppo molto grandi, infatti, ogni nodo è eseguito su un server diverso.

Per realizzare le proprie funzionalità, Druid utilizza tre dipendenze esterne:

- **Deep Storage**: si tratta dell'archivio distribuito, condiviso e accessibile da ogni server Druid utilizzato per memorizzare i dati presenti nel sistema. Esso contiene una copia aggiuntiva di ogni segmento di dati utilizzando il cloud storage, HDFS o il file system locale. Consente, inoltre, lo spostamento dei dati tra i processi Druid in background e fornisce anche una fonte di dati altamente durevole per il recupero da guasti del sistema.
- **Metadata storage**: implementato come un database relazionale, contiene i metadati del sistema, come la percentuale di utilizzo dei segmenti, le informazioni riguardanti i tasks in esecuzione e le varie informazioni di configurazione.
- **ZooKeeper**: funge da componente di coordinamento distribuito che consente ai nodi all'interno del cluster di coordinarsi e sincronizzarsi tra loro.

1.2 Ingestione

I dati vengono letti da una fonte esterna, elaborati e poi caricati nel cluster Druid. Ci sono due tipi di ingestion:

- **Batch ingestion**: i dati vengono letti da una fonte di dati statica, come file locali, HTTP o database in vari formati. Questo metodo è adatto per l'importazione di grandi quantità di dati storici e per l'indicizzazione periodica di dati da fonti esterne.
- **Streaming ingestion**: i dati vengono letti in tempo reale da una fonte di dati continua, come Kafka o Kinesis. Questo metodo è ideale per l'analisi di dati in tempo reale.

Druid memorizza i dati in datasources, ognuno dei quali è partizionato in chunks temporali e, eventualmente, ulteriormente suddiviso in base ad altri attributi. Ogni chunk contiene uno o più segment, dei file immutabili che raccolgono un gran numero di righe di dati, solitamente nell'ordine di qualche milione.

Il processo di **ingestion** (o **indexing**), gestito dal Middle Manager, consiste nel leggere i dati dalle sorgenti, batch o stream, che poi vengono suddivisi in blocchi chiamati segments, utilizzando un partizionamento basato sul tempo.

L'indexing task determina, innanzitutto, l'identificatore del segmento, poi inizia a leggere i dati e scriverli nel segmento in forma tabellare. In questa fase si dice che i segmenti sono mutabili: i dati sono subito disponibili per le interrogazioni mentre processi in background provvedono ad indicizzarli e compattarli per velocizzare future query.

Una volta finito il processo di scrittura dei dati, il segmento viene registrato nel deep storage e pubblicato scrivendo le relative informazioni nello storage dei metadati, come lo schema, la dimensione e la posizione.

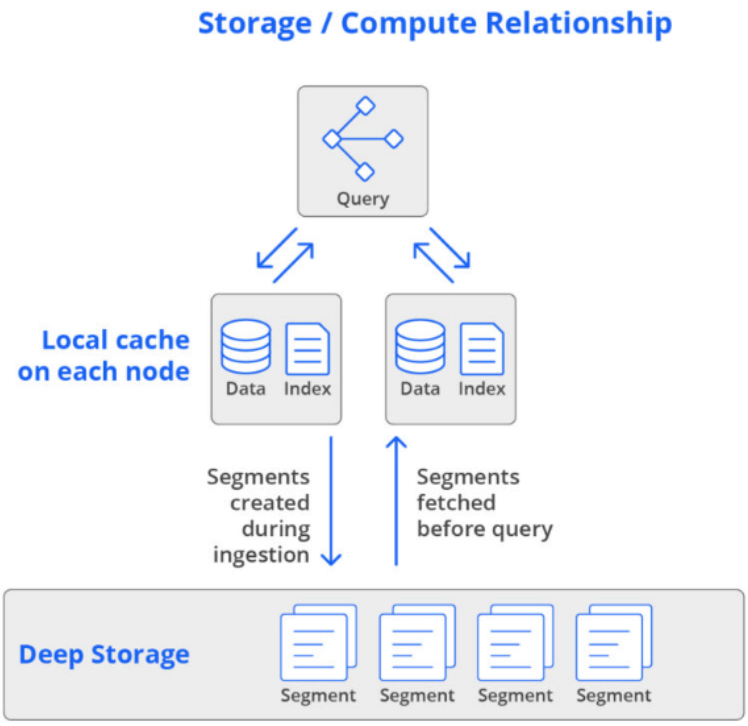
Periodicamente, il coordinator legge dallo storage dei metadati quali sono i nuovi segmenti pubblicati. Quando ne trova uno pubblicato, ma non assegnato a nessun Historical Process, ne sceglie uno per caricare quel segmento. Questo processo è chiamato Handoff.

1.3 Gestione dello storage

Druid utilizza un approccio ibrido: combina l'utilizzo di sistemi di archiviazione locali e remoti per offrire alte prestazioni e scalabilità a costi contenuti. Grazie a questo approccio architetturale ibrido, la soluzione Druid garantisce la velocità di accesso ai dati dei sistemi di archiviazione locali, combinata con i vantaggi dei costi contenuti offerti dagli storage remoti.

I dati vengono memorizzati su uno storage esterno condiviso e centralizzato, consentendo di scalare agilmente aggiungendo o rimuovendo nodi.

Quando deve essere eseguita una query, i dati necessari vengono copiati nella cache locale dei nodi interessati, evitando di dover effettuare l'accesso alla rete.



Per quanto riguarda i singoli segmenti, essi sono gestiti come file singoli che tipicamente contengono alcuni milioni di righe ciascuno.I segmenti sono il cuore dell'storage in Druid: ogni tabella può averne da uno fino a milioni.

All'interno di ogni segmento i dati sono organizzati per colonna. In questo modo le query devono caricare solo gli attributi necessari, migliorando sensibilmente le prestazioni. Inoltre ciascuna colonna è ottimizzata in base al tipo di dato.

Di norma viene creato un segmento per ogni intervallo temporale, con dimensioni ottimali (300-700MB). Segmenti dello stesso intervallo sono raggruppati automaticamente in blocchi.

Se sono necessarie modifiche a segmenti già consolidati, questi vengono sostituiti atomicamente prima di renderli disponibili alle query.

1.4 Esecuzione delle Query

Le query in Druid sono veloci, supportano alta concorrenza e possono essere eseguite su dataset di qualsiasi dimensione.

Druid supporta due linguaggi di query: SQL e nativo Druid (in JSON). Le query permettono di eseguire le principali funzioni di selezione, raggruppamento, aggregazione, scansione e filtraggio. Le query SQL Druid sono tradotte nel linguaggio nativo Druid: entrambi forniscono prestazioni molto veloci.

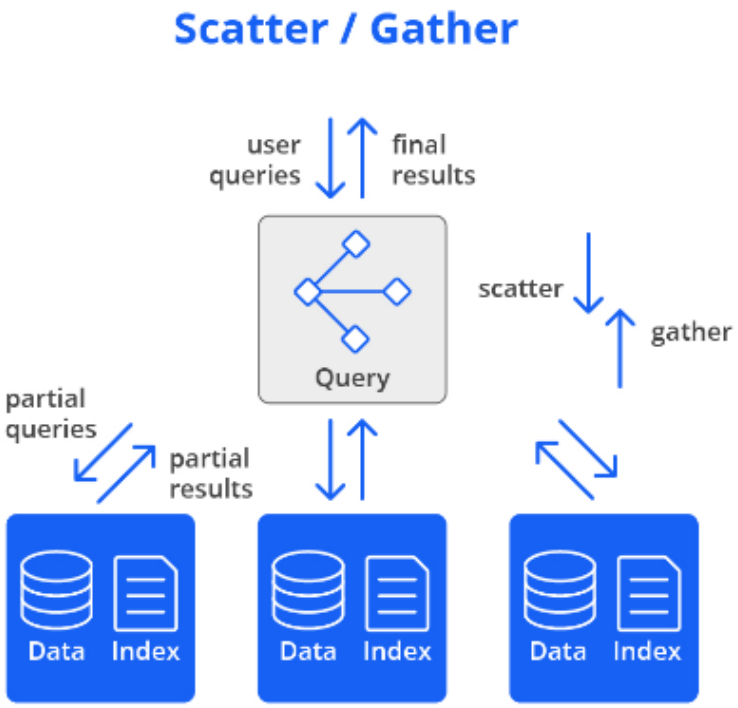
Druid, inoltre, fornisce ulteriori strumenti per ottimizzare il roll up e la compressione di dati nel caso di colonne con alta cardinalità: funzioni come HyperLogLog, Theta

sketches e Quantiles sketches forniscono delle stime probabilistiche della distribuzione dei dati all'interno degli intervalli in cui viene effettuato il roll up.

Le query sono gestite dai brokers, di solito eseguiti su un query nodes. La maggior parte delle query utilizzano il core query engine, che utilizza un approccio di tipo scatter / gather per distribuire le interrogazioni per ottenere prestazioni veloci.

In particolare questo approccio suddivide la gestione delle query in due fasi:

- **Scatter**: Il core query engine suddivide la query in sub-query, in seguito seleziona un elenco di segmenti rilevanti per ottenere i risultati e invia in parallelo le sub-query ai data nodes che contengono i relativi segmenti.
- **Gather**: Sui data nodes, le sub-query vengono elaborate e i risultati parziali vengono restituiti al query node che li integra e fornisce una risposta finale all'utente



Con questo approccio, Druid legge solo da segmenti precaricati nella memoria o nello storage locale dei nodi dati. Questo garantisce prestazioni veloci, poiché i dati sono localizzati insieme alle risorse di calcolo e non devono spostarsi su una rete.

L'alta concorrenza è stata uno degli obiettivi di progettazione originali di Druid, i cluster Druid supportano centinaia di migliaia di query al secondo.

La chiave della concorrenza in Druid è il sistema di archiviazione in segmenti, questi vengono scansionati in parallelo dalle query scatter/gather. Di solito, la scansione di

ogni segmento richiede circa 250ms, raramente più di 500ms.

In Druid quando più query tentano di scansionare lo stesso segmento, le risorse vengono rilasciate a una nuova query immediatamente al termine della scansione. Questo mantiene il tempo di calcolo su ciascun segmento molto contenuto e consente un numero molto elevato di query concorrenti.

Inoltre, Druid mette in cache automaticamente i risultati delle query, riducendo ulteriormente i tempi di esecuzione e i carichi computazionali.

1.5 Sicurezza

Druid include una serie di funzioni per garantire che i dati siano sempre protetti.

Gli amministratori di Druid possono definire utenti con nomi e password gestiti da Druid o utilizzare sistemi di autorizzazione esterni tramite LDAP o Active Directory.

Il controllo degli accessi basato sui ruoli a grana fine consente a ciascun utente di interrogare solo le fonti di dati (come le tabelle) a cui è stato concesso l'accesso, sia utilizzando le API che le query SQL.

Gli utenti appartengono a ruoli che possono utilizzare solo le risorse per le quali il ruolo è stato autorizzato.

Tutte le comunicazioni tra i processi Druid sono crittografate tramite HTTPS e TLS.

Le migliori pratiche per la distribuzione in produzione di Druid richiedono che i client utilizzino HTTPS per comunicare con i server Druid, compreso il requisito di certificati SSL validi.

Capitolo 2

Esempio di utilizzo di Apache Druid

Per l'analisi è stata utilizzata la configurazione micro quickstart di Apache Druid, si tratta di un'installazione leggera di Druid che viene eseguita su una singola macchina virtuale, ideale per per eseguire l'analisi su un set di dati limitato a scopo di sviluppo o test.

2.1 Definizione dei Dati

DC Metro Crime Data

Il dataset utilizzato per l'analisi è **DC Metro Crime Data**. Si tratta di un file formato csv contente i dati inerenti ai crimini avvenuti a Washington DC nell'arco temporale compreso tra il 2008 e il 2017.

Ognuno dei 342868 record è associato ad un identificativo univoco, si tratta di un codice intero autoincrementale.

Per quanto riguarda gli altri attributi, essi conengono informazioni temporali, spaziali e relative alla tipologia del crimine; possono essere divisi come segue:

- **Attributi Temporali**:

Nome	**Tipo**	**Descrizione**
Report Date	Data	Data di registrazione del crimine.
DATE	Data	Data di avvenimento del crimine.
Shift	Stringa	Momento della giornata in cui è avvenuto il crimine.
Year	Intero	L'anno di DATE

Month	Intero	Il mese di DATE
Day	Intero	Il giorno di DATE
Hour	Intero	L'ora di DATE
Minute	Intero	Il minuto di DATE
Second	Intero	Il secondo di DATE

- **Attributi Geografici**:

Nome	**Tipo**	**Descrizione**
QUAD	Stringa	Sono i quattro quadranti in cui è divisa Washington DC: Northwest (NW), Northeast (NE), Southwest (SW) e Southeast (SE).
EW	Stringa	Longitudine del quadrante
NS	Stringa	Latitudine del quadrante
WARD	Stringa Numerica	Gli 8 quartieri che forniscono una suddivisione utilizzata per scopi di rappresentazione politica e amministrativa.
District	Stringa Numerica	Washington DC è divisa in 7 distretti di Polizia, ognuno si occupa di una specifica area.
PSA	Stringa Numerica	Le Police Service Area (Area di servizio della polizia) sono 56 sottodivisioni dei 7 distretti di polizia. Queste aree sono utilizzate per facilitare l'allocazione delle risorse e il monitoraggio delle attività di polizia.
ANC	Stringa Alfanumerica	Advisory Neighborhood Commission (Commissione consultiva di quartiere), la città è divisa in 40 ANCs, organi elettivi incaricati di rappresentare i residenti dei quartieri in merito a questioni locali e di pianificazione.
Neighborhood Cluster	Stringa	Sono dei raggruppamenti di 4/5 quartieri vicini, in totale ne esistono 45.

Nome	Tipo	Descrizione
Census Tract	Stringa Numerica	Suddivisioni del territorio utilizzate per raccogliere e analizzare i dati demografici: Washington è divisa in oltre 170 census tract.
Block GROUP	Stringa Alfanumerica	Una divisione ulteriore rispetto a Census Tract.
Voting Precinct	Stringa Alfanumerica	Suddivision3 utilizzata per organizzare le elezioni e i voti (sono 145).
Block	Stringa	Indirizzo specifico del blocco.
XBlock e YBlock	Valore Numerico	Coordinate del blocco.

- **Attributi Riguardanti la Tipologia del crimine**:

Nome	Tipo	Descrizione
Crimetype	Stringa	I crimini sono classificati come violenti o non.
Offense Type	Stringa	Tipologia specifica di crimine.
Method	Stringa	Strumento utilizzato per compiere il crimine.

- **Altri attributi**:

Nome	Tipo	Descrizione
ID	Intero	Identificativo univoco e auto incrementale del crimine
CNN	Stringa Numerica	"Codice numerico nazionale" del crimine, viene in assegnato per identificare in modo univoco un crimine o un caso penale.
Optional	Boolean	Contiene True o False

Generazione nuovo Dataset

Affinchè l'analisi venga svolta su un numero sufficiente di dati, utilizzando le librerie **Pandas** e **Numpy** di **Python**, si è deciso di replicare **DC Metro Crime Data** fino

ad ottenerne un nuovo dataset contenente 10 milioni di record, che comprono un arco temporale compreso tra il 1990 e il 2020.

innanzitutto, sono state effettuate le seguenti operazioni preliminari:

- I record con valori nulli sono stati eliminati.
- Le colonne del precedente dataset sono state divise in 3 gruppi in base al tipo di informazione che rappresentano.
- Le colonne relative alle dimensioni spaziali sono state ulteriormente suddivise in base all'attributo QUAD, esso infatti contiene soltanto 3 diversi valori (Northwest, Northeast, Southeast), permettendo una suddivisione omogenea dei dati. Gli altri attributi geografiche non seguono un preciso ordine gerarchico: le suddivisioni territoriali hanno scopi diversi, di conseguenza dividono la città in aree spesso sovrapposte non permettendo la creazione di una gerarchia.
- Le colonne relative alle dimensioni tipologiche sono state suddivise in base all'attributo crimetype, esso ha 2 soli possibili valori (Non-Violent, Violent).
- L'attributo ReportDate e Date all'interno dello stesso record nella maggior parte dei casi contenevano date associate allo stesso giorno, nell'ottica di apache druid non avrebbe comportato alcuna modifica nel partizionamento dei dati, quindi si è deciso di tenere soltanto il campo Date.

Una volta effettuata questa divisione, ogni record del nuovo dataset viene generato come segue:

- Si sceglie un valore casuale tra Northwest, Northeast e Southeast, poi si estarae casualmente dal sottinsieme di colonne con informazioni spaziali un record con quel valore di QUAD.
- Allo stesso modo, selezionando causalmente un valore tra Non-Violent e Violent, si seleziona un record con informazioni sulla tipologia del crimine.
- Viene generato un timestamp casuale da un range di valori che viene incrementato ogni iterazione (in modo che risulti una sequenza temporale).

Elenco delle Query

Per quanto riguarda le query esse sono state scelte in modo da coinvolgere tutti gli attributi in modo omogeneo, cosi da valutare le prestazioni di druid per le diverse tipologie di operazioni:

- Conteggio del numero di crimini, in un determinato QUAD.
- Distribuzione in percentuale dei crimini, di tipologia violenti o non violenti, in una delle divisioni territoriali e di cui si è occupato un determinato DISTRICT in un determinato anno.
- Conteggio di quante diverse PSA sono intervenute in una specfica area in un determinato anno.

- Per ogni anno mostrare, in base ad una delle divisioni territoriali, l'area in cui sono avvenuti piu crimini.
- Mostrare l'andamento annuale dei crimini avvenuti in due divisioni territoriali a confronto.
- Mostrare in un determinata combinazione di ANC, NEIGHBORHOO CLUSTER, CENSUS TRACT, CRIMETYPE, in quanti diversi BLOCK sono avvenuti i crimini.
- Il diverso numero di Offense Type che ha dovuto gestire una specifica DISTRICT in una delle divisioni territoriali.
- Per ogni mese mostrare la media dei crimini avvenuti nel corso degli anni di una tipologia in una specifica area di una delle divisioni territoriali.
- Mostrare il numero di crimini avvenuti in un certo intervallo temporale, in una specifica area di una delle divisioni territoriali.

2.2 Ingestione dei Dati

Questo processo viene gestito dai **Middle Manager**, nodi specializzati nell'ingestione presenti nei cluster Druid. Essi ricevono le specifiche su cosa e come ingerire dall'**Overlord**, un nodo che coordina e supervisiona l'intero cluster.

Per inviare un task di ingestione ad un cluster Druid, bisogna fare una richiesta HTTP POST all'endpoint "/druid/indexer/v1/task" su uno dei Middle Manager.

Se il Middle Manager è in ascolto sulla porta xxxx (solitamente la porta 8091), l'URL completo per inviare il task sarebbe:

http://middleManager_ip:xxxx/druid/indexer/v1/task.

In alternativa, è possibile inviare il task all'Overlord, che si occuperà di smistarlo ad un Middle Manager. In questo caso l'endpoint è lo stesso, ma bisogna inviare la richiesta POST all'indirizzo dell'Overlord anziché del Middle Manager.

Specifiche del task di ingestione

Il corpo della richiesta POST contiene il file JSON in cui sono indicate le specifiche del task di ingestione, in questo ile vengono definite le modalità di lettura, processamento e indicizzazione dei dati.

In particolare le sezioni principali del file json sono le seguenti:

- **type**: Il tipo di task da eseguire. In questo caso, si tratta del task tramite cui vengono ingeriti i dati. L'operazione è di tipo "index_parallel", ovvero un **native batch indexing task**: i dati infatti vengono letti in blocchi da un file locale, distribuito su più nodi.

- **spec**: Specifica nel dettaglo i parametri dell'operazione di indicizzazione. Questa sezione è ulteriormente suddivisa in tre sottosezioni principali:
  - **ioConfig**: indica la sorgente dei dati:
    - **inputSource**: definisce il percorso nella directory locale dei dati da ingerire.
    - **inputFormat**: parametro che specifica il formato dei dati di input, in questo caso si tratta di un file .csv.
  - **tuningConfig**: Fornisce ulteriori dettagli sull'operazione di indicizzazione. In particolare:
    - **forceGuaranteedRollup**: quando è valorizzato come true, significa che druid utilizza il rollup perfetto: i dati vengono aggregati evitando che diversi segmenti contengano gli stessi record.
    - **partitionsSpec**: specifica il metodo di rollup, in questo caso è stato utilizzato il metodo '**hashed**'. Questo algoritmo partiziona i record in base al valore dell'hash delle dimensioni specificate.
- **dataSchema**: Definisce lo schema dei dati. Questa sezione contiene:
  - **dataSource**: Il nome del datasource in Druid dove i dati verranno indicizzati.
  - **timestampSpec**: Indica quale colonna contiene i timestamp e il formato dei timestamp.
  - **dimensionsSpec**: Elenca le dimensioni dei dati, ovvero le colonne che contengono valori categorici.
  - **granularitySpec**: Specifica la granularità dei dati, ovvero come devono essere aggregati.
  - **metricsSpec**: Definisce le metriche, ovvero le colonne che contengono valori numerici che devono essere aggregati in qualche modo.

Dimensioni e Metriche

Inizialmente lo schema dei dati è stato definito lasciando le impostazioni di default. In questa prima fase sono stati analizzate le cardinalità e le informazioni relative a ciascun attributo, di seguito sono state selezionate le colonne da rimuovere:

- L'attributo **ID** contiene un valore diverso per ogni record, gli attributi con alta cardinalità peggiorano molto le prestazioni di apache druid, si è deciso di rimuoverlo.
- L'attributo **CNN** è stato rimosso per lo stesso motivo.
- L'attributo **OPTIONAL** è stato rimosso perchè non contiene informazioni d'interesse.

- Gli attributi **XBlock** e **YBlock** sono stati rimossi sia perchè hanno alta cardinalità, sia perchè non aggiungono alcuna informazione rispetto a Block.
- L'attributo **Census Tract** è stato rimosso perchè contiene la stessa informazione di **Block GROUP**.
- L'attributo Method è stato rimosso perchè assume quasi sempre il valore "Others", di conseguenza non è un attributo d'interesse nell'ottica dello sviluppo delle query.
- In ogni record gli attributi EW e NS contengono le informazioni QUAD

Le dimensioni sono attributi che definiscono le modalità di suddivisione dei dati in segmenti. In altre parole, le dimensioni sono utilizzate per definire i gruppi di dati all'interno di un segmento. Questi attributi sono utilizzati all'intero delle query per realizzare le operazioni di aggregazione, raggruppamento e filtraggio.

Gli attributi scelti come dimensione sono i seguenti:

- Crimetype
- EW
- NS
- WARD
- DISTRICT
- ANC
- NEIGHBORHOOD_CLUSTER
- BLOCK_GROUP
- VOTING_PRECINCT
- CENSUS_TRACT

Le metriche, invece, sono gli attributi utilizzati come misure numeriche che vengono utilizzate per analizzare i dati all'interno di un segmento:

- theta_BLOCK
- theta_OFFENSE_TYPE
- theta_PSA
- count

L'attributo count indica quante occorrenze sono state raggruppate all'interno di ogni record dopo il rollup, per le altre tre metriche invece è stato utilizzato il modulo Theta Sketch, si tratta di un aggregatore che utilizza gli algoritmi di sketch Theta della libreria Apache DataSketches per stimare il numero di elementi unici in un insieme di dati.

Più in generale questo operatore offre i seguenti vantaggi:

- Ha una bassa complessità computazionale e con un basso consumo di memoria, il che lo rende adatto per l'elaborazione di grandi quantità di dati.
- fornisce una stima approssimata del numero di elementi unici in un insieme di dati, garantendo una bassa latenza delle query in cui è utilizzato.
- supporta operazioni di unione, intersezione e differenza su sketch Theta, che lo rendono flessibile e adatto a una vasta gamma di casi d'uso.

Partizionamento dei Dati

Come livello di granularità scelto per il partizionamento temporale dei dati è stato scelto il valore 'year', questo perchè un livello di granularità più fine avrebbe portato ad un alto numero di segments di piccola dimensione, aumentando il tempo di esecuzione delle query.
Per quanto riguarda il livello di aggregazione per il rollup, è stato optato il valore 'month', questa scelta permette di ridurre sensibilmente il numero di record mantenendo un quantitativo sufficiente di informazione.

La seguente immagine mostra il numero di righe ottenute utilizzando il rollup, è stata effettuata una riduzione di oltre il 92% dei record mantenendo un certo quantitativo di informazione.

123 Number of rows SUM("count")	123 Reduced number of rows COUNT(*)
10,000,000	780,919

2.3 Esecuzione delle Query

Per l'esecuzione delle query è stata utilizzata la libreria di python **pydruid**, esse fonrnisce un API per inviare le query all'endpoint /druid/v2/sql/, si tratta dell'URL del servizio che accetta query SQL come richieste HTTP POST e restituisce i risultati come risposta HTTP.
L'utilizzo di python permette di realizzare query parametriche, in questo modo i valori su cui vado ad eseguire le operazioni aggrgazione, raggruppamento, filtraggio e join sono variabili.
Questa scelta consente di valutare quali sono le prestazioni di druid nel caso in cui

riceva richieste diverse.

Per ogni sessione di esecuzione delle richieste è stata creata una lista di query secondo la seguente logica:

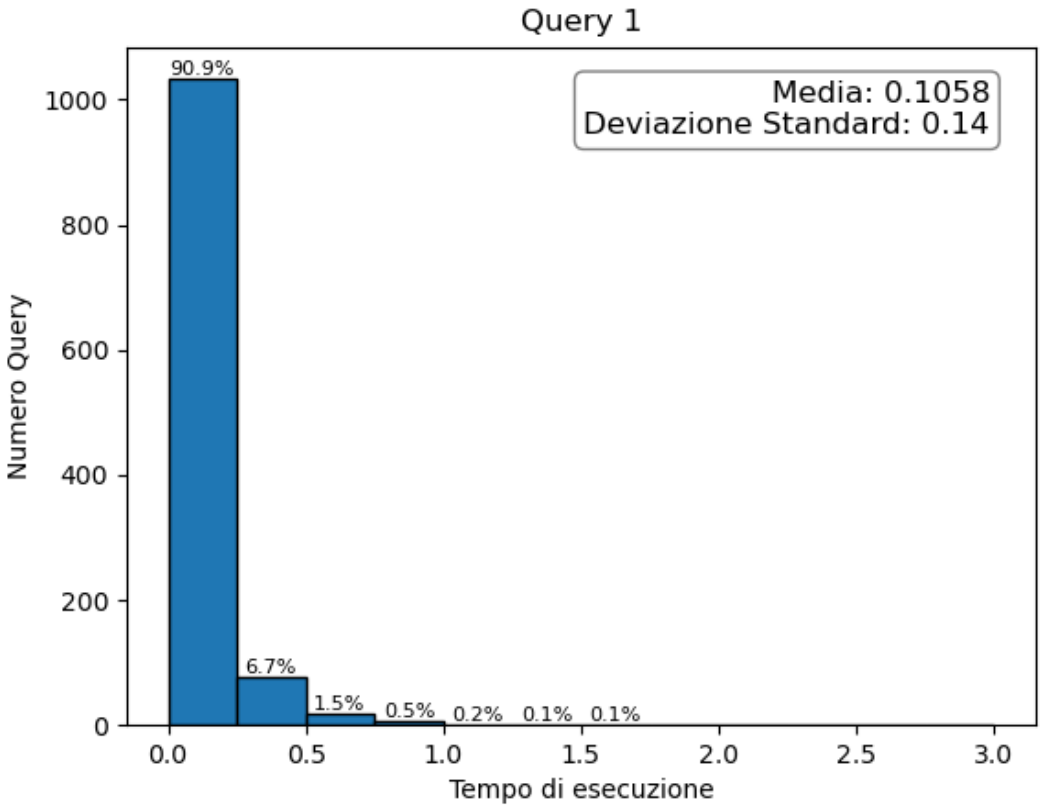
1. Viene creata una lista vuota.
2. Le 9 query parametriche vengono valorizzate con parametri casuali e aggiunte alla lista
3. I primi due passaggi sono ripetuti N volte fino ad ottenere una lista della lunghezza desiderata.

Per valutare le prestazioni di Druid in diverse condizioni di carico di lavoro, le query all'interno della lista sono state inviate a Druid utilizzando la funzione **ThreadPoolExecutor()** del modulo **concurrent.futures**. Questa funzione permette di creare un pool di thread per processare i task in modo concorrente. Tra l'invio di un insieme di richieste e il successivo, è stato inserito un determinato periodo di attesa per evitare un sovraccarico del sistema. Il numero di task concorrenti e il tempo di attesa, inoltre, variano in ogni ciclo.

Di seguito, per ogni query è riportata la distribuzione dei tempi di esecuzione, tramite l'utilizzo di istogrammi, la media e la deviazione standard, quest'ultima permette di valutare se druid mantiene una bassa latenza in base al tipo di query e al numero di richieste in un certo arco temporale.

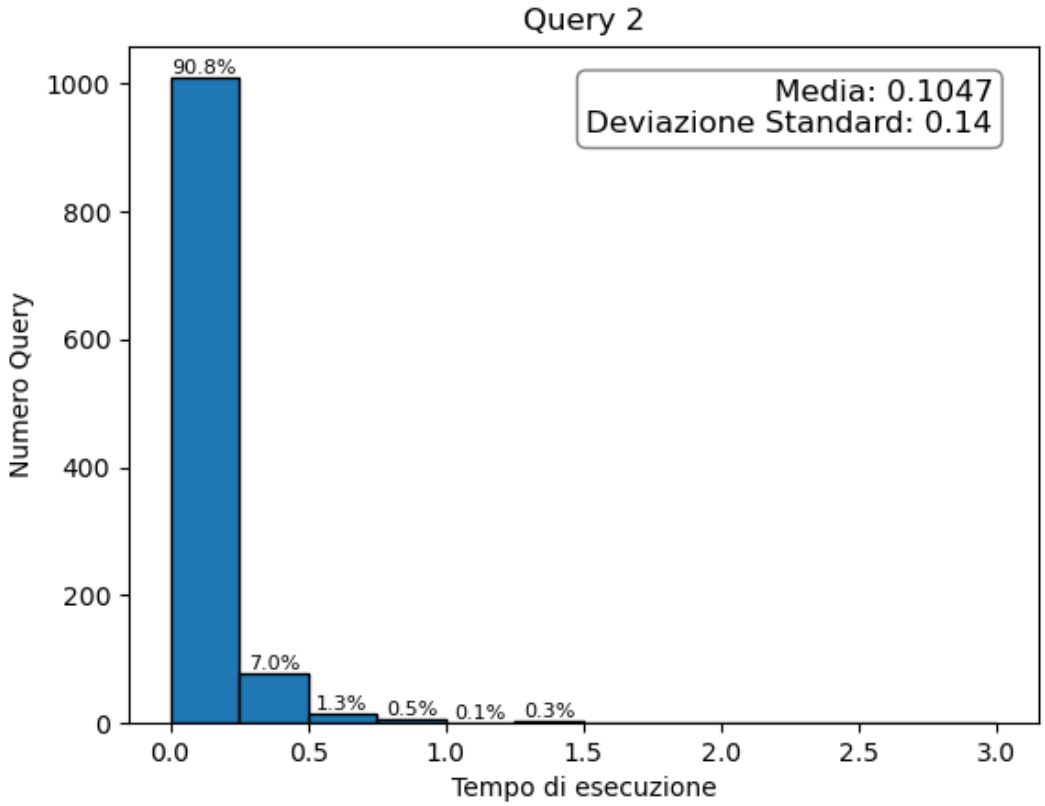
Query 1

```
SELECT SUM(count) AS Count
FROM crimesOtt
WHERE NS = ns AND EW = ew
GROUP BY *
```



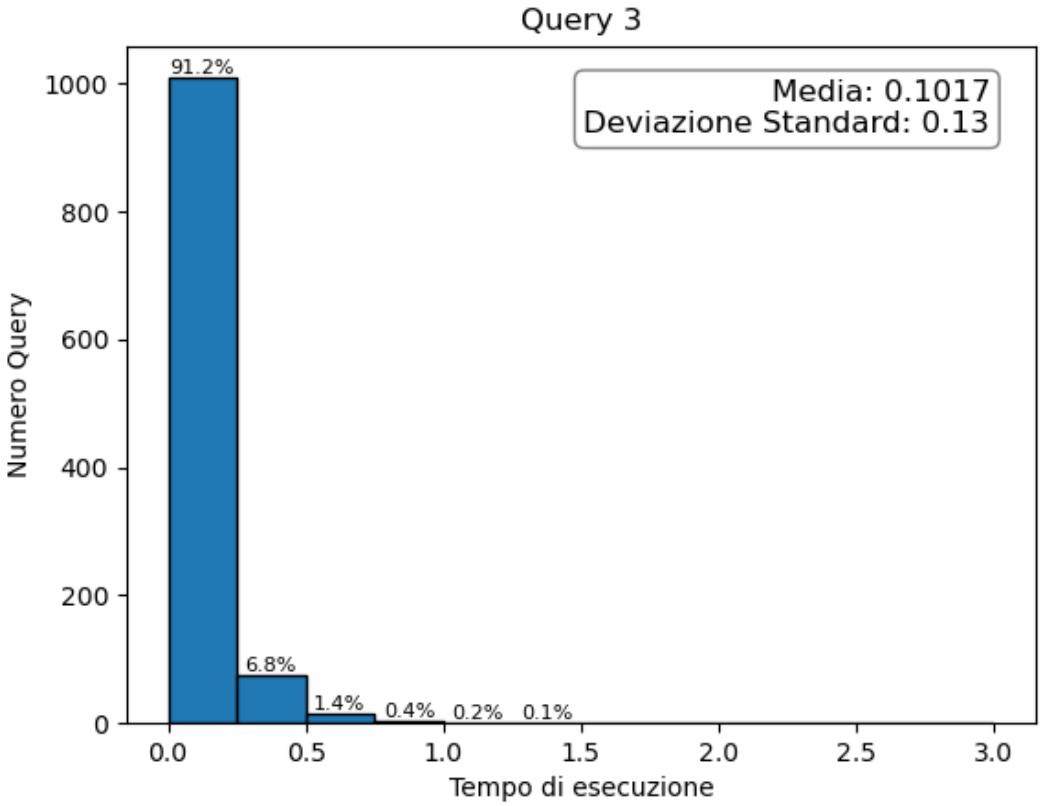
Query 2

SELECT
- dim_1,
- SUM(count) AS total_crimes,
- CONCAT(ROUND(SUM(CAST(count AS NUMERIC(10,2)))*100/ (SELECT SUM(CAST(count AS NUMERIC(10,2))) FROM crimesOtt), 4), %) AS rate
FROM crimesOtt
WHERE
- dim_1 = value AND
- CRIMETYPE = type AND
- DISTRICT = district AND
- EXTRACT(YEAR FROM FLOOR(__time TO YEAR)) = year
GROUP BY 1



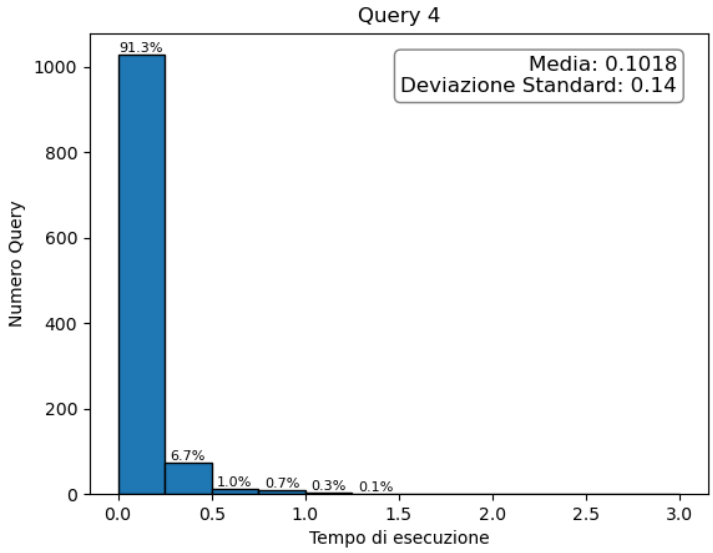
Query 3

SELECT
- APPROX_COUNT_DISTINCT_DS_THETA(theta_PSA) AS approx_count_psa
FROM crimesOtt
WHERE
- dim = value AND
- EXTRACT(YEAR FROM FLOOR(__time TO YEAR)) = year



Query 4

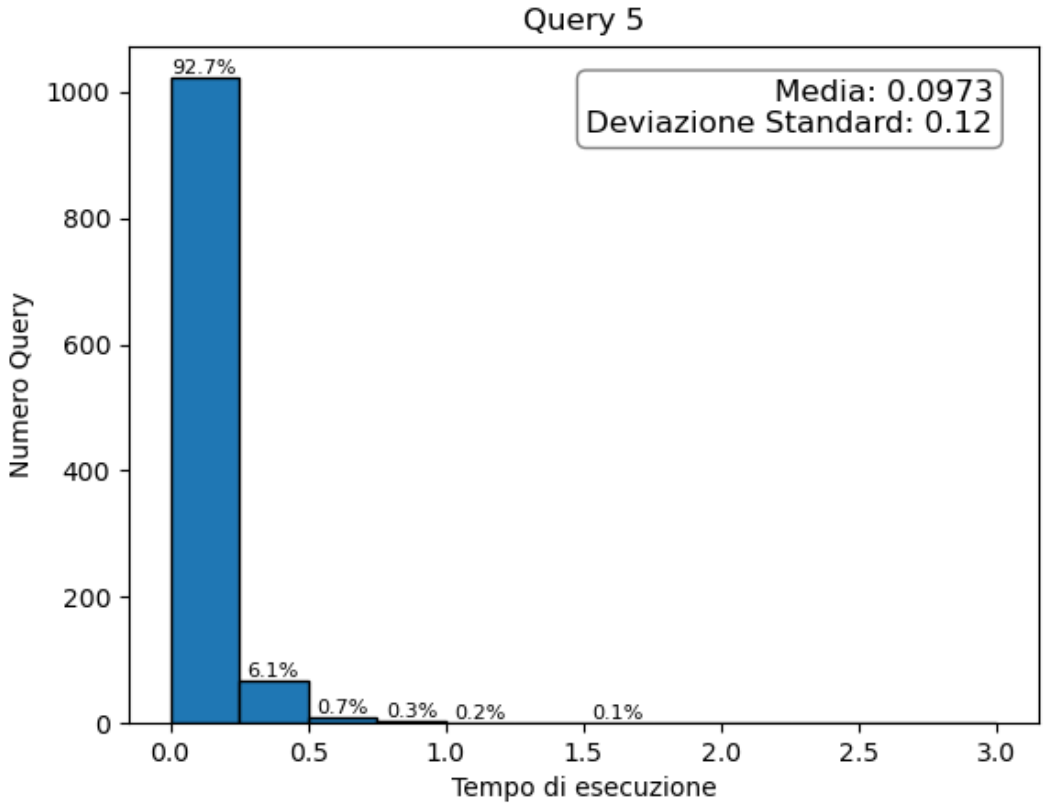
SELECT
- year1,
- dim_1,
- max_count
FROM(
– SELECT
— year1,
— MAX(count1) AS max_count
– FROM(
— SELECT
—— EXTRACT(YEAR FROM FLOOR(__time TO YEAR)) AS year1,
—— dim_1,
—— SUM(count) AS count1
— FROM crimesOtt
— GROUP BY 1, 2
–)
– GROUP BY 1
)
LEFT JOIN(
- SELECT
— EXTRACT(YEAR FROM FLOOR(__time TO YEAR)) AS year2,
— dim_1,
— SUM(count) AS count2
- FROM crimesOtt
- GROUP BY 1, 2
)
ON year1 = year2 AND max_count = count2



Query 5

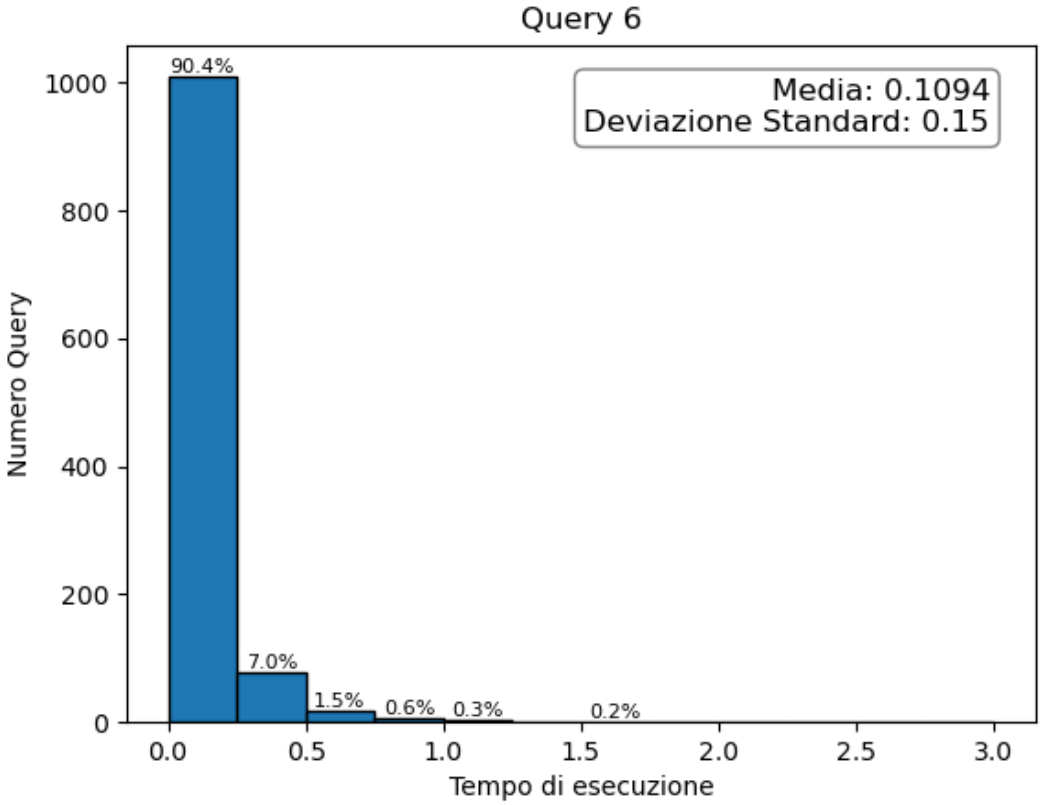
```
WITH
crimes_dim1 AS (
- SELECT
- EXTRACT(
— YEAR FROM FLOOR(__time TO YEAR)) AS year,
— dim1,
— SUM(count) AS count
– FROM crimesOtt
– GROUP BY 1, 2
),
crimes_dim2 AS (
- SELECT
- EXTRACT(
— YEAR FROM FLOOR(__time TO YEAR)) AS year,
— dim2,
— SUM(count) AS count
– FROM crimesOtt
— GROUP BY 1, 2
),


SELECT
- t1.year,
- t1.dim1,
- t1.max_dim1_count,
- t2.dim2,
- t2.max_dim2_count
FROM(
– SELECT
— a.year,
— a.dim1,
— a1.max_dim1_count
– FROM crimes_dim1 a
– INNER JOIN (
— SELECT
—- year,
—- MAX(count) AS max_dim1_count
— FROM crimes_dim1
— GROUP BY 1
– ) a1
– )ON a.year = a1.year AND a.count = a1.max_dim1_count
) AS t1
LEFT JOIN(
– SELECT
— b.year,
— b.dim2,
— b1.max_dim2_count
```

– FROM crimes_dim2 b
– INNER JOIN (
— SELECT
—- year,
—- MAX(count) AS max_dim2_count
— FROM crimes_dim2
— GROUP BY 1
–) b1
– ON b.year = b1.year AND b.count = b1.max_dim2_count
) AS t2
ON t1.year = t2.year



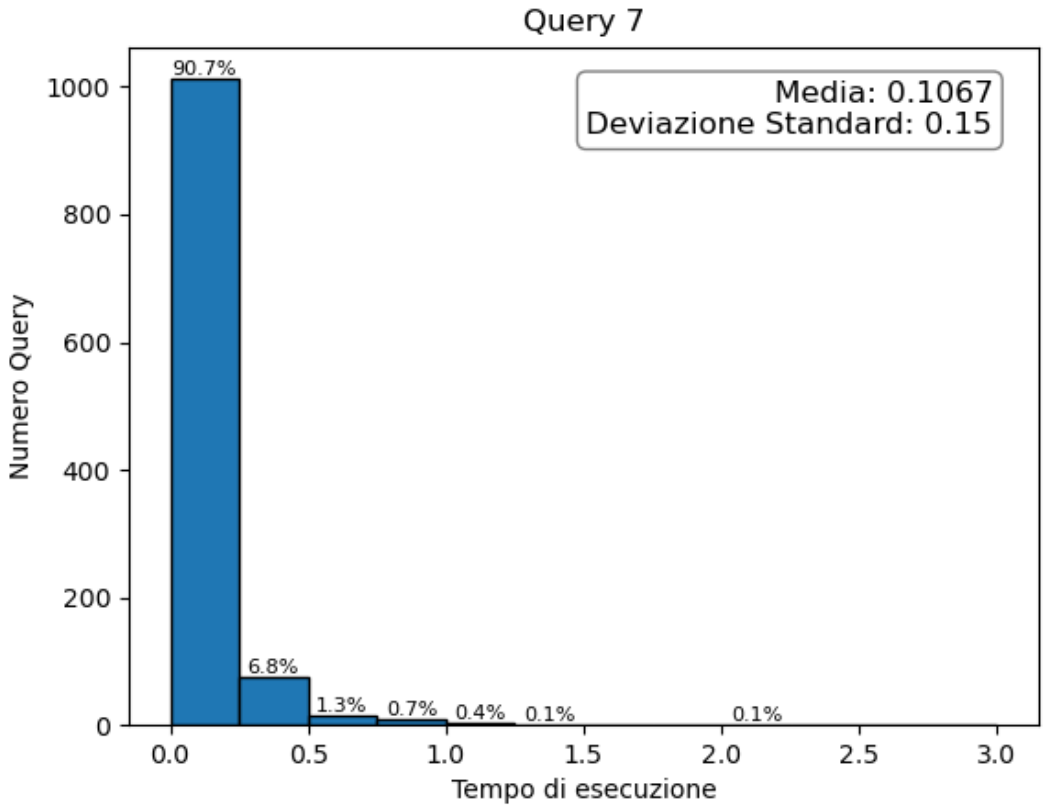
Query 6

SELECT
- APPROX_COUNT_DISTINCT_DS_THETA(theta_BLOCK) AS approx_count_blocks
FROM crimesOtt
WHERE ANC = value_A AND
NEIGHBORHOOD_CLUSTER = value_NC AND
VOTING_PRECINCT = value_VP AND
BLOCK_GROUP = value_BG AND
CENSUS_TRACT = value_CT AND
CRIMETYPE = value_C AND
EXTRACT(YEAR FROM FLOOR(__time TO YEAR)) = year



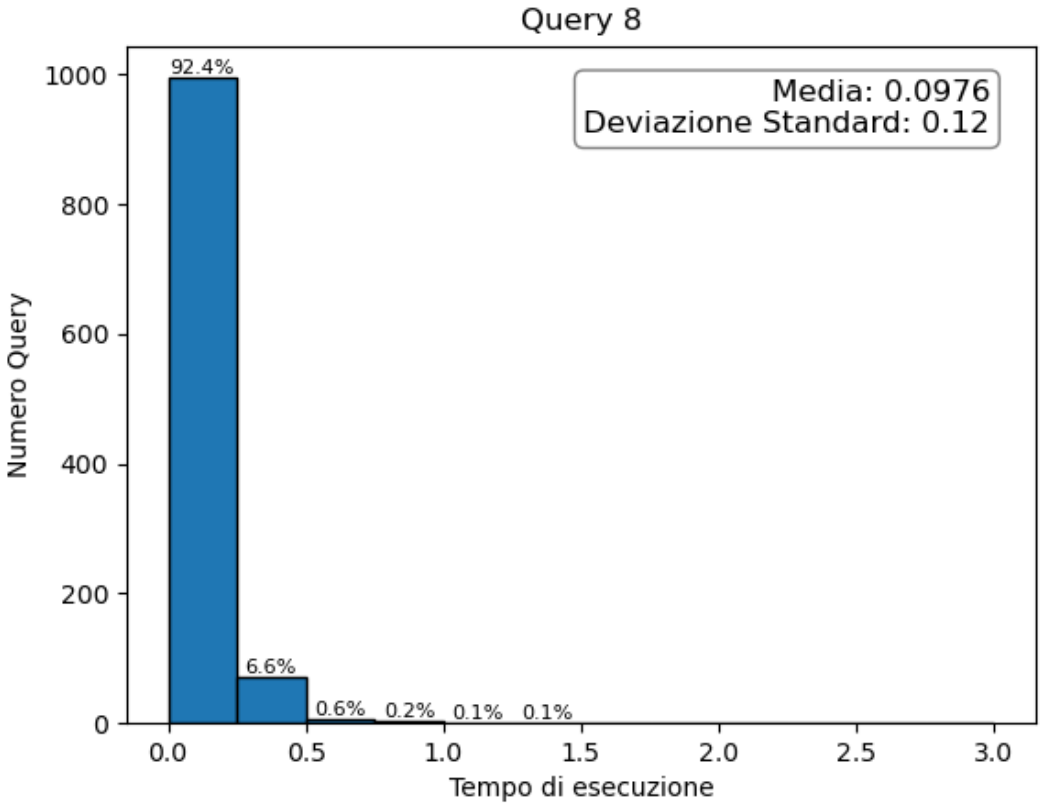
Query 7

SELECT
- APPROX_COUNT_DISTINCT_DS_THETA(theta_OFFENSE_TYPE) AS different_offense_type
FROM crimesOtt
WHERE DISTRICT=value_dist AND dim = value_dim



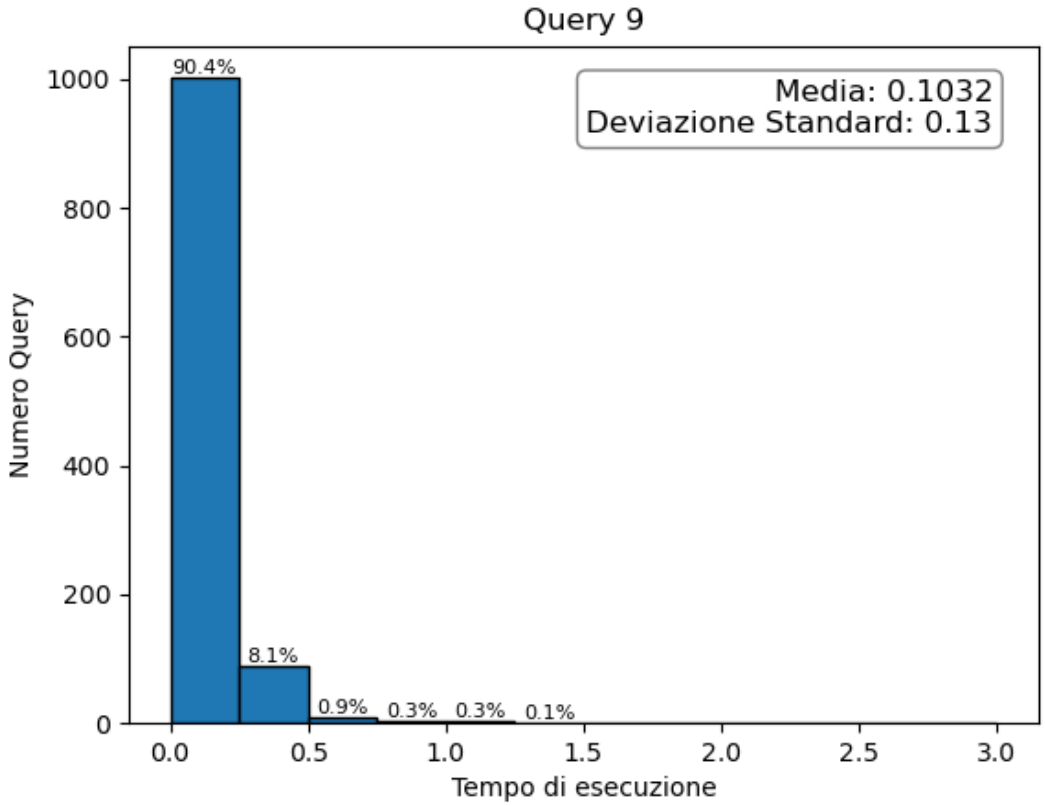
Query 8

SELECT
- EXTRACT(MONTH FROM FLOOR(__time TO MONTH)) AS month,
- ROUND(SUM(CAST(count AS NUMERIC(10,2)))*100/
- (SELECT COUNT(DISTINCT(EXTRACT(YEAR FROM FLOOR(__time TO YEAR))))
FROM crimesOtt)) FROM crimesOtt
WHERE dim = value AND CRIMETYPE = type
GROUP BY 1



Query 9

SELECT COUNT(*)
FROM crimesOtt
WHERE
– __time > = date_init AND
– __time < date_endAND
– dim = value



Capitolo 3

Conclusioni

L'analisi delle distribuzioni dei tempi di esecuzione delle diverse query permette di effettuare le seguenti considerazione:

- Per ogni query il tempo di latenza medio è intorno ai 100 ms, la caratteristica di druid è quella di riuscire a fornire i risultati richiesti molto velocemente.
- Anche le deviazioni standard sono molto contenute, ciò significa che il sistema è in grado di scalare a fronte di un diverso numero di richieste in parallelo.
- Le distribuzioni sono molto simili tra di loro, il che garantisce che Apache Druid sia in grado di gestire in modo ottimale insiemi di query eterogenei dal punto di vista degli operatori utilizzati e dei dati richiesti.

In questo caso, Apache Druid è stato utilizzato per analizzare un dataset locale di grandi dimensioni. Il software si dimostra particolarmente adatto per fornire risultati sull'intera popolazione o su particolari sottogruppi di dati, consentendo di ottenere rapidamente informazioni sulla distribuzione e sull'andamento temporale di specifiche caratteristiche dei dati.
Grazie alla funzionalità di rollup, che permette di aggregare i dati, e alla possibilita di partizionare in base al tempo, è possibile effettuare le analisi ottimizzando sia lo spazio di archiviazione, riducendo il numero dei dati all'interno dei segmenti, che il tempo di risposta delle query, pre aggregando i dati.
Tuttavia, le operazioni di aggregazione comportano la perdita delle informazioni relative ai singoli record, pertanto Druid non è adatto per l'analisi puntuale dei dati.

Druid, di conseguenza, è un software utile nelle seguenti situazioni:

- Contesti in cui si hanno grandi moli di dati da ingerire in breve tempo. Druid è particolarmente utile quando deve interfacciarsi con applicazioni di streaming, situazione in cui è necessario elaborare milioni o miliardi di eventi in eventi al giorno.
- Necessità di performance elevate nell'elaborazione dei dati.

- Analisi di serie temporali mediante query di raggruppamento dei dati per una o più dimensioni o conteggi di attributi a cardinalità elevata