Problemi dei bigdata

Programmazione

La crescita esponenziale della quantità di informazioni prodotte da sensori, dispositivi IoT, social network e sistemi enterprise ha portato alla nascita del paradigma dei Big Data, un insieme di tecniche e tecnologie volte alla gestione, archiviazione ed elaborazione di dati con volumi, velocità e varietà tali da superare le capacità dei sistemi tradizionali.

Un ecosistema Big Data moderno si basa su architetture distribuite (come Hadoop, Spark o Flink), su file system scalabili (HDFS, S3, ADLS) e su modelli di elaborazione parallela orientati alla resilienza e alla scalabilità orizzontale.

Dal punto di vista ingegneristico, la complessità di questi ambienti non risiede solo nella quantità di dati ma anche nella loro eterogeneità e variabilità temporale: schemi che cambiano nel tempo, flussi continui di eventi e la necessità di garantire qualità, sicurezza e tracciabilità generano sfide architetturali e operative.

Per questo motivo, la progettazione di un sistema Big Data richiede una visione olistica che combini data engineering, data governance, monitoraggio e ottimizzazione delle performance, adottando pratiche che riducano la latenza, migliorino l'affidabilità e minimizzino i costi di calcolo e storage.

Problema del Volume e della Scalabilità

Il primo ostacolo dei sistemi Big Data è la gestione del volume crescente di informazioni, spesso nell'ordine di terabyte o petabyte.

I tradizionali sistemi relazionali non sono progettati per gestire dataset distribuiti su più nodi, con costi elevati in termini di I/O e storage.

La sfida consiste nell'ottimizzare letture e scritture parallele, riducendo la latenza e il numero di accessi a disco.

Soluzioni: utilizzo di formati colonnari (Parquet, ORC), partitioning coerente con i filtri di query e strategie di compaction periodica dei file.

Esempio (PySpark):

```
1. df = spark.read.parquet("s3://bucket/events/")
2. filtered = df.filter("year = 2025 AND country = 'IT'")
3. filtered.write.mode("overwrite").parquet("s3://bucket/output/it2025/")
```

Problema della Velocità e del Flusso dei Dati

Molti sistemi devono elaborare stream di dati in tempo reale provenienti da sensori, log applicativi o piattaforme di e-commerce.

Qui il rischio è non riuscire a processare gli eventi con sufficiente rapidità, o perdere dati in arrivo in ritardo (latenza o lateness).

Soluzioni: uso di framework di streaming distribuito come Apache Spark Streaming o Kafka Streams, con tecniche di watermarking e window aggregation per gestire correttamente gli eventi tardivi.

Esempio (Spark Streaming):

```
1. from pyspark.sql.functions import col, window
2. events = spark.readStream.format("kafka").option("subscribe","events").load()
3. agg = (events
4.        .withWatermark("event_time","10 minutes")
5.        .groupBy(window(col("event_time"),"5 minutes"), col("event_type"))
6.        .count())
```

Problema della Qualità e Coerenza dei Dati

La veridicità e consistenza delle informazioni rappresentano una delle criticità principali. I dataset possono contenere duplicati, valori nulli, inconsistenze di formato o violazioni di vincoli referenziali.

Questi problemi compromettono l'affidabilità delle analisi e dei modelli di machine learning.

Soluzioni: introduzione di processi di data validation automatica con strumenti come Great Expectations o test di qualità integrati nelle pipeline di ETL.

Esempio (Python/Great Expectations):

```
1. import great_expectations as ge
2. df = ge.from_pandas(spark.table("sales").toPandas())
3. df.expect_column_values_to_not_be_null("order_id")
4. df.expect_column_values_to_be_between("price", 0, 10000)
5. result = df.validate()
```

Problema della Sicurezza e Governance dei Dati

Con l'aumento delle informazioni sensibili e delle normative (es. GDPR), la protezione dei dati è diventata un aspetto centrale.

I sistemi Big Data devono garantire autenticazione, autorizzazione, audit trail e mascheramento delle informazioni personali.

Inoltre, è cruciale mantenere un lineage dei dati (tracciabilità delle trasformazioni) per motivi di controllo e conformità.

Soluzioni: politiche di masking, pseudonimizzazione, row-level security e strumenti di catalogazione come Apache Atlas o OpenLineage.

Esempio (PySpark – mascheramento PII):

```
from pyspark.sql.functions import sha2, concat_ws, lit
pii = spark.table("users")
safe = pii.select(
    sha2(concat_ws(":", "email", lit("salt")), 256).alias("email_hash"),
    "country", "signup_date"
)
safe.write.mode("overwrite").saveAsTable("users_masked")
```

Tabella Riassuntiva — Principali Problemi dei Big Data

Problema	Descrizione tecnica	Rischi principali	Soluzioni principali
Volume e Scalabilità	Dati di grandi dimensioni distribuiti su più nodi, difficoltà di I/O e scansioni complete	Latenze elevate, costi di storage, inefficienza computazionale	Formati colonnari (Parquet/ORC), partizionamento logico, compaction periodica
Velocità e Streaming	Flussi continui da sorgenti real-time con latenze variabili	Perdita di dati, ritardi di elaborazione, risultati incompleti	Framework di streaming (Spark/Flink), watermark, windowing, checkpoint
Qualità e Coerenza	Dati duplicati, incompleti o incoerenti tra fonti diverse	Analisi errate, modelli ML inaffidabili, scarsa fiducia nei dati	Data validation automatica (Great Expectations), controlli ETL, deduplica
Sicurezza e Governance	Presenza di dati sensibili (PII), scarsa tracciabilità e mancanza di regole di accesso	Violazioni GDPR, perdita di reputazione, accessi non autorizzati	Masking, pseudonimizzazione, controllo accessi, lineage e audit trail

Affrontare i problemi dei Big Data non significa soltanto ottimizzare le prestazioni o ridurre i costi, ma progettare un ecosistema dati resiliente, osservabile e conforme.

Le soluzioni moderne combinano strumenti di data engineering (Spark, Kafka, Delta Lake), pratiche di data quality e governance e approcci architetturali distribuiti.

Le tecniche di partizionamento e l'uso di formati colonnari migliorano l'efficienza I/O, mentre i framework di streaming permettono di gestire eventi in tempo reale garantendo latenze ridotte e consistenza dei risultati.

Parallelamente, l'adozione di strumenti di validazione automatica e di monitoraggio continuo assicura la qualità dei dati lungo tutto il ciclo di vita, e le soluzioni di masking e lineage tutelano sicurezza e trasparenza.

In sintesi, la gestione dei Big Data non può prescindere da un approccio integrato che coniughi scalabilità, qualità, sicurezza e governance, trasformando la complessità in valore operativo e decisionale.

Soluzione al problema del Volume e della Scalabilità

Obiettivo: Ridurre i costi di I/O e migliorare le prestazioni delle query su dataset massivi.

Strategia:

- **Utilizzare formati colonnari (Parquet, ORC) che comprimono i dati e permettono di leggere solo le colonne necessarie.**
- **Implementare partizionamento logico dei dataset per anno, mese o altra chiave significativa.**
- **Eseguire compaction periodica per evitare la proliferazione di piccoli file.**

Esempio (PySpark – lettura e filtri su partizioni):

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

# Lettura di un dataset Parquet partizionato
df = spark.read.parquet("s3://data/events/")

# Filtering coerente con le partizioni per migliorare le prestazioni
filtered = df.filter("country = 'IT' AND year = 2025 AND month = 10")
filtered.write.mode("overwrite").parquet("s3://data/output/it2025/")
```

Soluzione al problema della Velocità e del Flusso dei Dati

Obiettivo: Gestire flussi di eventi in tempo reale garantendo consistenza e nessuna perdita.

Strategia:

- **Usare framework di stream processing come Apache Spark Streaming o Kafka Streams.**
- **Impiegare windowing e watermarking per gestire eventi in ritardo.**
- **Definire sink idempotenti e transazionali (Delta, Iceberg, Hudi).**

Esempio (Spark Structured Streaming – gestione eventi tardivi):

```
from pyspark.sql.functions import window, col

stream = (spark.readStream
        .format("kafka")
        .option("subscribe", "events")
        .load())

agg = (stream
      .withWatermark("event_time", "10 minutes")
      .groupBy(window(col("event_time"), "5 minutes"), col("event_type"))
      .count())

(agg.writeStream
   .format("delta")
   .option("checkpointLocation", "/chk/events")
   .outputMode("append")
   .start("/tables/agg_events"))
```

Soluzione al problema della Qualità e Coerenza dei Dati

Obiettivo: Garantire che i dati siano affidabili, coerenti e pronti all'uso analitico.

Strategia:

- **Integrare controlli di validazione nelle pipeline ETL (es. null check, range check, referential check).**
- **Usare strumenti come Great Expectations per automatizzare i test di qualità.**
- **Applicare deduplica e schema enforcement a livello di Data Lake.**

Esempio (Python – validazione automatica con Great Expectations):

```
import great_expectations as ge
df = ge.from_pandas(spark.table("sales").toPandas())

df.expect_column_values_to_not_be_null("order_id")
df.expect_column_values_to_be_between("amount", 0, 10000)
results = df.validate()
if not results["success"]:
    raise ValueError("Data quality check failed!")

```

Soluzione al problema della Sicurezza e Governance dei Dati

Obiettivo: Proteggere i dati sensibili, rispettare le normative e mantenere tracciabilità.

Strategia:

- **Applicare masking o hashing per nascondere dati personali (PII).**
- **Implementare controlli di accesso a livello di riga o colonna (Row/Column-Level Security).**
- **Introdurre sistemi di lineage e auditing per monitorare l'intero ciclo di vita dei dati.**

Esempio (PySpark – mascheramento delle informazioni sensibili):

```
from pyspark.sql.functions import sha2, concat_ws, lit

users = spark.table("customers")
safe_users = users.select(
    sha2(concat_ws(":", "email", lit("pepper")), 256).alias("email_hash"),
    "country", "signup_date"
)
safe_users.write.mode("overwrite").saveAsTable("customers_masked")
```


Buon Davante a tutti