



# 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):**

```
1.from pyspark.sql.functions import sha2, concat_ws, lit  
2.pii = spark.table("users")  
3.safe = pii.select(  
4.    sha2(concat_ws(":", "email", lit("salt")), 256).alias("email_hash"),  
5.    "country", "signup_date"  
6.)  
7.safe.write.mode("overwrite").saveAsTable("users_masked")
```



# Tabella Riassuntiva — Principali Problemi dei Big Data

Problema	Descrizione tecnica	Rischi principali	Soluzioni principali
<b>Volume e Scalabilità</b>	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
<b>Velocità e Streaming</b>	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
<b>Qualità e Coerenza</b>	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
<b>Sicurezza e Governance</b>	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):

```
1.from pyspark.sql import SparkSession  
2.spark = SparkSession.builder.getOrCreate()  
3.  
4.# Lettura di un dataset Parquet partizionato  
5.df = spark.read.parquet("s3://data/events/")  
6.  
7.# Filtering coerente con le partizioni per migliorare le prestazioni  
8.filtered = df.filter("country = 'IT' AND year = 2025 AND month = 10")  
9.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):**

```
1.from pyspark.sql.functions import window, col  
2.  
3.stream = (spark.readStream  
4.    .format("kafka")  
5.    .option("subscribe", "events")  
6.    .load())  
7.  
8.agg = (stream  
9.    .withWatermark("event_time", "10 minutes")  
10.   .groupBy(window(col("event_time"), "5 minutes"), col("event_type"))  
11.   .count())  
12.  
13.(agg.writeStream  
14.  .format("delta")  
15.  .option("checkpointLocation", "/chk/events")  
16.  .outputMode("append")  
17.  .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 deduplicazione e schema enforcement a livello di Data Lake.

### Esempio (Python – validazione automatica con Great Expectations):

```
1.import great_expectations as ge  
2.df = ge.from_pandas(spark.table("sales").toPandas())  
3.  
4.df.expect_column_values_to_not_be_null("order_id")  
5.df.expect_column_values_to_be_between("amount", 0, 10000)  
6.results = df.validate()  
7.if not results["success"]:  
8.    raise ValueError("Data quality check failed!")  
9.
```



## 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):**

```
1.from pyspark.sql.functions import sha2, concat_ws, lit  
2.  
3.users = spark.table("customers")  
4.safe_users = users.select(  
5.    sha2(concat_ws(":", "email", lit("pepper")), 256).alias("email_hash"),  
6.    "country", "signup_date"  
7.)  
8.safe_users.write.mode("overwrite").saveAsTable("customers_masked")
```





**Buon Davante a tutti**