Strumenti per i BigData

Programmazione

Gli strumenti per i Big Data rappresentano l’infrastruttura fondamentale che consente di gestire, analizzare e visualizzare enormi quantità di dati eterogenei provenienti da fonti diverse.

Questi strumenti si dividono in più categorie: piattaforme di storage e gestione dei dati (come Hadoop Distributed File System e Apache HBase), strumenti di elaborazione e calcolo distribuito (Apache Spark, Flink, MapReduce) e soluzioni di integrazione e streaming (come Apache Kafka e NiFi).

Il loro scopo principale è garantire la scalabilità orizzontale cioè la capacità di distribuire i dati e i processi su più nodi e la tolleranza ai guasti, elementi chiave per l’affidabilità dei sistemi Big Data.

Parallelamente, esistono strumenti orientati all’analisi, alla visualizzazione e al machine learning, che permettono di estrarre informazioni utili dai dati raccolti.

Framework come TensorFlow, PyTorch o Scikit-learn vengono spesso integrati con i sistemi Big Data per eseguire analisi predittive o classificazioni su larga scala.

Inoltre, strumenti di data visualization come Tableau, Power BI o Apache Superset consentono di trasformare i risultati delle analisi in dashboard interattive e comprensibili, rendendo i dati realmente accessibili ai decisori aziendali.

In sintesi, l’ecosistema dei Big Data si basa su un insieme di tecnologie complementari che lavorano in sinergia, dall’acquisizione fino alla rappresentazione dei risultati.

Storage e gestione dei dati

- **Hadoop HDFS (Hadoop Distributed File System) → Sistema di file distribuito che consente di memorizzare grandi volumi di dati su cluster di macchine, garantendo scalabilità e tolleranza ai guasti.**

- **Apache HBase → Database NoSQL basato su colonne, progettato per l'accesso rapido a dati non strutturati e per integrarsi con Hadoop.**

- **MongoDB → Database NoSQL orientato ai documenti, ideale per gestire dati semi-strutturati come JSON in modo flessibile e scalabile.**

Storage & gestione dati

Teoria (HDFS / MongoDB).

Nel livello storage i requisiti chiave sono scalabilità orizzontale, replica e throughput. HDFS distribuisce file in blocchi su più nodi con replica per tolleranza ai guasti; è ottimo per dati “append-only” e workload batch. I database NoSQL (es. MongoDB) offrono schemi flessibili e accesso a bassa latenza su dati semi-strutturati. Spesso si usa HDFS come data lake “grezzo” e un NoSQL per le query operative.

```
# Carica un CSV locale su HDFS (cartella /data/raw)
hdfs dfs -mkdir -p /data/raw
hdfs dfs -put customers.csv /data/raw/

# Lista e lettura
hdfs dfs -ls /data/raw
hdfs dfs -cat /data/raw/customers.csv | head -n 5
```

```
from pymongo import MongoClient

# Connessione a MongoDB (default locale)
client = MongoClient("mongodb://localhost:27017")
db = client["retail"]
coll = db["customers"]

# Inserimento documento (JSON-like)
doc = {"id": 1001, "name": "Alice", "city": "Torino", "spend": 320.5}
coll.insert_one(doc)

# Query con filtro e proiezione
for c in coll.find({"city": "Torino"}, {"_id": 0, "name": 1, "spend": 1}):
    print(c)
```

Elaborazione e calcolo distribuito

- **Apache Spark → Framework per il calcolo distribuito in memoria, molto più veloce di MapReduce, usato per analisi batch e real-time.**
- **Apache Flink → Sistema per l'elaborazione di flussi di dati in tempo reale (stream processing), utile per applicazioni che richiedono aggiornamenti continui.**
- **MapReduce → Modello di programmazione che suddivide grandi volumi di dati in blocchi elaborati in parallelo, poi aggregati per ottenere risultati finali.**

Elaborazione e calcolo distribuito

Teoria (Spark).

**I motori come Apache Spark eseguono trasformazioni su RDD/DataFrame distribuendo i task sul cluster.
Il calcolo in memoria riduce la latenza rispetto a MapReduce. Il modello è “lazy”: le trasformazioni costruiscono un piano logico, le azioni lo materializzano. Spark è adatto a batch, ML e, con Structured Streaming, anche a near real-time.**

```
1. from pyspark.sql import SparkSession
2. from pyspark.sql.functions import explode, split, col
3.
4. spark = SparkSession.builder.appName("WordCount").getOrCreate()
5.
6. # Carico un testo su HDFS o locale (qui: locale per semplicità)
7. df = spark.read.text("alice.txt")  # una riga per record
8.
9. # Tokenizzo e conteggio parole
10. words = df.select(explode(split(col("value"), r"\W+")).alias("word")).where(col("word") != "")
11. counts = words.groupBy("word").count().orderBy(col("count").desc())
12.
13. counts.show(10, truncate=False)
14. spark.stop()
```

Integrazione e streaming dei dati

- **Apache Kafka → Piattaforma di streaming distribuita che gestisce grandi flussi di dati in tempo reale, spesso usata per connettere sistemi diversi.**
- **Apache NiFi → Strumento per l'automazione dei flussi di dati (dataflow) tra sistemi, con un'interfaccia grafica per definire pipeline di elaborazione.**
- **Airflow → Framework per la gestione e l'automazione dei workflow di dati, utile per pianificare e monitorare processi ETL complessi.**

Integrazione e streaming dei dati

Teoria (Kafka).

Apache Kafka è un log distribuito publish/subscribe a throughput elevato. I producer scrivono su topic partizionati; i consumer leggono in gruppi, scalando orizzontalmente.
È usato per ingest in tempo reale e come “backbone” eventi tra microservizi. La durabilità su disco e la retention configurabile lo rendono adatto a re-processing.

```
# Producer
from kafka import KafkaProducer
import json

producer = KafkaProducer(
    bootstrap_servers=["localhost:9092"],
    value_serializer=lambda v: json.dumps(v).encode("utf-8")
)

for i in range(5):
    event = {"event_id": i, "type": "purchase", "amount": 9.99 + i}
    producer.send("events_purchases", value=event)

producer.flush()
```

```
# Consumer
from kafka import KafkaConsumer
import json

consumer = KafkaConsumer(
    "events_purchases",
    bootstrap_servers=["localhost:9092"],
    auto_offset_reset="earliest",
    value_deserializer=lambda v: json.loads(v.decode("utf-8")),
    group_id="analytics-g1"
)

for msg in consumer:
    print("EVENT:", msg.value)
    break  # demo: leggi un messaggio e termina
```

Analisi, machine learning e intelligenza artificiale

- **TensorFlow → Libreria open source di Google per il deep learning e l'analisi su larga scala.**
- **PyTorch → Framework di machine learning sviluppato da Meta, noto per la flessibilità nello sviluppo di reti neurali.**
- **Scikit-learn → Libreria Python per l'apprendimento automatico tradizionale (classificazione, regressione, clustering) integrabile nei flussi Big Data.**

Analisi, machine learning e intelligenza artificiale

Teoria (scikit-learn / Spark ML).

Per ML “tradizionale” su dataset medio-grandi si usa spesso scikit-learn (feature engineering, cross-validation, pipeline).

Per scala elevata si passa a Spark MLlib, che distribuisce sia trasformazioni sia training.

Best practice: Pipeline end-to-end, train/test split, metriche riproducibili e versionamento dei modelli.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Dataset di esempio
df = pd.DataFrame({
    "age": [25, 45, 36, 52, 23, 40],
    "income": [30_000, 60_000, 45_000, 80_000, 28_000, 55_000],
    "city": ["TO", "MI", "BO", "MI", "TO", "BO"],
    "label": [0, 1, 0, 1, 0, 1]
})

X = df[["age", "income", "city"]]
y = df["label"]

num_cols = ["age", "income"]
cat_cols = ["city"]

pre = ColumnTransformer([
    ("num", StandardScaler(), num_cols),
    # One-hot encoding automatico con get_dummies pre-fit: per semplicità uso pandas
], remainder="passthrough")

X_enc = pd.get_dummies(X, columns=cat_cols)  # semplice OHE

Xtr, Xte, ytr, yte = train_test_split(X_enc, y, test_size=0.33, random_state=42)

pipe = Pipeline([
    ("scaler", StandardScaler(with_mean=False)),  # with_mean=False per sparse
    ("clf", LogisticRegression(max_iter=500))
])

pipe.fit(Xtr, ytr)
pred = pipe.predict(Xte)
print(classification_report(yte, pred))
```

Visualizzazione e business intelligence

- **Tableau → Software di visualizzazione dati che consente di creare dashboard interattive e comprensibili anche a utenti non tecnici.**
- **Power BI → Strumento Microsoft per analizzare e rappresentare dati aziendali integrandosi con Excel, SQL Server e servizi cloud.**
- **Apache Superset → Piattaforma open source per la visualizzazione e l'esplorazione di dataset Big Data, integrabile con database SQL e NoSQL.**

Visualizzazione e business intelligence

Teoria (Superset / Power BI / Tableau).

I tool BI si collegano a data warehouse/lakehouse (es. Presto/Trino, DuckDB, BigQuery, Redshift, Spark SQL), eseguono SQL e forniscono dashboard interattive con filtri, controlli e condivisione.
La logica resta lato DB; il tool gestisce caching, permessi, e governance di dashboard. La chiave è mantenere modelli semantici chiari (metriche, dimensioni, calcoli) e query performanti (indici, partizionamento).

```sql
-- Aggregazione vendite per città e fascia di età
SELECT
  city,
  CASE
    WHEN age < 30 THEN 'U30'
    WHEN age BETWEEN 30 AND 49 THEN '30-49'
    ELSE '50+'
  END AS age_band,
  COUNT(*) AS customers,
  ROUND(SUM(spend), 2) AS total_spend,
  ROUND(AVG(spend), 2) AS avg_spend
FROM analytics.customers_fact
WHERE dt BETWEEN DATE '2025-09-01' AND DATE '2025-09-30'
GROUP BY city, age_band
ORDER BY total_spend DESC;
```

```python
import pandas as pd
import matplotlib.pyplot as plt

data = {
    "city": ["MI", "TO", "BO"],
    "total_spend": [120000, 95000, 70000]
}
df = pd.DataFrame(data)

plt.figure()
plt.bar(df["city"], df["total_spend"])
plt.title("Spesa totale per città (Set 2025)")
plt.xlabel("Città")
plt.ylabel("Spesa totale")
plt.show()
```


Buon Davante a tutti