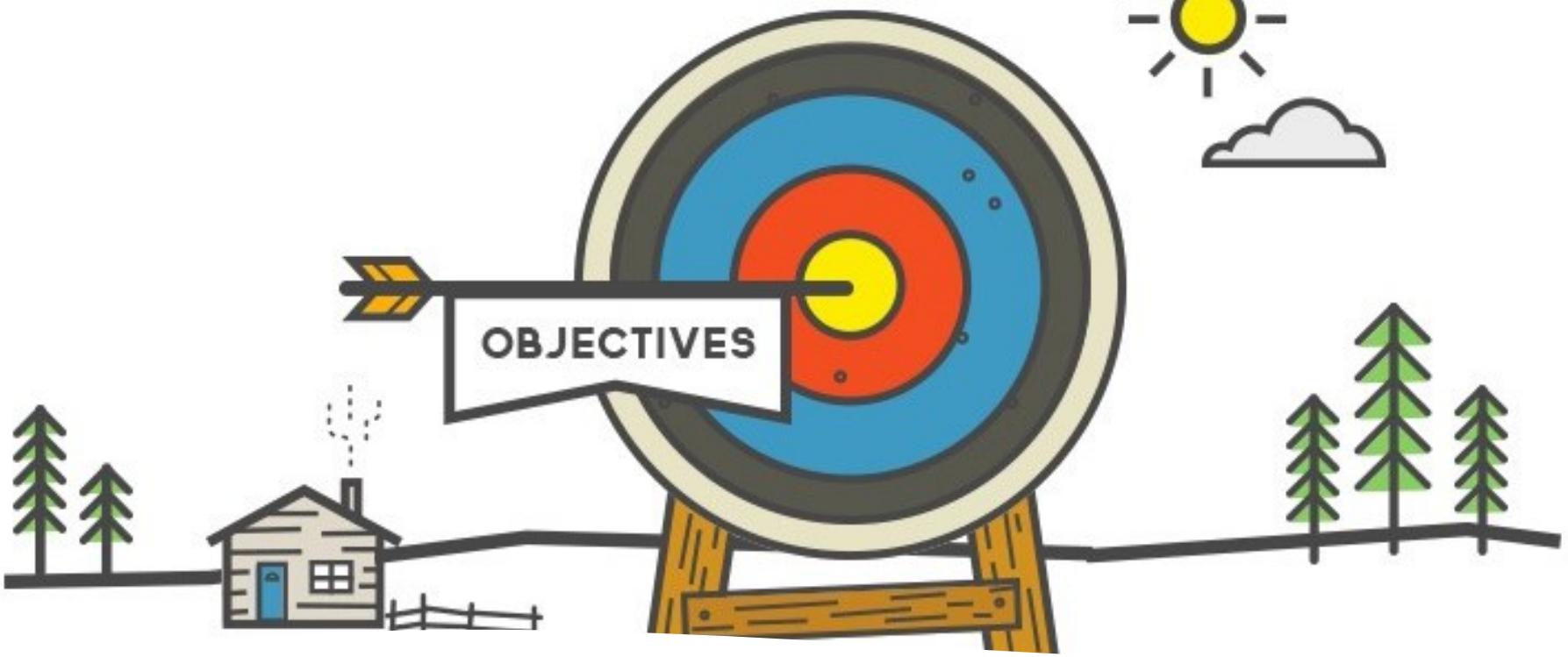
Big Data Processing

Prof. Carlos J. Costa, PhD

Saeed Angorani, DBA



Learning Goals

- Explain why big-data processing frameworks exist (scale, fault tolerance)
- Describe Hadoop and Spark at a high level (how they execute work)
- Compare batch vs real-time processing (trade-offs + when to use)
- Match sustainability use-cases to the right processing style

Big data processing: the core problem

- When data is too large/fast for one machine:
  - Storage is distributed (HDFS / object storage)
  - Compute must run “where the data is”
  - Failures are normal → systems must recover automatically
- Key ideas
  - Parallelism (split work)
  - Partitioning (divide data)
  - Fault tolerance (re-run work safely)

The Hadoop ecosystem

- **Hadoop Distributed File System (HDFS):**
  - Primary data storage system that manages large data sets running on commodity hardware. It also provides high-throughput data access and high fault tolerance.
- **Yet Another Resource Negotiator (YARN):**
  - Cluster resource manager that schedules tasks and allocates resources (e.g., CPU and memory) to applications.
- **Hadoop MapReduce:**
  - Splits big data processing tasks into smaller ones, distributes the small tasks across different nodes, then runs each task.
- **Hadoop Common (Hadoop Core):**
  - Set of common libraries and utilities that the other three modules depend on.

The Spark ecosystem

- Spark Core:
  - Underlying execution engine that schedules and dispatches tasks and coordinates input and output (I/O) operations.
- Spark SQL:
  - Gathers information about structured data to enable users to optimize structured data processing.
- Spark Streaming and Structured Streaming:
  - Both add stream processing capabilities. Spark Streaming takes data from different streaming sources and divides it into micro-batches for a continuous stream. Structured Streaming, built on Spark SQL, reduces latency and simplifies programming.
- Machine Learning Library (MLlib):
  - A set of machine learning algorithms for scalability plus tools for feature selection and building ML pipelines. The primary API for MLlib is DataFrames, which provides uniformity across different programming languages like Java, Scala and Python.
- GraphX:
  - User-friendly computation engine that enables interactive building, modification and analysis of scalable, graph-structured data.

Apache Spark vs. Hadoop

Aspect	Apache Spark	Hadoop
Performance	Faster due to in-memory processing (RAM)	Slower due to disk-based processing (MapReduce)
Cost	Higher cost (requires large RAM for in-memory computation)	Lower cost (uses standard disk storage)
Processing	Ideal for real-time processing and unstructured data streams	Ideal for batch processing and linear data processing
Scalability	Relies on HDFS for handling large-scale data	Highly scalable via HDFS (Hadoop Distributed File System)
Security	Uses authentication (shared secrets, logging); can integrate with Hadoop	More robust security with multiple authentication and access control mechanisms
Machine Learning	Strong support with MLlib (regression, classification, pipelines, etc.)	Limited native ML capabilities

Batch vs Real-time (clear comparison)

- Batch processing
  - Data arrives → stored → processed on schedule (minutes to hours)
  - Pros: simpler, cheaper, easier to make consistent
  - Cons: latency (answers come later)
- Real-time (streaming/near real-time)
  - Data processed continuously (seconds to minutes)
  - Pros: fast reaction, live monitoring
  - Cons: more complexity (state, late events, exactly-once semantics)

Rule of thumb

- If action can wait → batch
- If value depends on immediate response → real-time

Sustainability applications (which fits what?)

- Batch examples
  - Monthly carbon accounting (Scope reporting)
  - Fleet efficiency analysis from historical telematics
  - Waste/recycling audits and trend reports

Sustainability applications (which fits what?)

- Real-time examples
  - Smart grid load balancing, anomaly detection
  - Real-time air quality alerts and forecasting inputs
  - Water leak detection from sensor streams

Sustainability applications (which fits what?)

- Hybrid pattern (common)
  - Stream for alerts + batch for compliance and long-term optimization

Mini activity (15–20 min): choose the architecture Prompt (groups or solo)

- Pick ONE use-case and decide: batch, real-time, or hybrid.
  1. City-wide air quality monitoring
  2. Supply-chain emissions reporting
  3. Building energy optimization
- Deliverables (1 slide or 5 bullets)
  - Data sources
  - Processing choice + why (latency, cost, accuracy)
  - Framework fit (MapReduce vs Spark; Spark Streaming if relevant)
  - One risk (data quality, late data, scale) + mitigation

Takeaways

- MapReduce: durable batch pattern, disk-based between stages
- Spark: faster general engine, supports batch + streaming
- Batch vs real-time is a business latency decision first

Quick check (answers verbally)

- Which approach for monthly compliance reporting?
- Which for live anomaly alerts?

- Why is partitioning important?

Big Data Fundamentals with Apache Spark

(Databricks Serverless Edition)

Prof. Carlos J. Costa, PhD

Saeed Angorani, DBA

What is Big Data?

- Data too large for a single machine
- Requires distributed processing
- Key challenges:
  - Volume
  - Velocity
  - Variety

Why Apache Spark?

- Apache Spark
  - Processes large-scale data in parallel
  - Uses memory + disk efficiently
  - Supports batch + streaming
  - Automatically optimizes execution

Modern Databricks Architecture

Databricks

- Serverless compute (hidden infrastructure)
- No visible cluster management
- Spark runs automatically
- Optimized execution engine (Photon)

Your First Big Data Job

CODE (CELL 1)

```
# Create distributed dataset
df = spark.range(1000000)
df
```

What is Happening Behind the Scenes?

- CODE (CELL 2)

```
df.explain()
```

- EXPECTED OUTPUT (simplified)

```
== Physical Plan ==
PhotonResultStage
+- PhotonColumnarToRow
   +- PhotonRange Range (0, 1000000, step=1, splits=8)
```

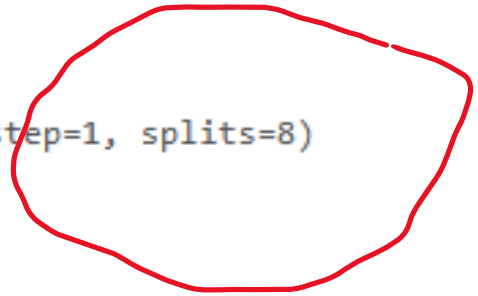
Understanding Parallelism

- CODE (CELL 2)

```
df.explain()
```

- EXPECTED OUTPUT (simplified)

```
== Physical Plan ==
PhotonResultStage
+- PhotonColumnarToRow
   +- PhotonRange Range (0, 1000000, step=1, splits=8)
```



What is a Partition?

- CODE (CELL 3)

```
df2 = spark.range(1000000).repartition(4)
df2.explain()
```

Data Transformation (Big Data Model)

- CODE (CELL 5)

```
df_filtered = df2.filter("id > 500000")
df_filtered
```

When Does Spark Execute?

- CODE (CELL 6)

```
df_filtered.count()
```

Execution Plan Deep Dive

- CODE (CELL 6)

df_filtered.explain(True)

What is Photon?

OUTPUT

- PhotonResultStage
- PhotonColumnarToRow
- PhotonRange

Columnar vs Row Processing

OUTPUT LINE

- PhotonColumnarToRow

Performance Experiment

- CODE (CELL 7)

```
import time
start = time.time()
spark.range(10000000).count()
print(time.time() - start)
```

Big Data Summary Model

- Data is split into partitions
- Execution is distributed automatically
- Spark builds execution plan
- Optimization engine (Photon) executes efficiently
- Computation happens only when needed

Real-World Meaning

- Used in:
- Netflix (recommendations)
- Banks (fraud detection)
- IoT systems (sensor streams)

- Big Data = decision at scale

Final Takeaway

- You don’t manage machines
- You describe data transformations
- Spark handles distribution
- Databricks handles infrastructure

References

- Team, I. C. E. (2021, maio 27). *Hadoop vs. Spark: What's the Difference? | IBM*. https://www.ibm.com/think/insights/hadoop-vs-spark