

# DBD

## SUMMARY

### 1. Big Data Architectures

[Lambda Architecture](#)

[Lambda architecture: A more detailed view](#)

### 2. Hadoop and MapReduce

**2.1 Introduction to Apache Hadoop and the MapReduce programming paradigm**

[Apache Hadoop](#)

[MapReduce](#)

[The MapReduce and Programming Paradigm](#)

**2.1.1 Interaction with HDFS and Hadoop by means of the command line**

[HDFS: command line commands](#)

[HDFS and Hadoop](#)

**2.2 Hadoop implementation of MapReduce**

[MapReduce program in Java](#)

[Basic structure of a MapReduce program \(skeleton\)](#)

**2.2.1** [BigData@Polito environment + Jupyter – How to submit MapReduce jobs on BigData@Polito](#)

**2.3 Design patterns - Part 1**

[Summarization patterns](#)

[Filtering patterns](#)

**2.4 Advanced Topics: Multiple inputs, Multiple outputs, Distributed cache**

**2.5 Design Patterns - Part 2**

[Data Organization Patterns](#)

[Meta patterns](#)

[Join Patterns](#)

**2.6 Relational Algebra/SQL operators**

### 3. Spark

**3.1 Introduction to Apache Spark**

[Resilient Distributed Datasets \(RDDs\)](#)

[Spark Programs](#)

**3.1.1** [How to submit Spark applications](#)

**3.1.2** [How to use Jupyter Notebooks for your Spark applications](#)

**3.2 RDD-based programs**

**3.2.1 RDDs: creation, basic transformations and actions**

[Create RDDs from files](#)

[Create RDDs from a local Python collection](#)

[Save RDDs](#)

[Retrieve the content of RDDs and “store” it in local python variables](#)

[Transformations and Actions](#)

[Transformation](#)

[Actions](#)

[Passing functions to Transformations and Actions](#)

[Basic Transformations](#)

[Basic Actions](#)

**3.2.2 Key-value RDDs: transformations and actions on key-value RDDs**

[Creating RDDs of key-value pairs](#)

[Transformation on RDDs of key-value pairs](#)

[RDD-based programming](#)

[Actions on RDDs of key-value pairs](#)

### **3.2.3 RDDs of numbers**

### **3.2.4 Advanced Topics: Cache, accumulators, broadcast variables, custom partitioners, broadcast join**

Persistence and Cache

Accumulators

Broadcast variables

RDDs and Partitions

#### **3.2.4.1 Introduction to PageRank**

### **3.3 Spark SQL and DataFrames**

#### **3.3.1 Spark SQL**

Operations on DataFrames

DataFrames and the SQL language

Save DataFrames

UDFs: User Defined Functions

Data Warehouse methods

Set methods

Broadcast join

Execution Plan

### **3.4 Data mining and Machine learning algorithms with Spark MLlib**

#### **3.4.1 Introduction and Preprocessing**

DataTypes

Main concepts

Data Preprocessing and Feature Transformations

#### **3.4.2 Classification algorithms**

Structured data classification

Decision trees and structured data

Categorical class labels

Textual data management and classification

Classification: Performance evaluation

Classification: Parameter Tuning

Sparse labeled data: the LIBSVM format

#### **3.4.3 Clustering algorithms**

K-means clustering algorithm

#### **3.4.4 Regression algorithms**

Linear regression and structured data

Linear regression and textual data

Linear regression and parameter setting

#### **3.4.5 Itemset and Association rule mining**

FP-Growth algorithm

Association Rule Mining

### **3.5 GraphX/GraphFrames**

#### **3.5.1 Introduction to GraphX and GraphFrames**

Graph analytics

Spark GraphX and GraphFrames

Building and querying graphs with GraphFrames

Querying the graph

Motif finding

Basic statistics

#### **3.5.2 Graph Algorithms with GraphFrames**

### **3.6 Streaming data analytics**

#### **3.6.1 Spark Streaming Spark Streaming (DStreams)**

Spark Streaming

Spark Streaming Programs

Windowed Computation

Stateful Computation

Transform transformation

#### **3.6.2 Spark Structured Streaming**

Key concepts

Event Time and Window Operations

Watermarking

Join Operations

#### **3.6.3 Introduction to other big stream processing frameworks: Apache Storm, Apache Flink, ...**

Introduction to Apache Storm

Storm core concepts

# 1. Big Data Architectures

Big data solutions typically involve one or more of the following types of workload:

- **Batch processing** of big data sources at rest: large amounts of data that are stored in a static or unchanging state, such as data stored in a database or data warehouse.
- **Real-time processing** of big data in motion
- **Interactive exploration** of big data
- **Predictive analytics** and **machine learning**

Big data architectures are different from traditional DBs. Specific big data architectures are needed when:

- Store and process large volumes of data
- transform unstructured data → structured data: analysis and reporting
- Capture, process, and analyze **unbounded streams of data in real time**, or with **low latency**

→ The most used architecture for big data is the **Lambda Architecture** (Nathan Marz in 2011).

## Lambda Architecture



A **Lambda Architecture** is a big data processing architecture designed to handle massive quantities of data by taking advantage of both batch and real-time processing methods. The architecture is composed of three layers: the batch layer, the speed layer, and the serving layer. Data is processed simultaneously in both the batch and speed layers, and the results are combined and made available for querying in the serving layer. The architecture is fault-tolerant, scalable, and able to handle large volumes of input data, making it a popular choice for big data processing.

Lambda architecture depends on a data model with an append-only, immutable data source that serves as a system of record. It is intended for ingesting and processing timestamped events that are appended to existing events rather than overwriting them. State is determined from the natural time-based ordering of the data.

### Requirements:

- **Fault-tolerant** against both hardware failures and human errors
- Support variety of use cases that include **low latency querying as well as updates**
- Linear **scale-out capabilities**
- **Extensible**: so that the system is manageable and can accommodate newer features easily

### Queries properties:

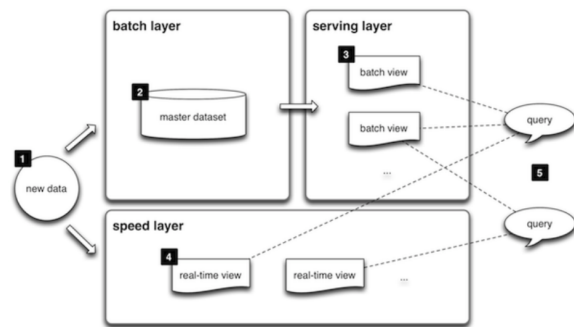
- **Latency**: the time it takes to run a query
- **Timeliness**: how up to date the query results are (**freshness** and **consistency**)
- **Accuracy**: Tradeoff between performance and scalability (**approximations**)

Lambda Architecture is based on 2 data paths:

- **Batch layer** (cold path): It stores all of the incoming data in its raw form and performs batch processing on the data. The result of this processing is stored as batch views.
- **Speed layer** (hot path): It analyzes data in real time. This path is designed for **low latency**, at the expense of **accuracy**.

The **Batch layer** is responsible for processing large volumes of data that are stored in a static or unchanging state, such as data stored in a database or data warehouse. The **Speed layer**, on the other hand, is responsible for processing unbounded streams of data in real time or with low latency. These two paths are then combined in the serving layer to provide a complete view of the data.

1. All data entering the system is dispatched to both the batch layer and the speed layer for processing
2. The batch layer has two functions:
  - managing the master dataset, an immutable, append-only set of raw data)
  - to pre-compute the batch views
3. The serving layer indexes the batch views so that they can be queried in low-latency, ad-hoc way
4. The speed layer compensates for the high latency of updates to the serving layer and deals with recent data only



→ Any incoming query can be answered by merging results from batch views and realtime views

### Lambda architecture: A more detailed view

