Introduction to Big Data

The 5 V's and Why It Matters

Professor Anis Koubaa

SE 446
Alfaisal University
https://github.com/aniskoubaa/big_data_course

Spring 2026

جامعة الفيصل

Outline

Motivating Question

How much data is generated every minute?

Every 60 seconds:

- 500 hours of YouTube video uploaded
- 6 million Google searches
- 500,000 tweets posted
- 200 million emails sent

The Challenge:

- Too large for one machine
- Too fast for batch processing
- Too complex for simple queries
- Traditional DBs can't cope

Welcome to the Big Data Era

We need new tools and techniques to handle this scale!

What is Big Data?

Definition

Big Data refers to datasets that are too large, fast, or complex for traditional data processing tools.

- Cannot fit on a single machine
- Cannot be processed in reasonable time
- Requires **distributed computing**

Key Insight

It's not just about *size* — it's about the *challenges* of handling the data.

The Scale of Big Data

Company	Data Generated	Scale
Facebook	4 PB / day	250 billion photos
YouTube	500 hours video / minute	1 billion hours watched/day
Twitter	500 million tweets / day	6,000 tweets / second
Google	20 PB processed / day	3.5 billion searches / day

Perspective

1 Petabyte = 1,000 Terabytes = 1,000,000 Gigabytes

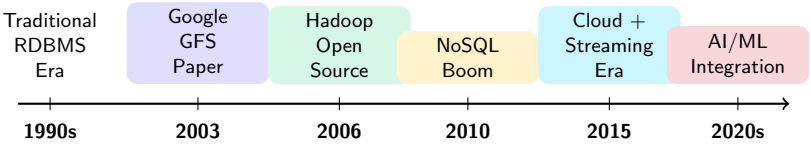
Big Data Use Cases & Industry Applications

Industry	Applications	Scale
Healthcare	Genomics, Patient Records, Drug Discovery, Epidemic Tracking	30+ PB human genome data
Finance	Fraud Detection, High-Frequency Trading, Credit Scoring, Anti-Money Laundering	Millions of transactions/sec
Smart Cities	Traffic Sensors, IoT Monitoring, Energy Grids, Pollution Tracking	Billions of sensor readings/day
Retail	Recommendation Engines, Demand Forecasting, Inventory Optimization	Amazon: 1.6M packages/day
Cybersecurity	Log Analysis, Threat Detection, Intrusion Prevention, SIEM	10TB+ logs/day in enterprises

Why This Matters

Every industry generates and depends on Big Data. The tools you learn in this course apply across all domains!

Historical Evolution of Big Data



Key Milestones:

- **2003**: Google GFS paper → distributed storage
- **2004**: Google MapReduce paper → parallel processing
- **2006**: Yahoo! releases Hadoop as open-source

Why This Matters:

- Hadoop was born from *real problems* at Google
- Open-source democratized Big Data
- Today: Cloud-native, real-time, AI-powered

Google File System (GFS) & The Birth of HDFS

The Game Changer: GFS Paper (2003)

Authors: Ghemawat, Gobioff, Leung (Google)
"The Google File System" outlined storing the web index on **cheap, unreliable commodity hardware**.

Key Innovations

1. **Failure is Norm**: Design for constant hardware failure.
2. **Huge Files**: Optimized for multi-GB/TB files.
3. **Append-Only**: Write once, read many (WORM).
4. **Data Locality**: Move computation to data.

Influence on HDFS

HDFS is the open-source clone of GFS.

Google GFS	Hadoop HDFS
Master	NameNode
ChunkServer	DataNode
Chunk	Block

Without this paper, modern Big Data might not exist!

The WORM Model: Write Once, Read Many

What is WORM?

Write Once, Read Many

- **Write Once**: Data is written or appended, then becomes **immutable** (cannot be modified).
- **Read Many**: Data can be accessed unlimited times.

Key Concept: Immutability

You don't *update* old records; you **add new events**.

Why WORM for Big Data?

1. **Simplifies Consistency**: No need for complex locking (as in RDBMS).
2. **Fault Tolerance**: Easier to replicate immutable files.
3. **History Preservation**: Full audit trail (e.g., bank ledgers).

Examples:

- HDFS Files (Append-only)
- Kafka Topics (Log events)
- Blockchain (Ledgers)

The Modern Big Data Ecosystem

Big Data = Storage + Compute + Ingestion + Analytics + Visualization

Layer	Category	Technologies
Storage	Data Lakes	S3, ADLS, MinIO, HDFS
	Data Warehouses	BigQuery, Snowflake, Redshift
Processing	Batch	MapReduce, Spark
	Stream	Kafka, Flink, Storm
	Interactive Query	Presto, Trino, Athena
NoSQL DBs	Key-Value	Redis, DynamoDB
	Document	MongoDB, Couchbase
	Columnar	Cassandra, HBase
	Graph	Neo4j, Neptune
Analytics	BI / Visualization	Tableau, Looker, Power BI
	ML Platforms	Spark MLlib, SageMaker

Course Focus

We focus on the **core**: HDFS, MapReduce, Hive, Spark, Kafka — the foundation for all the above.

Cloud & Modern Big Data Stack

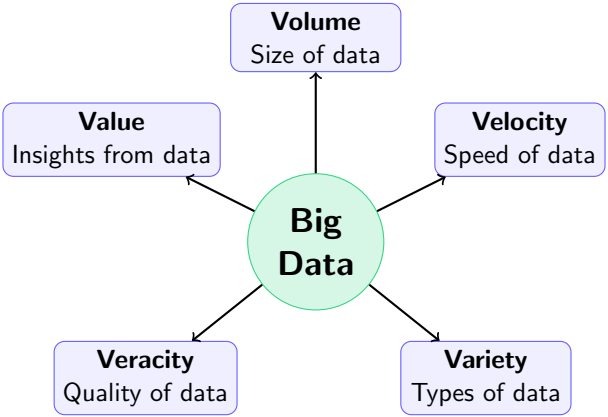
In 2026, Big Data is **cloud-native**. Key platforms you'll encounter:

Provider	Category	Services
AWS	Storage, Compute, ETL	S3, EMR (Hadoop/Spark), Glue, Kinesis, Redshift
GCP	Analytics, Streaming	BigQuery, Dataflow, Dataproc, Pub/Sub
Azure	Data Lake, Warehousing	ADLS, Synapse Analytics, HDInsight, Event Hubs
Databricks	Unified Analytics	Lakehouse, MLflow, Delta Lake, Spark (founders!)

Future-Proof Your Skills

The concepts you learn (HDFS, Spark, Kafka) translate directly to cloud equivalents. Databricks Community Edition (free) is our lab environment!

The 5 V's of Big Data



Volume: Size of Data

Challenge: Exponential Growth

Datasets have grown from Terabytes to **Petabytes** and **Exabytes**.

- Traditional RDBMS cannot scale horizontally.
- Too expensive to store on enterprise SANs.
- Single servers cannot process the data in time.

Examples:

- **CERN**: 1 Petabyte of particle collision data per day.
- **Genomics**: 200 GB+ per human genome sequence.
- **Social**: Facebook stores Exabytes of photos/videos.

Solution

Distributed Storage

- **HDFS**: Store across cheap commodity hardware.
- **Cloud**: S3 / ADLS (Scale to infinity).
- **Scale-Out**: Just add more nodes.

Velocity: Speed of Data

Challenge: The Need for Speed

Data is generated continuously at high speed. Batch processing (nightly jobs) creates **latency**, making insights outdated.

- "Stale" data loses value quickly.
- Traditional disks are too slow for ingestion.

Examples:

- **Finance**: High-frequency trading (microseconds).
- **IoT**: Industrial sensors (thousands/sec) detecting failure.
- **Cybersecurity**: Real-time intrusion detection.

Solution

Stream Processing

- **Ingest**: Kafka (Buffer events).
- **Process**: Spark Streaming / Flink.
- **Action**: Real-time alerts.

Variety: Diversity of Data

Challenge: Rigid Schemas

80% of modern data is unstructured or semi-structured.

- Relational Databases (SQL) require fixed tables.
- Changing schema breaks applications.
- Data comes as Text, Video, XML, JSON, Audio.

Examples:

- **Unstructured**: PDF Contracts, CCTV Video, Call Logs.
- **Semi-Structured**: IoT JSON payloads, XML Configs.
- **Graph**: Social connections, Fraud networks.

Solution

Flexible Storage

- **NoSQL**: MongoDB, Cassandra.
- **Data Lakes**: Store raw now, schema later.
- **Formats**: Parquet, JSON, Avro.

Veracity: Reliability of Data

Challenge: Trustworthiness

Big Data is often messy, incomplete, noise-filled, or inconsistent.

- **"Garbage In, Garbage Out"** (GIGO).
- Identifying fake data, bots, or sensor errors.
- Data from diverse sources might conflict.

Impact: Poor quality leads to wrong strategic decisions and loss of revenue.

Solution

Data Quality

- **Cleanse**: ETL pipelines to fix errors.
- **Validate**: Schema checks.
- **Lineage**: Track data origin.

Value: Worth of Data

Challenge: Turning Bits into Money

Data itself is a **cost** (storage/compute) until it creates value.

- Massive data graveyards (Data Swamps).
- Finding the "signal" in the "noise".
- Operationalizing insights is difficult.

Goal: Transform raw data into **Actionable Intelligence**.

Solution

Advanced Analytics

- **BI**: Dashboards (Tableau).
- **ML/AI**: Predictive models.
- **Action**: Auto-bidding, Recommendations.

Concept: Schema-on-Write vs. Schema-on-Read

1. Schema-on-Write (SQL)

- **Definition**: You must define the table structure (columns, types) **before** you can load data.
- **Analogy**: Like filing documents into specific, labeled folders. If a document doesn't fit existing folders, you can't file it.
- **Pros**: Fast reads, strict data quality.
- **Cons**: Slow writes, very rigid.

2. Schema-on-Read (Big Data)

- **Definition**: You load raw data as-is. You define the structure only when you **read/query** it.
- **Analogy**: Like dumping everything into a big box. You decide how to organize it only when you start looking for something.
- **Pros**: Fast writes, maximum flexibility.
- **Cons**: Slower reads (parsing needed).

Big Data systems prefer Schema-on-Read to handle Variety (JSON, Logs, Images).

Example: JSON Handling (Schema-on-Read)

Raw Data (JSON Log)

```
{
  "user_id": 101,
  "action": "click",
  "meta": {
    "page": "home",
    "browser": "Chrome"
  }
}
{
  "user_id": 102,
  "action": "purchase",
  "amount": 50.0   // New field!
}
```

Processing (Spark/Pandas)

- We didn't need to CREATE TABLE first.
- We didn't crash when line 2 had an extra "amount" field.
- The schema was **inferred** automatically when we read the file.

Resulting DataFrame:

user_id	action	meta.page	amount
101	click	home	NULL
102	purchase	NULL	50.0

Structured vs. Unstructured Data

Structured Data

- **Definition**: Highly organized, fixed format.
- **Model**: Rows & Columns (RDBMS).
- **Examples**:
  - Excel spreadsheets
  - SQL Databases
  - Transaction logs
- **Search**: Easy to search.

Unstructured Data

- **Definition**: No tangible structure.
- **Model**: Binary large objects (BLOBs).
- **Examples**:
  - Images, Videos, Audio
  - Social Media Posts
  - PDFs, Emails
- **Search**: Requires advanced AI/ML.

Key Stats: Unstructured data accounts for $> 80\%$ of all enterprise data!

Semi-Structured Data: The Missing Middle

Definition

Data with **some organization** but not fixed rows/columns. Self-describing with tags, keys, or markers.

Characteristics:

- Flexible, evolving schema
- Human & machine readable
- Hierarchical or nested structure
- No strict RDBMS constraints

Common Examples

- **JSON**: APIs, NoSQL (MongoDB)
- **XML**: Configuration, legacy systems
- **CSV**: Evolving headers, mixed types
- **Logs**: Apache, Nginx, application
- **Sensor Data**: IoT payloads

Why It Matters

80% of Kafka + IoT + API data is semi-structured! Tools like Spark and Hive have native support for JSON/XML parsing.

Big Data File Formats: Row vs. Columnar

Row-Based Formats

- **Examples**: CSV, JSON, Avro
- **Storage**: Row 1, Row 2, Row 3...
- **Best For**: Insert-heavy, full-row access
- **Limitation**: Read entire row even for 1 column

Columnar Formats

- **Examples**: Parquet, ORC
- **Storage**: Col A, Col A, Col A... then Col B...
- **Best For**: Analytics (SELECT specific columns)
- **Benefit**: Read only needed columns $\rightarrow$ 10x faster!

Format	Type	Compression	Use Case
CSV/JSON	Row	Poor	Interchange, APIs
Avro	Row	Good	Kafka, streaming
Parquet	Columnar	Excellent	Spark, Hive, BigQuery
ORC	Columnar	Excellent	Hive (optimized)

Industry standard: Store raw data in Data Lake, convert to Parquet for analytics!

Row vs. Columnar: Visual Example

Sample Data Table (3 rows, 3 columns)

Name	Age	City
Alice	25	Riyadh
Bob	30	Jeddah
Carol	28	Dammam

Row Storage (CSV)

```
Alice, 25, Riyadh
Bob, 30, Jeddah
Carol, 28, Dammam
```

Query `SELECT Age`: Read ALL data!

Columnar Storage (Parquet)

Name: `Alice, Bob, Carol`
Age: `25, 30, 28`
City: `Riyadh, Jeddah, Dammam`

Query `SELECT Age`: Read Age column only!

Result: Columnar reads 33% of data vs 100% for row-based → 3x faster!

Limitations of RDBMS

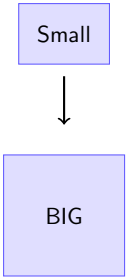
Challenge	RDBMS	Big Data
Scaling	Vertical (bigger server)	Horizontal (add nodes)
Schema	Fixed, predefined	Flexible, schema-on-read
Data Types	Structured only	All types
Cost	Expensive hardware	Commodity hardware
Speed	Slow for massive writes	Parallel distributed writes

The Solution

Distributed Systems: Hadoop, Spark, NoSQL databases

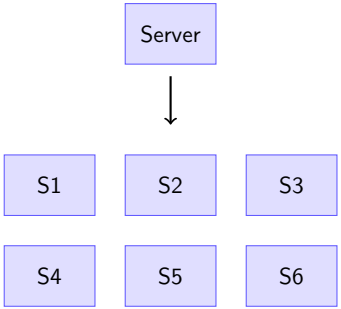
Vertical vs. Horizontal Scaling

Vertical Scaling



Expensive, limited

Horizontal Scaling



Scalable, cost-effective

Vertical Scaling vs. Horizontal Scaling (Deep Dive)

Vertical Scaling (Scale Up)

Increasing the capacity of a SINGLE machine (e.g., adding more RAM, stronger CPU).

Horizontal Scaling (Scale Out)

Adding MORE machines (nodes) to the system to work as a single cluster.

Feature	Vertical Scaling	Horizontal Scaling
Cost	High (Exponential)	Low (Commodity hardware)
Complexity	Low (Single System)	High (Distributed System)
Fault Tolerance	Single Point of Failure	High (Replication)
Downtime	Required for upgrades	Zero (Add nodes live)
Limit	Hardware Ceiling	Virtually Unlimited

The Secret Sauce: Data Locality

Traditional Approach

Move Data to Code

Query → Fetch 1 PB over network → Process

Problem: Network = Bottleneck!

Big Data Approach

Move Code to Data

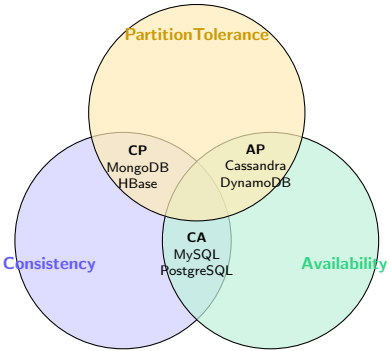
Send tiny Spark/MR task → Run locally on each node

Result: Parallelism, no network flood!

Metric	Traditional	Data Locality
Network Traffic	1 PB (entire dataset)	<1 MB (code only)
Processing Speed	Hours/Days	Minutes
Scalability	Limited by bandwidth	Scales with nodes

This is why HDFS stores data across nodes — so Spark/MapReduce can process locally!

The Theoretical Limit: CAP Theorem



Eric Brewer's Theorem (2000)

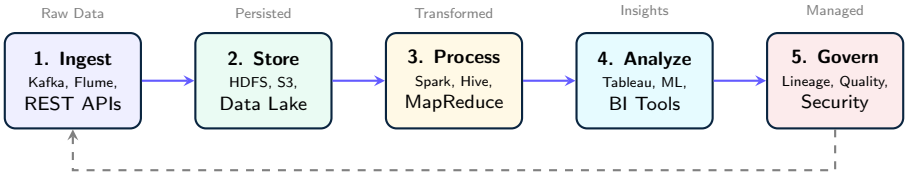
In a distributed system, you can only guarantee **two** of three properties:

- **C (Consistency)**: All nodes see the same data at the same time.
- **A (Availability)**: Every request gets a response (even if stale).
- **P (Partition Tolerance)**: System works despite network failures.

Why This Matters

Big Data systems (Hadoop, NoSQL) prioritize **P** (network failures are inevitable), forcing a trade-off between C and A.

The Big Data Pipeline Lifecycle



Processing Paradigms

- **Batch**: Daily/hourly (Spark, MapReduce)
- **Stream**: Real-time (Kafka, Flink)
- **Interactive**: Ad-hoc SQL (Presto)

ETL vs ELT

- **ETL**: Extract → Transform → Load (Data Warehouse)
- **ELT**: Load raw first, transform later (Data Lake)

How ELT Differs From ETL

Feature	ETL (Extract-Transform-Load)	ELT (Extract-Load-Transform)
Order	Extract → Transform → Load	Extract → Load → Transform
Transformation	Before loading	After loading
Compute Loc.	ETL server / Middleware	Data Warehouse / Data Lake
Best For	Traditional BI, Clean Data	Big Data, ML, Data Science
Data Volume	Smaller	Very large (TB–PB)

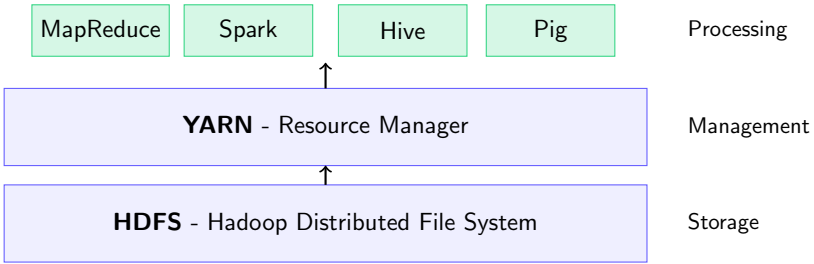
ETL

Traditional approach. Clean data *before* storing. Good for compliance and predictable reports.

ELT (Modern)

Cloud-native. Load raw data **fast**, transform on demand. Enables "Schema-on-Read" and AI/ML on raw data.

The Hadoop Ecosystem



This Course Covers

HDFS, MapReduce, Hive, Spark, Kafka (Streaming)

Appendix: Key Terminology

Storage Concepts

- **Data Lake**: Central repository storing huge amounts of raw data (structured, semi-structured, unstructured) "as-is".
- **Data Warehouse**: Repository for structured, filtered data already processed for specific purposes (reporting).
- **Schema-on-Read**: defining the structure of data at the time of analysis (query), not when storing it.

Processing Concepts

- **ETL**: Extract, Transform, Load. Data is cleaned **before storage** (Data Warehouse style).
- **ELT**: Extract, Load, Transform. Raw data stored first, transformed later (Data Lake style).
- **Data Locality**: Moving computation code to where the data resides (instead of moving data to code).

Big Data Career Paths

Learning Big Data opens doors to multiple career paths:

Data Engineer

- Build pipelines
- Manage storage (HDFS, S3)
- Spark, Kafka, Airflow

Avg: $130K/year

Data Scientist

- ML on Big Data
- Statistical analysis
- Python, Spark MLlib

Avg: $140K/year

Data Analyst

- Dashboards & BI
- SQL, Hive, Presto
- Tableau, Power BI

Avg: $85K/year

This course gives you foundational skills for **all three paths**!

Summary: Key Takeaways

1. **Big Data** = Volume + Velocity + Variety + Veracity + Value
2. **Three data types**: Structured, Semi-structured, Unstructured
3. **RDBMS limitations** solved by distributed systems
4. **Hadoop Ecosystem**: HDFS (storage), YARN (resources), Spark/Hive (processing)

Next Session

HDFS Architecture: NameNode, DataNode, Replication

Homework

1. Watch the pre-class video for Session 2B:
   - "HDFS Tutorial" - Edureka (20 min)
2. Setup your accounts (if you haven't):
   - Google Colab: `colab.google.com`
   - GitHub: `github.com`
3. Review the notebook from today's session

Questions?

Prof. Anis Koubaa
akoubaa@alfaisal.edu