

VU LEARNING HUB

vulearninghub.xyz

HANDOUT

CS441

Big Data Concepts

CS441 by VU Learning HUB

Prepared for VU students • Free study resources

<https://vulearninghub.xyz>

CS441

Big Data Concepts

STUDY HANDOUT

A complete topic-wise study handout for CS441 Big Data Concepts, prepared from big-data video lecture series (Hadoop ecosystem, HDFS, MapReduce, Spark). Covers the 5 Vs, Hadoop and its ecosystem, HDFS, MapReduce, YARN, Spark, NoSQL databases, Hive, Pig, stream processing, and big-data applications - with definitions, explanations, examples and exam pointers.

VU Learning HUB | vulearninghub.xyz

Table of Contents

1. What is Big Data? The 5 Vs
2. History and Drivers of Big Data
3. Structured, Semi-structured and Unstructured Data
4. Why Traditional Databases Are Not Enough
5. Introduction to Hadoop
6. HDFS Architecture
7. HDFS Read and Write Operations
8. MapReduce Programming Model
9. MapReduce Internals: Splits, Shuffling and Fault Tolerance
10. YARN: Yet Another Resource Negotiator
11. Apache Spark: In-Memory Processing
12. Spark SQL, Streaming and MLlib
13. NoSQL Databases: Concepts and CAP Theorem
14. NoSQL Data Models I: Document and Key-Value
15. NoSQL Data Models II: Column-Family and Graph
16. Apache Hive
17. Apache Pig
18. Apache HBase
19. Data Ingestion: Sqoop and Flume
20. Apache Kafka and Stream Processing
21. Data Warehouses vs Data Lakes
22. Big Data Analytics: Descriptive to Prescriptive
23. Data Visualisation for Big Data
24. Big Data in the Cloud
25. Privacy, Security and Ethics in Big Data
26. Big Data Applications and Case Studies
27. Revision: Midterm Practice Set
28. Revision: Final-Term Practice Set

Lecture 1: What is Big Data? The 5 Vs

Key terms: big data, 5 Vs: volume, velocity, variety, veracity, value

Big Data refers to datasets so large, fast-changing and diverse that traditional tools (single-server databases, spreadsheets) cannot store, process or analyse them. Doug Laney's 2001 3Vs model - Volume, Velocity, Variety - was later extended to 5 Vs with Veracity (trustworthiness of data) and Value (useful insight extracted).

- Volume: terabytes to zettabytes - e.g. Walmart handles over a million transactions per hour.
- Velocity: speed of generation and required processing - stock markets need microsecond updates.
- Variety: structured (tables), semi-structured (JSON/XML), unstructured (video, text, images).
- Veracity: noisy, uncertain data - fake reviews, sensor errors. Value: the business benefit extracted.

EXAM POINTER: Define big data and explain each of the 5 Vs with an example - the opening question of the course.

Lecture 2: History and Drivers of Big Data

Key terms: Moore's law, Hadoop 2006, GFS, MapReduce paper, data explosion

Before 2000, data was mostly structured and lived in RDBMS. Google's 2003-2004 papers on GFS (Google File System) and MapReduce showed how to store and process web-scale data on commodity hardware. Doug Cutting's open-source implementation became Apache Hadoop (2006). After 2010, smartphones, social media and IoT caused an explosion of unstructured data, making big-data tools mainstream.

- 2001: Laney's 3Vs. 2006: Hadoop released. 2014: Spark gains popularity for in-memory speed.
- Drivers: cheaper storage, faster networks, mobile/IoT sensors, and demand for data-driven decisions.
- Data is now called 'the new oil' - but like oil, it must be refined (processed) to be useful.

Lecture 3: Structured, Semi-structured and Unstructured Data

Key terms: structured, semi-structured, unstructured data, schema

Structured data fits neat tables with a fixed schema (relational databases, spreadsheets). Semi-structured data has organisational markers but no rigid schema (JSON, XML, log files).

Unstructured data has no predefined format - text documents, images, audio, video - and makes up roughly 80% of all data. Big-data tools must handle all three, whereas traditional databases handle mainly structured data.

- Example: a weather system produces structured sensor readings plus unstructured satellite images.
- Schema-on-write (RDBMS) vs schema-on-read (data lakes): big data often applies structure only when analysing.
- Text analytics, image recognition and NLP unlock value from unstructured data.

Lecture 4: Why Traditional Databases Are Not Enough

Key terms: scalability, vertical vs horizontal scaling, RDBMS limits

Traditional RDBMS scale vertically (bigger, costlier servers) and enforce rigid schemas and ACID transactions - excellent for banking records but poor for petabytes of varied, fast-arriving data. They become bottlenecks in storage capacity, write throughput and flexible querying. Big-data systems scale horizontally: add cheap commodity machines to a cluster and distribute both storage and computation.

- Vertical scaling hits physical and cost limits; horizontal scaling is near-linear and fault-tolerant.
- CAP theorem trade-offs: distributed systems sacrifice some consistency for availability and partition tolerance.
- Use RDBMS for structured transactional workloads; big-data platforms for volume/variety/velocity workloads.

EXAM POINTER: Compare vertical vs horizontal scaling with a diagram - a favourite conceptual question.

Lecture 5: Introduction to Hadoop

Key terms: Apache Hadoop, distributed storage, distributed processing, commodity hardware

Apache Hadoop is an open-source framework for distributed storage and processing of very large datasets across clusters of commodity computers. Its two original core components: HDFS (Hadoop Distributed File System) for storage and MapReduce for processing. Hadoop assumes hardware will fail and handles faults in software through replication and task retries.

- Design principles: move computation to data (not data to computation), scale out with cheap nodes, tolerate failures automatically.
- Hadoop 1.x: HDFS + MapReduce (with JobTracker). Hadoop 2.x added YARN for general resource management.

- Ecosystem tools (Hive, Pig, HBase, Sqoop, Flume, Spark) run on top of the Hadoop core.

Lecture 6: HDFS Architecture

Key terms: HDFS, NameNode, DataNode, blocks, replication

HDFS stores files as blocks (default 128 MB) distributed across DataNodes. The NameNode (master) keeps metadata - the file-to-block mapping - in memory, while DataNodes (slaves) store the actual blocks. Each block is replicated (default 3 copies) on different nodes and racks, giving fault tolerance: if a node dies, data is still available and is re-replicated automatically.

- NameNode is the single point of failure in classic HDFS - addressed by HA (standby NameNode) and federation.
- Write-once, read-many: HDFS suits large sequential reads, not random updates.
- Rack awareness: replicas are spread across racks so a rack failure does not lose data.

EXAM POINTER: Draw the HDFS architecture showing NameNode, DataNodes, blocks and replication - asked in almost every paper.

Lecture 7: HDFS Read and Write Operations

Key terms: HDFS write pipeline, replication pipeline, read path

Write: the client asks the NameNode for DataNode locations, then streams block data through a pipeline of DataNodes (each forwards to the next); acknowledgements flow back up the pipeline. Read: the client gets block locations from the NameNode and reads directly from the nearest DataNode - data transfer bypasses the NameNode, which only handles metadata.

- Coherency: a file being written is visible to readers only after it is closed (or hflush/hsync).
- Checksums on blocks detect corruption; the client re-reads from another replica on failure.
- Secondary NameNode (or Checkpoint Node) merges FsImage and EditLog - it is NOT a backup NameNode.

Lecture 8: MapReduce Programming Model

Key terms: MapReduce, map, shuffle, reduce, key-value pairs

MapReduce processes huge datasets in parallel in two phases. Map: input splits are processed independently, each emitting intermediate key-value pairs. Shuffle/sort: pairs with the same key are grouped and sent to reducers. Reduce: each reducer aggregates the values for its keys and writes output. The classic example is word count: mappers emit (word, 1), reducers sum counts per word.

- The framework handles parallelisation, data distribution, fault tolerance and load balancing - the programmer writes only map and reduce functions.
- Combiners (mini-reducers on map output) cut network traffic.
- Limitation: batch-oriented with heavy disk I/O between jobs - slow for iterative algorithms, which motivated Spark.

EXAM POINTER: Trace the word-count example through map, shuffle and reduce stages - the standard MapReduce question.

Lecture 9: MapReduce Internals: Splits, Shuffling and Fault Tolerance

Key terms: input splits, partitioning, speculative execution

Input data is divided into splits (usually one per HDFS block); each split gets one map task, preferably scheduled on the node holding the data (data locality). Partitioning decides which reducer receives each key (default: hash partitioning). If a task is slow or fails, the framework re-runs it elsewhere - speculative execution launches backup copies of straggler tasks.

- Data locality levels: node-local, rack-local, off-rack - the scheduler prefers node-local.
- Counters track progress; task attempts are retried a fixed number of times before the job fails.
- Skewed keys (one key with huge data) create straggler reducers - use combiners or salting.

Lecture 10: YARN: Yet Another Resource Negotiator

Key terms: YARN, ResourceManager, NodeManager, ApplicationMaster, containers

YARN (Hadoop 2.x) separates resource management from MapReduce. The cluster-wide ResourceManager allocates resources; per-node NodeManagers monitor and launch containers (CPU + memory slices); each application runs an ApplicationMaster that negotiates containers from the ResourceManager and coordinates its tasks. This lets non-MapReduce engines - Spark, Tez, Storm - run on the same cluster.

- Schedulers: FIFO, Capacity (queues with guaranteed shares), Fair (equal shares).
- Containers are the unit of allocation - e.g. 2 GB RAM + 1 vcore.
- YARN turned Hadoop from a batch engine into a general data-processing platform.

Lecture 11: Apache Spark: In-Memory Processing

Key terms: Apache Spark, RDD, in-memory, DAG

Apache Spark is a fast, general-purpose cluster engine that keeps intermediate data in memory instead of writing to disk between stages, making iterative algorithms (machine learning, graph processing) 10-100x faster than MapReduce. Its core abstraction is the RDD (Resilient Distributed Dataset) - an immutable, partitioned collection with lineage for fault recovery - plus higher-level DataFrames/Datasets today.

- Transformations (map, filter) are lazy; actions (count, collect) trigger execution via a DAG scheduler.
- Lineage: lost partitions are recomputed from source - no replication needed.
- Spark modules: Spark SQL, Spark Streaming (micro-batch), MLlib, GraphX.

EXAM POINTER: Explain why Spark is faster than MapReduce for iterative jobs, using the RDD/lineage concept.

Lecture 12: Spark SQL, Streaming and MLlib

Key terms: Spark SQL, DataFrame, Spark Streaming, MLlib

Spark SQL queries structured data with SQL through DataFrames - distributed tables with schema - and the Catalyst optimiser. Spark Streaming processes live data in micro-batches (discretised streams), enabling near-real-time analytics on logs, sensor feeds and social media. MLlib provides distributed machine-learning algorithms: classification, regression, clustering and collaborative filtering.

- Structured Streaming (newer API) treats streams as unbounded tables with event-time support.
- Typical pipeline: Kafka -> Spark Streaming -> dashboard/alerts.
- MLlib pipelines chain feature transformers and estimators for reusable ML workflows.

Lecture 13: NoSQL Databases: Concepts and CAP Theorem

Key terms: NoSQL, CAP theorem, BASE, eventual consistency

NoSQL ('not only SQL') databases trade the rigid schema and strong consistency of RDBMS for horizontal scalability and flexible data models. The CAP theorem states a distributed store can guarantee only two of Consistency, Availability and Partition tolerance - during network partitions, systems choose between consistency (CP) and availability (AP). BASE (Basically Available, Soft state, Eventually consistent) is the NoSQL counterpart of ACID.

- ACID: atomicity, consistency, isolation, durability - for transactions. BASE: availability-first for scale.
- Choose CP (HBase, MongoDB) when correctness matters; AP (Cassandra, DynamoDB) when uptime matters.

- NoSQL is not 'no SQL' - many offer SQL-like query languages.

EXAM POINTER: State the CAP theorem and classify two NoSQL systems as CP or AP with justification.

Lecture 14: NoSQL Data Models I: Document and Key-Value

Key terms: document store, key-value store, MongoDB, Redis

Document stores (MongoDB, CouchDB) keep self-describing JSON-like documents in collections; each document can have different fields, and queries use indexes and aggregation pipelines. Key-value stores (Redis, DynamoDB) are the simplest model: opaque values accessed by key with $O(1)$ lookups - ideal for sessions, caches, counters and shopping carts. Redis adds data structures (lists, sets, hashes) and in-memory speed.

- Documents suit catalogues, profiles and content with varying attributes.
- Key-value suits high-speed lookups where the value needs no querying.
- Trade-off: flexible schema shifts data-validation responsibility to the application.

Lecture 15: NoSQL Data Models II: Column-Family and Graph

Key terms: column-family, HBase, Cassandra, graph database, Neo4j

Column-family stores (HBase, Cassandra) organise data as rows with column families; sparse, versioned and sorted by row key - perfect for time-series, messaging and wide tables with billions of rows. HBase runs on HDFS with strong consistency; Cassandra offers masterless AP design with tunable consistency. Graph databases (Neo4j) store nodes and relationships natively, making friend-of-friend, fraud-ring and recommendation queries fast via graph traversals.

- Row key design is the most important HBase/Cassandra skill - it determines distribution and query speed.
- Graph use cases: social networks, knowledge graphs, network/IT topology, fraud detection.
- Time-series data fits column-family stores naturally (row key = metric + timestamp).

Lecture 16: Apache Hive

Key terms: Hive, HiveQL, data warehouse, metastore

Apache Hive provides a SQL-like interface (HiveQL) over Hadoop: analysts write familiar queries, which Hive compiles into MapReduce/Tez/Spark jobs. It is a data-warehouse layer - best for batch analytics on large structured datasets, not low-latency queries. The metastore stores table schemas; data itself lives in HDFS files (managed tables) or external locations

(external tables).

- Partitioning (e.g. by date) and bucketing prune data scanned, speeding queries hugely.
- Managed vs external tables: dropping a managed table deletes data; external only drops metadata.
- ORC/Parquet columnar formats compress well and accelerate analytic queries.

EXAM POINTER: Distinguish managed and external tables and explain how partitioning improves Hive query performance.

Lecture 17: Apache Pig

Key terms: Pig, Pig Latin, ETL

Apache Pig is a scripting platform for ETL (extract, transform, load) pipelines: Pig Latin scripts describe data flows step by step (LOAD, FILTER, GROUP, JOIN, STORE), which Pig compiles into MapReduce jobs. It suits data engineers who think in pipelines rather than SQL, handling messy, semi-structured data with schema flexibility.

- Pig Latin is procedural (how) while HiveQL is declarative (what).
- UDFs (user-defined functions) extend Pig in Java/Python for custom logic.
- Today Spark has largely replaced Pig, but the ETL concepts transfer directly.

Lecture 18: Apache HBase

Key terms: HBase, Bigtable, column-oriented, row key

Apache HBase is a distributed, column-oriented database modelled on Google's Bigtable, running on HDFS. It provides random real-time reads/writes on billions of rows with strong consistency, automatic sharding (regions) and failover. Tables have column families defined at creation; cells are versioned by timestamp. It backs use cases like Facebook messaging and real-time counters.

- Architecture: HMaster coordinates; RegionServers serve regions; ZooKeeper handles coordination.
- Row key design rules: keep keys short, distribute writes (avoid hotspotting with salting/hashing), design for query patterns.
- Compactions merge store files; Bloom filters skip irrelevant files on reads.

Lecture 19: Data Ingestion: Sqoop and Flume

Key terms: Sqoop, Flume, data ingestion, ETL vs ELT

Getting data into Hadoop is called ingestion. Sqoop imports structured data from relational databases into HDFS/Hive (and exports back): it parallelises transfer with multiple mappers, supports incremental imports (--check-column/--last-value) and direct Hive table creation. Flume streams semi-structured log/event data: sources collect, channels buffer, sinks deliver to HDFS - a classic pipeline for web logs and sensor feeds.

- Sqoop import syntax essentials: --connect, --table/--query, --target-dir, --split-by, -m (mappers).
- Flume reliability: file channels survive agent crashes; memory channels are faster but volatile.
- Modern stacks often use Kafka instead of Flume for streaming ingestion.

EXAM POINTER: The SlideShare CS441 exam deck emphasises Sqoop import arguments - memorise --table, --columns, --where and --target-dir.

Lecture 20: Apache Kafka and Stream Processing

Key terms: Kafka, topics, partitions, producers, consumers, stream processing

Apache Kafka is a distributed streaming platform: producers publish records to topics, which are split into partitions for parallelism; consumers (in consumer groups) read them, each partition consumed by exactly one group member. Kafka durably logs streams, letting systems replay history - the backbone of real-time pipelines feeding Spark/Flink or data lakes.

- Use cases: activity tracking, log aggregation, IoT telemetry, event sourcing, messaging.
- Stream processing frameworks: Spark Streaming (micro-batch), Flink and Storm (per-event).
- Windowing (tumbling/sliding/session) aggregates streams over time - e.g. ad clicks per minute.

Lecture 21: Data Warehouses vs Data Lakes

Key terms: data warehouse, data lake, schema-on-read, ELT

A data warehouse stores cleaned, structured data with schema-on-write for BI reporting (classic vendors: Teradata, now cloud: BigQuery, Snowflake, Redshift). A data lake stores raw data of all types cheaply with schema-on-read, letting data scientists explore freely (HDFS, S3, Azure Data Lake). Modern 'lakehouse' architectures combine both: cheap object storage with warehouse-grade management (Delta Lake, Iceberg, Hudi).

- ETL (transform before load) suits warehouses; ELT (load raw, transform later) suits lakes.
- Governance risk: an unmanaged lake becomes a 'data swamp' - catalogue and quality controls matter.

- Medallion architecture: bronze (raw) -> silver (cleaned) -> gold (aggregated) layers.

Lecture 22: Big Data Analytics: Descriptive to Prescriptive

Key terms: descriptive, diagnostic, predictive, prescriptive analytics

Analytics maturity has four levels. Descriptive: what happened (dashboards, reports). Diagnostic: why it happened (drill-down, correlations). Predictive: what will happen (regression, classification, forecasting). Prescriptive: what should we do (optimisation, recommendation engines, A/B-tested actions). Big-data platforms power the upper levels by training models on massive historical data.

- Example: telecom churn - descriptive (churn rate 8%), diagnostic (poor coverage areas), predictive (customers likely to churn), prescriptive (targeted retention offers).
- Machine learning on big data: Spark MLlib scales training across clusters.
- Feature engineering often matters more than the algorithm choice.

Lecture 23: Data Visualisation for Big Data

Key terms: visualisation, dashboards, aggregation, sampling

Humans cannot read billions of rows - visualisation aggregates big data into charts, maps and dashboards that reveal patterns. Principles: choose the right chart (trends: line; comparison: bar; distribution: histogram), aggregate before plotting (plotting raw billions is impossible), and design dashboards around decisions, not data dumps. Tools range from Python (Matplotlib, Plotly) to BI platforms (Power BI, Tableau) connected to big-data backends.

- Pre-aggregation in Hive/Spark, then visualise the summary - the standard big-data viz workflow.
- Geospatial visualisation suits IoT, logistics and telecom data.
- Misleading visuals (truncated axes, 3D distortion) are an ethics issue in analytics.

Lecture 24: Big Data in the Cloud

Key terms: cloud big data, EMR, Dataproc, separation of storage and compute

Cloud platforms (AWS EMR, Google Dataproc, Azure HDInsight) run Hadoop/Spark clusters elastically: spin up hundreds of nodes for a job, then shut them down - paying only for use. Object storage (S3, GCS) decouples storage from compute, so data persists cheaply while clusters are transient. Managed services (BigQuery, serverless Spark) remove cluster administration entirely.

- Advantages: elasticity, no hardware capex, managed upgrades and security.
- Watch-outs: data-egress costs, vendor lock-in, and governance across regions.

- Hybrid patterns keep sensitive data on-premise while bursting compute to cloud.

Lecture 25: Privacy, Security and Ethics in Big Data

Key terms: data privacy, anonymisation, GDPR, data governance

Big data magnifies privacy risks: combining 'anonymised' datasets can re-identify individuals (the Netflix Prize lesson). Techniques: anonymisation, pseudonymisation, differential privacy (adding calibrated noise), and strict access control with encryption. Regulations (GDPR in Europe; PECA and data-protection developments in Pakistan) require consent, purpose limitation and breach notification. Ethical analytics avoids biased models and transparently discloses automated decisions.

- Security layers: Kerberos authentication, Ranger/Sentry authorisation, wire encryption, audit logs.
- Bias: models trained on historical data can entrench discrimination - audit outcomes by group.
- Data governance: catalogues, lineage and quality rules keep big data trustworthy.

EXAM POINTER: Discuss one privacy technique and one ethical risk of big-data analytics - a strong final-term essay question.

Lecture 26: Big Data Applications and Case Studies

Key terms: use cases: recommendation, fraud detection, IoT, smart cities

Applications span industries. E-commerce: recommendation engines and dynamic pricing (Amazon). Finance: real-time fraud detection on transaction streams. Telecom: churn prediction and network optimisation. Healthcare: outbreak surveillance and medical imaging analytics. Smart cities/IoT: traffic and energy optimisation from sensor streams. In Pakistan, banks use big data for credit scoring, and NADRA-scale identity data shows public-sector potential.

- Common pattern: ingest streams (Kafka) -> process (Spark) -> serve (dashboard/alerts).
- Success factors: clear business question, quality data pipelines, and measurable outcomes.
- Start small: a pilot on one use case beats a giant platform-first programme.

Lecture 27: Revision: Midterm Practice Set

1. Define big data and explain the 5 Vs with one example each. (10 marks)
2. Draw and explain the HDFS architecture, including the roles of NameNode and DataNodes. (10 marks)

3. Trace the word-count program through the MapReduce phases. (10 marks)
4. Compare vertical and horizontal scaling; explain why big data needs the latter. (5 marks)
5. State the CAP theorem and give one CP and one AP example. (5 marks)

Lecture 28: Revision: Final-Term Practice Set

1. Explain Apache Spark's RDD and lineage-based fault tolerance, and why Spark beats MapReduce on iterative jobs. (10 marks)
2. Compare the four NoSQL data models with one example database and one use case each. (10 marks)
3. Distinguish data warehouses from data lakes; describe the medallion architecture. (10 marks)
4. Write the Sqoop import command structure and explain --table, --columns, --where and --target-dir. (5 marks)
5. Describe a Kafka-based real-time fraud-detection pipeline end to end. (10 marks)

End of Handout - Good luck with your exams!