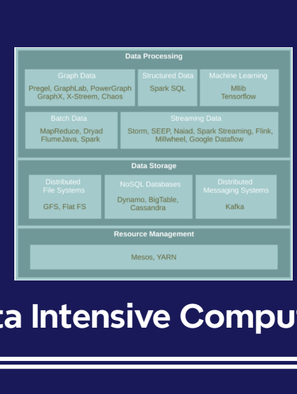

Data Intensive Computing

- Data Storage
  - Distributed File Systems(GFS)
    - Files are split into Chunks
    - Master, Chunk Servers, Client
    - Read
    - Write
    - Metadata Management
      - lookup table mapping pathnames to metadata
      - read lock on internal nodes, read/write lock on leaf
    - Replica Placement
    - Garbage Collection: timestamp
  - NoSQL Database: easy and frequent changes
    - CAP
    - Data Model
      - key/value
      - Column-oriented: value has multople attributes
      - Document
      - Graph
    - BigTable
      - Column-oriented
      - master, tablet server, client library
        - Tablet Serving
        - Finding a tablet
      - GFS, SSTable, Chubby
      - CP
        - If a tablet server fails, its portion of data is temporarily unavailable until a new server is assigned.
        - Consistency: GFS
    - Cassandra
      - Column-oriented
      - Consistency hashing: Hash both data and node ids using the same hash function in a same id space
        - Replication
      - Gossip-based mechanism: eventual consistency
      - AP
    - Neo4j
      - Graph
      - Cypher
      - CA
- Data Processing
  - Parallel Processing
    - MapReduce: expensive and slow always go to disk and HDFS
      - Programming Model
        - Mapper
        - Reducer
          - Shuffle and Sort
      - Execution Framework
        - Local Aggregation: Merge partially data before it is sent over the network to the reducer
        - Joining
          - Reduce-side: two or more large datasets
          - Map-side: one of the datasets is small enough to cache
        - Sorting: For multiple Reducers we need to choose a partitioning function key1 < key2 ⇒ partition(key1) ≤ partition(key2)
        - The buffered pairs are periodically written to local disk. They are partitioned into R regions (hash(key) mod R).
        - Failure Tolerance
          - Worker: periodic heartbeats Re-execute
          - Master: State is periodically checkpointed
    - Spark
      - RDD
        - Immutable collections
        - Partitions of an RDD can be stored on different nodes of a cluster.
      - RDD operations
        - Transformations: build the logical plan
        - Actions: trigger the computation
      - Lineage graph: logical transformation plan
        - No replication
        - Fault tolerance
        - Recompute only the lost partitions of an RDD
      - Caching: memory
        - checkpoint saves an RDD to disk.
        - Checkpointed data is not removed after SparkContext is destroyed.
      - RDD dependencies encode when data must move across network
        - pipeline
        - shuffle
      - Job: one action
      - Stage: groups of tasks that can be executed together
      - Task: All of the tasks in one stage execute the same code on a different piece of the data
  - Structured Data Processing(Spark SQL)
    - RDD vs. DataFrame vs. DataSet
      - RDD: don't know anything about the schema
      - DataFrame: Equivalent to a table in a relational database
        - Transformation: select, selectExpr, filter, where, distinct, withColumn, withColumnRenamed, drop, udf
        - Action: collect, count, first, head, show, take
        - DataFrames elements are Rows, which are generic untyped JVM objects
        - Scala compiler cannot type check Spark SQL schemas in DataFrames
      - DataSet: typed distributed collections of data
        - type DataFrame = Dataset[Row]
    - Aggregation
      - summarize: count, countDistinct, first, last, min, max, sum, avg
      - group by
      - windowing
    - Join
      - Shuffle join: big table to big table Every node talks to every other node
      - Broadcast join: big table to small table fit into the memory of a single worker node
    - SQL queries: on views/tables via the method sql on the SparkSession object
  - Stream Data Processing
    - Data stream, unbounded data, tuples
    - SPS vs. DBMS
      - DBMS: Store and index data before processing it. Process data only when explicitly asked by the users.
      - SPS: Processing information as it flows, without storing them persistently
    - Event-time vs. processing time
      - Triggering determines when in processing time the results of groupings are emitted as panes.
        - time-based triggering
        - data-driven triggering
      - Windowing determines where in event time data are grouped together for processing.
        - time-based windowing
          - watermark
        - Fixed window, trigger at period (micro-batch)
        - Fixed window, trigger at watermark (streaming)
    - Messaging system and partitioned logs
      - Message broker: Decoupling producers and consumers
      - In typical message brokers, once a message is consumed, it is deleted.
      - Log-based message brokers durably store all events in a sequential log
    - Kafka: distributed, topic oriented, partitioned, replicated log service
    - Logs, topics, partition
      - Topics are logical collections of partitions (the physical files)
        - Ordered
        - Append only
        - Immutable
      - One broker is the leader of a partition
    - Kafka architecture: producer, consumer, broker, coordinator
      - Zookeeper: Keeping track of the consumed offset of each partition.
      - Brokers are sateless: no metadata for consumers-producers in brokers.
      - Consumers are responsible for keeping track of offsets
      - Messages from a single partition are delivered to a consumer in order
      - Guarantees at-least-once delivery
    - Mini-batch processing: Run a streaming computation as a series of very small, deterministic batch jobs
    - DStream: sequence of RDDs
    - RDD and window operations
      - tumbling window: slide interval = window length
      - Spark supports stateful streams.
    - Structured streaming
  - Graph Data Processing
    - PageRank
      - Problem with MapReduce
        - MapReduce does not directly support iterative algorithms.
        - Materializations of intermediate results at every MapReduce iteration harm performance.
    - vertex: individually computes value, depend on neighbors
      - Pregel: BSP, synchronous parallel model, message passing
      - GraphLab: asynchronous model, shared memory, GAS
        - Gather: accumulate information from neighborhood.
        - Apply: apply the accumulated value to center vertex.
        - Scatter: update adjacent edges and vertices.
    - table
      - Graphx: unifies data-parallel and graph-parallel systems.
- Resource Management
  - Monolithic scheduler: centralized scheduling algorithm for all jobs
    - Optimal Schedule
    - Complex, hard to anticipate future frameworks, refactor for existing frameworks
    - Borg
      - Request-based
      - Cell, Job, Task, and Alloc
      - BorgMaster, Borglet
      - Container orchestration tools: Kubernetes (based on Borg), Marathon (runs on Mesos)
  - Two-Level scheduler: separate resource allocation and task placement. An active resource manager offers compute resources to multiple parallel, independent scheduler frameworks.
    - Simple, easy to port
    - Not optimal
    - Mesos
      - Framework->jobs->tasks(allocation)
      - Offer-based
        - Master sends resource offers to frameworks.
        - Frameworks select which offers to accept and which tasks to run.
      - Max-Min fairness
        - Share guarantee
          - Each user can get at least 1/n of the resource.
          - But will get less if her demand is less.
        - Strategy proof
          - Users are not better off by asking for more than they need.
          - Users have no reason to lie.
      - multiple resources Problem: violates share grantee
      - DRF
    - YARN
      - Request-based
      - Resource Manager
        - One per cluster (Central: global view)
        - Job requests are submitted to RM. To start a job, RM finds a container to spawn AM.
        - Only handles an overall resource profile for each job. Local optimization is up to the job
      - Application Manager
        - The head of a job.
        - Runs as a container.
        - Request resources from RM
      - Node Manager
        - The worker daemon.
        - Registers with RM.
        - One per node.
        - Report resources to RM

- DataLake
  - Data Warehouses
    - ETL
    - ACID
    - not support unstructured or semi-structured
    - not support data science and ML
  - Data Lakes
    - all raw data
    - ETL to warehouse
    - directly readable in ML
    - complex architecture
    - data reliability suffers
    - Timeliness suffer: Extra ETL steps before data is available in data warehouses
  - Data Lake stores all data irrespective of the source and its structure whereas Data Warehouse stores data in quantitative metrics with their attributes.
  - Data Lake defines the schema after data is stored whereas Data Warehouse defines the schema before data is stored.
  - Data Lake uses the ELT process while the Data Warehouse uses ETL process.
  - Lakehouse
    - Metadata layers for Data Lakes
      - DeltaLog
      - Provides ACID transactions.
      - Provides scalable metadata handling.
      - Provides time travel and versioning.
      - Unifies streaming and batch data processing.
    - New query engine designs
    - Declarative access for data science and ML