10. Batch Processing

- Batch Processing with Unix Tools
  - Simple Log Analysis
    - Unix commands: awk, sed, grep, sort, uniq, and xargs
    - Chain of commands versus custom program
    - Sorting versus in-memory aggregation
      - For large datasets - unix commands can be better - as sort will never exceed memory limits - it can sort on disk if needed
  - The Unix Philosophy
    - Pipes
    - A uniform interface
      - One tool can be in pipe with another tool
    - Separation of logic and wiring
      - good design!
      - But hard with multiple inputs or outputs
    - Transparency and experimentation
      - Immutable input
      - But can run only on single machine
- MapReduce and Distributed Filesystems
  - HDFS (Hadoop Distributed File System)
    - an open source reimplementation of the Google File System (GFS)
    - shared-nothing principle
  - MapReduce Job Execution
    - Mapper
      - For each input record extracts key and value
    - Reducer
      - Collects all values belonging to the same key and calls reducer with operator over collection of values
    - Distributed execution of MapReduce
      - Scheduler tries to put computation near data
      - Number of mappers = number of input files
      - Number of reducers is configured by author
        - And each mapper produces sorted files for each reducer using hash functions
    - MapReduce workflows
      - MapReduce has limited functionality => need workflow of few MapReduces
        - Such workflow is implemented using workflow schedulers, such as
          - Oozie, Azkaban, Luigi, Airflow, and Pinball
        - There are higher-level tools for Hadoop, such as
          - Pig, Hive, Cascading, Crunch, and FlumeJava
  - Reduce-Side Joins and Grouping
    - Example: analysis of user activity events
    - 1 Sort-merge joins
      - User's day of birth will go to reduced before all user's click data
    - Bringing related data together in the same place
    - GROUP BY
    - Handling skew
      - Most of people has some followers but celebs has millions of followers
        - linchpin objects or hot keys
      - All celeb's data to single reducer is not efficient
        - skew
          - Aka hot spots
      - Such data is distributed across multiple reducers
        - The cost is we should also distribute the join records for that keys
  - Map-Side Joins
    - 2 Broadcast hash joins
      - aka replicated join or MapJoin
      - Often requested join data is in cache => fast enough
    - 3 Partitioned hash joins
      - aka bucketed map joins
      - Data that should be joined is partitioned and distributed to Mappers
    - Map-side merge joins
      - If data is pre-sorted then mapper can join efficiently without sorting or accessing by hash index
    - MapReduce workflows with map-side joins
  - The Output of Batch Workflows
    - Building search indexes
    - Key-value stores as batch process output
      - Example - some ML calculations
    - Philosophy of batch process outputs
      - Similar to Unix commands!
        - But data is more structured - no need to parse
        - And no need to run sort between map and reduce like in Unix
  - Comparing Hadoop to Distributed Databases
    - Diversity of storage
      - Bringing data together faster is better that planning structure ahead - so - data on HDFS can be unstructured in the beginning of experiments
      - So Hadoop is used for ETL
        - sushi principle
          - raw data is better
    - Diversity of processing models
    - Designing for frequent faults
      - MPP reruns the whole query
      - MapReduce reruns the task on failed node
        - Better for LARGE jobs
      - Overcommitting resources at Google datacenter
        - More efficient resource utilisation
        - But some low priority tasks will be killed
          - So - need to tolerate kills / restarts
- Beyond MapReduce
  - Problems with MapReduce
    - It is not simple
      - You need to implement all the joins
      - However higher-level programming models (Pig, Hive, Cascading, Crunch) make things easier
    - It is not fast for some kind of processing
      - Other tools can be used here
  - Materialization of Intermediate State
    - Problems with MapReduce
      - Job can start when all tasks in previous job finished
      - Mappers are redundant - previous Reducer is enough
      - Replicating intermediate files is overlill
    - Dataflow engines
      - Spark, Tez, Flink
      - Handle entire workflow as job
        - More optimisations => faster than MapReduce!
      - Typically arrange the operators in a job as a directed acyclic graph (DAG)
        - flow of data from one operator to another is structured as a graph
        - the data itself typically consists of relational-style tuples
    - Fault tolerance
      - Computations should be deterministic!
    - Discussion of materialization
  - Graphs and Iterative Processing
    - the data itself has the form of a graph
      - Different from Dataflow engines
    - Repeat until done (iterative style algo)
      - Cannot be implemented in MapReduce
      - Graphs Processing is good for this
    - The Pregel processing model
      - bulk synchronous parallel (BSP)
      - Unlike in MapReduce, a vertex remembers the state
      - Implemented by
        - Apache Giraph
        - Spark's GraphX API
        - Flink's Gelly API
    - Fault tolerance
    - Parallel execution
  - High-Level APIs and Languages
    - The move toward declarative query languages
      - To reduce function calls and data fields where possible
        - e.g. select only needed columns
    - Specialization for different domains
      - Mahout
        - For ML on top of MapReduce, Spark, and Flink
      - MADlib
        - Similar functionality inside relational MPP database (Apache HAWQ)
- Summary
  - How batch-process data evolved
    - Unix commands
    - MapReduce
    - Data Flow engines
  - Two main problems
    - Partitioning
      - To bring all related data in the same place
    - Fault Tolerance
      - To recoved from individual failed tasks
  - Join algos
    - Sort-merge joins
    - Broadcast hash joins
    - Partitioned hash joins
  - Batch processing works well when the input data is bounded (fixed size)!
    - For unbounded data use Stream Processing (see chapter 11)