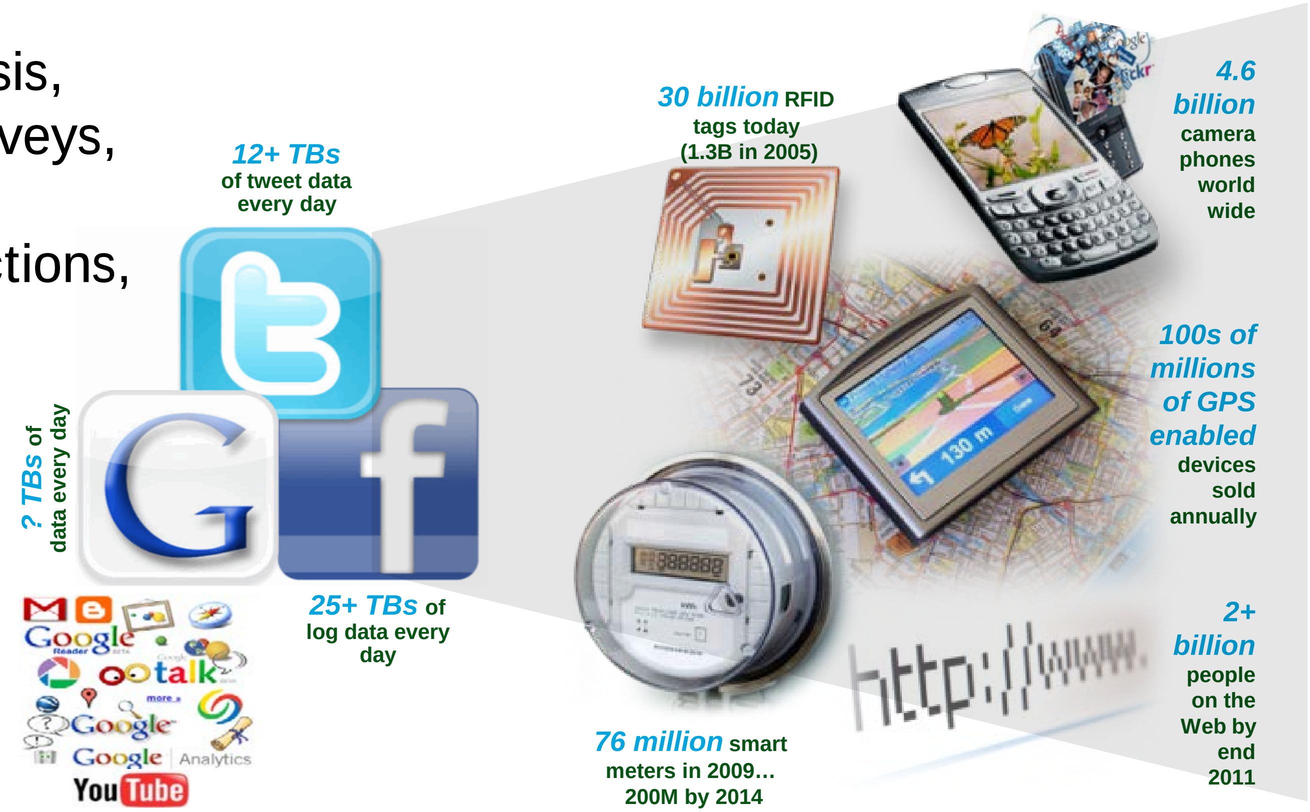
OLAP Queries on Big Data Processing Systems

Ulf Leser, Humboldt-Universität zu Berlin

Big Data

- What to do when data sets get really big?

  Web crawling,
  click-stream analysis,
  astronomy sky surveys,
  cellphone calls,
  credit card transactions,
  sensor readouts,
  ...



- Map/Reduce and Hadoop
- Big Data Processing Systems
- Example: HIVE

Two Options

- Buy supercomputers
  - Very fast networks, 10000+ cores, high-quality hardware
  - Very expensive, outdated quickly, cooling is an issue
- Buy lots of commodity hardware
  - Normal networks (10GB), multiple cores in 1000+ machines, cheap hardware with none-trivial probability of failures
  - Comparably cheap, renewal possible, less cooling issues
  - Difficult to program: Achieving high throughput on distributed machines with regular failures

The Advent of MapReduce

- Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. CACM, 51(1)
  - First paper in 2004; 23000+ citations today
- Main ideas
  - Focus on typical data analysis requirements (~OLAP)
    - No synchronization, no transactions, no multi-user, no time-critical operations, ... all the things which are difficult in distributed systems
  - Accept failing machine and single-point-of-failures
    - 1000+ cheap workers which may fail,
    - One more robust coordinator node which should not fail
  - Separate data analysis and file management
    - Use a distributed file system for data exchange
  - Wrap everything in MAP or REDUCE second-order-functions

Infrastructure

- Types of distributed analysis have many task in common
  - Manage cluster: IP, capacity, port number, credentials, ...
  - Start / stop / monitor tasks on worker nodes
  - Restart nodes in case of failure
  - Manage files and provide access to data (in a fail-safe manner)
  - Login, logging, administrative interfaces
  - Scheduling: Which task should start when on which node?
  - All these are provided by the MapReduce infrastructure
    - Open source: Hadoop
- Things that are not common
  - Perform the analysis (the first-order functions)
  - Build local environment to run first-order functions (libs, ...)
  - These must be provided by the developer (and nothing else)

Map / Reduce

- Second-order function g: Function with two parameters
  - Set D of data elements
  - Function f
  - General semantics: Apply f to all elements of D independently
- Like a loop through D, but with assertion that computation for an element d is independent of all other elements of D
- Map: f(d) must produce 0-n pairs <k,d’>
  - d may be filtered or produce multiple outputs
  - k is a (non-unique) key, d’ some payload derived from d
- Reduce: Group-by k and apply f on each group
  - f must be an aggregation function

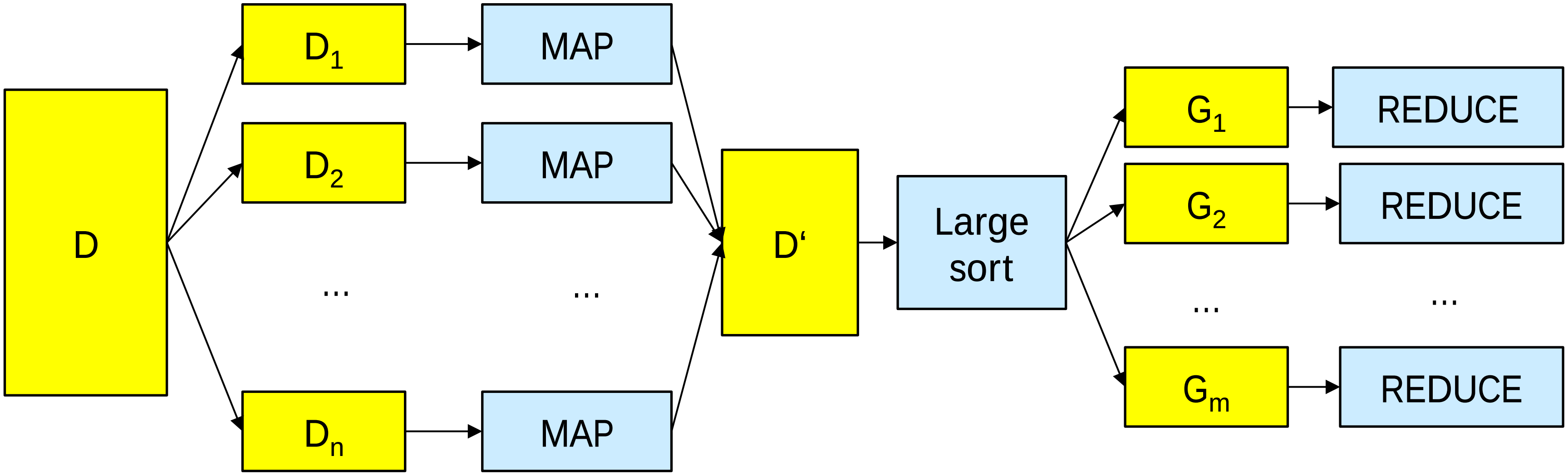
Example: Simple GROUP-BY Query

```
SELECT    year_id, sum(amount*price)
FROM      sales S
WHERE     shop_ID = 10 AND price>10
GROUP BY  year_id;
```

- Input set D: All tuples from S
- MAP(D, WHERE)
  - Read each tuple, check WHERE condition, write nothing if condition is not met and <year_id, amount*price> otherwise
- REDUCE(…, SUM)
  - Read all <k,d‘> output by MAP, group by year_id, call SUM for each group, and output <year_id, sum>

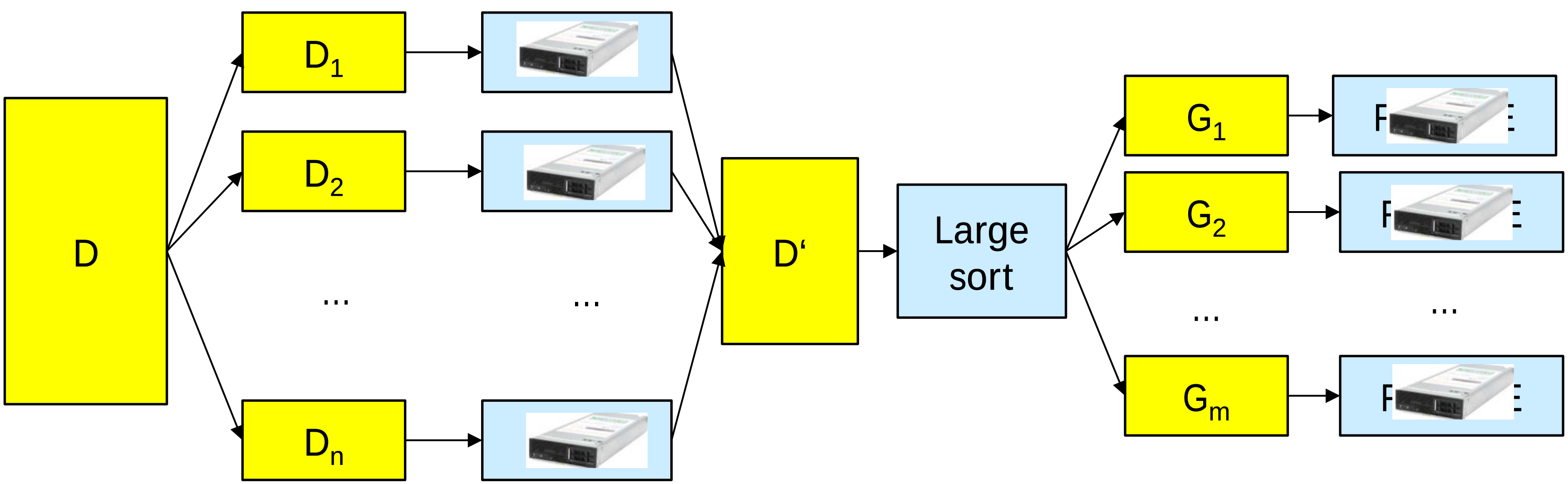
Distributed Processing

- Trick: We can very easily parallelize MAPs and REDUCEs
  - All tuples in MAP are treated independent – partition D equally between all available nodes
  - All groups in REDUCE can be treated independently – partition all groups equally between all nodes
- Works perfectly – centers with 10000+ nodes are known

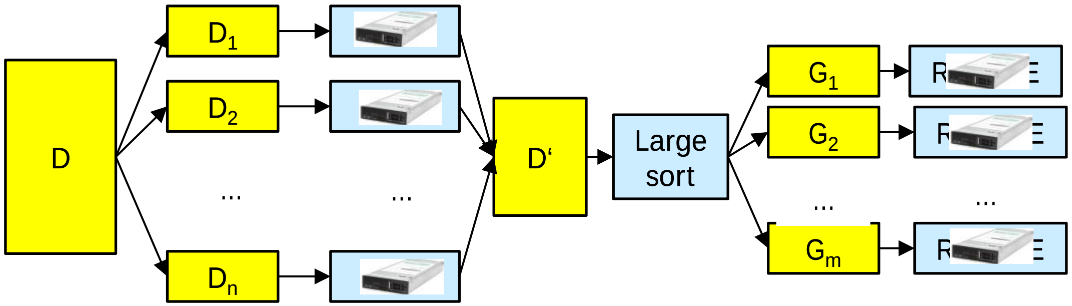


Distributed Processing

- Trick: We can very easily parallelize MAPs and REDUCEs
  - All tuples in MAP are treated independent – partition D equally between all available nodes
  - All groups in REDUCE can be treated independently – partition all groups equally between all nodes
- Works perfectly – centers with 10000+ nodes are known

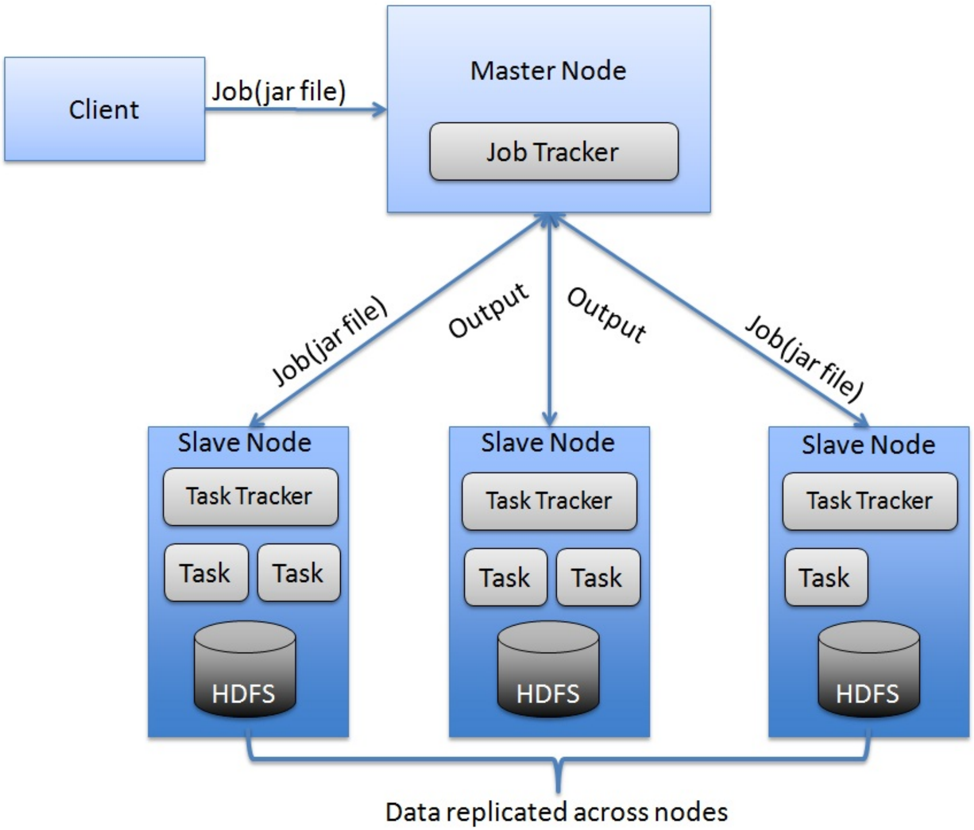


Systems Aspects



- Partitioning (splitting) can be done by system or custom
  - System: Only if records are distinguishable
- Functions for MAP and REDUCE are provided by developer
- Load-balancing
  - Produce many more partitions than nodes
    - Struggler: A partition taking much longer than others (e.g. non-linear runtime in size of d)
  - MAP: Simple, assume linear runtime in number of records
  - REDUCE: Tricky – # of groups is unknown, groups have dif. sizes
- Assignment of partitions/groups to node is performed by a central instance – Master node (scheduler)

Hadoop and HDFS

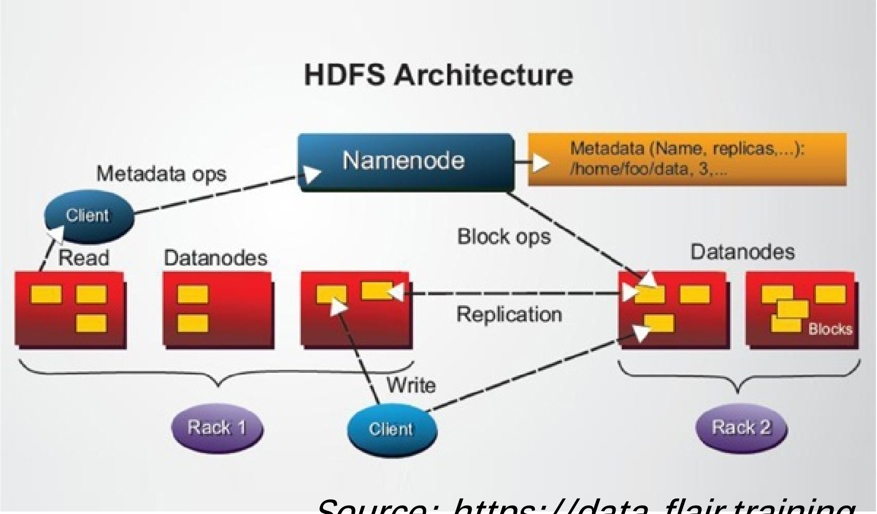


Source: http://blog.raremile.com/

- System by Google was never made public
- Hadoop: Yahoo / open source implementation of the MapReduce idea
  - Apache Top-Level project
- HDFS: Hadoop Distributed File System
- Hadoop 2 (Yarn): Arbitrary tasks & execution orders
- Hadoop 3: Erasure coding in HDFS

HDFS

- Batch processing: Tasks read/write data into/from HDFS
- HDFS Architecture
  - Files are split into chunks
    - e.g. 64MB
  - Chunks are replicated on multiple nodes (e.g. 3 times)
  - Client request chunk-Ids from name node
    - Trying to find „close“ chunks
  - Clients read directly (and possibly in parallel) from data nodes
  - If a data node crashes – replica survive
- Disadvantage: No POSIX interface
  - Clients must use special HDFS-API



Source: https://data-flair.training

Failure Tolerance and Scheduling in Hadoop

- Fault tolerance
  - Master node tracks worker nodes
  - Worker nodes taking „too“ long (hangs): Task is replicated
  - Worker node not responding (crash): Task is replicated
  - If master node crashes – system is dead
    - But (intermediate) data survives in HDFS
- Scheduling
  - Simple round-robin scheduling
  - Master node has queue of ready-to-run tasks
  - Worker nodes ask for tasks and report when finished
  - Ideally, tasks are assigned to workers having a local copy of the to-be-processed data
  - Reduce may only start after all Maps are ready
    - Inflexible! Key could already be used for splitting

Limitations (compared to a RDBMS)

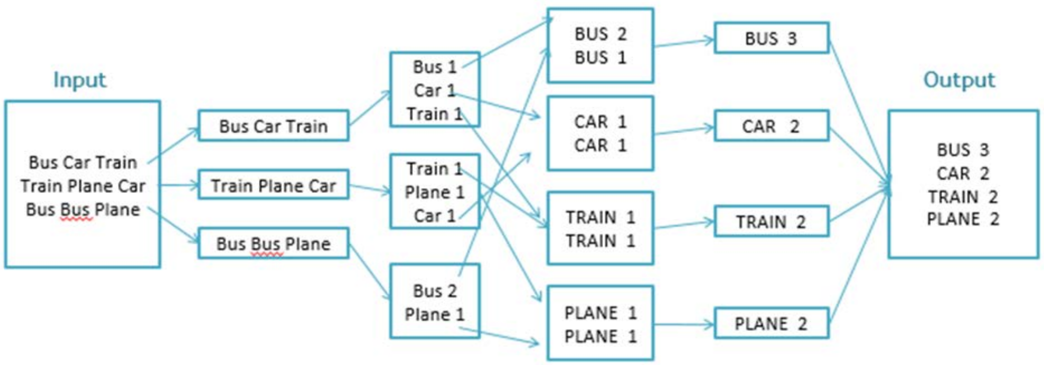
- MAP and REDUCE have only one input – what about JOIN and UNION (solved)?
- No indexing – always all the input is scanned (not solved)
- Slow data exchange – always IO+network (solved)
- No optimization of operator order (partly solved)
  - Which map should be executed first if there are multiple?
- Need to write JAVA instead of SQL (bug or feature?)
- Integration in existing systems? (solved – SQL on Hadoop)
- Data cannot be modified (not solved)
- Data is stored in verbose formats – expensive parsing (solvable)
- ...

But

- Commercial parallel DBMS are extremely expensive
- Commercial parallel DBMS do not scale
  - Consistent writes are quite difficult in distributed systems
  - Need to support distributed transactions, synchronized data access, replication strategies, failover modes, ...
- There don't exist any open source parallel databases or data processing systems

Classical Example: Word Count

- MapReduce is not SQL
- Word count: Given a very large collection of documents, report the frequency of each distinct word
  - Important step for indexing in information retrieval
- Idea
  - MAP: Take a document d as input, break into words, count frequencies, write <word, freq-word> for each distinct word
  - SHUFFLE: Sort all pairs by key <word>
  - REDUCE: Sum-up all <freq-word> per word



Source: https://dzone.com/

- Map/Reduce and Hadoop
- Big Data Processing Systems
- Example: HIVE

Hadoop for Structured Queries

- MapReduce fits perfectly to selections and group-bys
- MapReduce assumes data-parallel problems
  - An operation may be performed on single tuples without considering other tuples
  - “Embarrassingly parallel”
- Not all relational operators are data parallel
  - Only those that can be pipelined
  - Pipeline-breaker: order-by
  - Difficult to handle: Union, Join

Joins in MapReduce

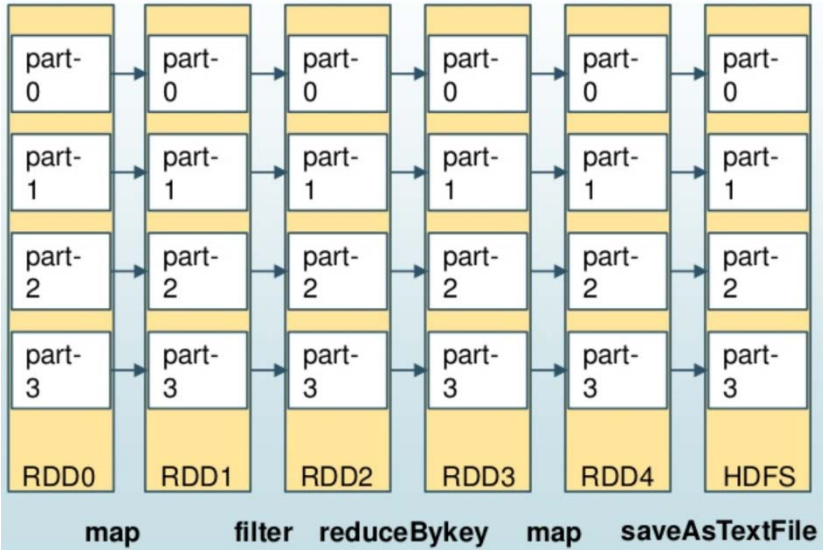
- Many suggestions
- Example: Map-side join (assume input s_1 is small)
  - MAP: Preload s_1 at once; read partition of s_2 and compute join
  - SHUFFLE, REDUCE: nothing
  - Problem: Each MAP task needs enough memory to hold s_1
- Example: Re-partition join (on join attribute k)
  - MAP: Read tuples <k,d> from source s and output <k,s,d>
    - s: 1 or 2 for the two input relations
  - SHUFFLE: Sort by <k>
  - REDUCE: Load all tuples with same key k, check if join-partner exists, and output <k,d1,d2>
  - Problem: First steps read/write all data three times

Big Data Processing Systems

- Several commercial and research systems building on MapReduce ideas but offering additional functionality
- Two classes
  - Focus on structured data and query-like analysis – white box data model, few known operators, reordering possible
  - Focus on unstructured data and arbitrary analysis – black box data and black-box operator model, no reordering
    - (Scientific) workflow systems

Example: Spark

- Main catch
  - Exchanging data through files is slow / unnecessary with todays memories
  - But if data is kept in memory, no intermediate data remains in case of a crashing node – need to restart entire job
- RDD: Resilient distributed datasets
  - Partitioned datasets become first-class objects
  - RDD are immutable: A step produces a new RDD
  - RDDs are kept in main memory and exchanged through sockets
  - System stores trace of RDD generation at partition level

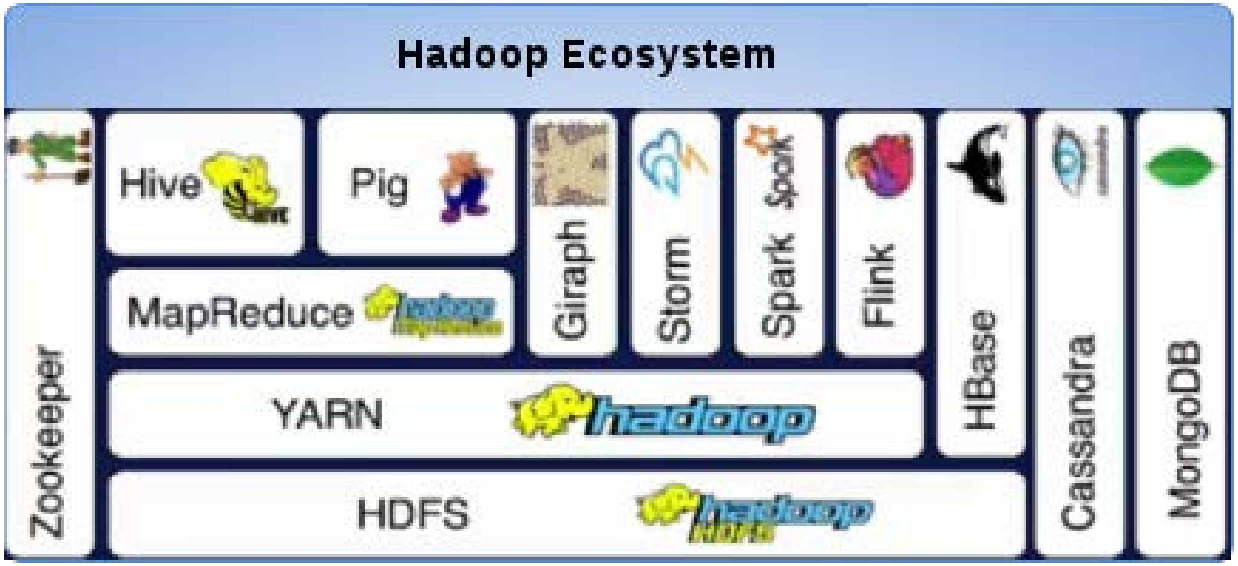


  - If a node crashes, only its current partitions need to be recreated (and the history tells us how)

Example: Stratosphere / Flink

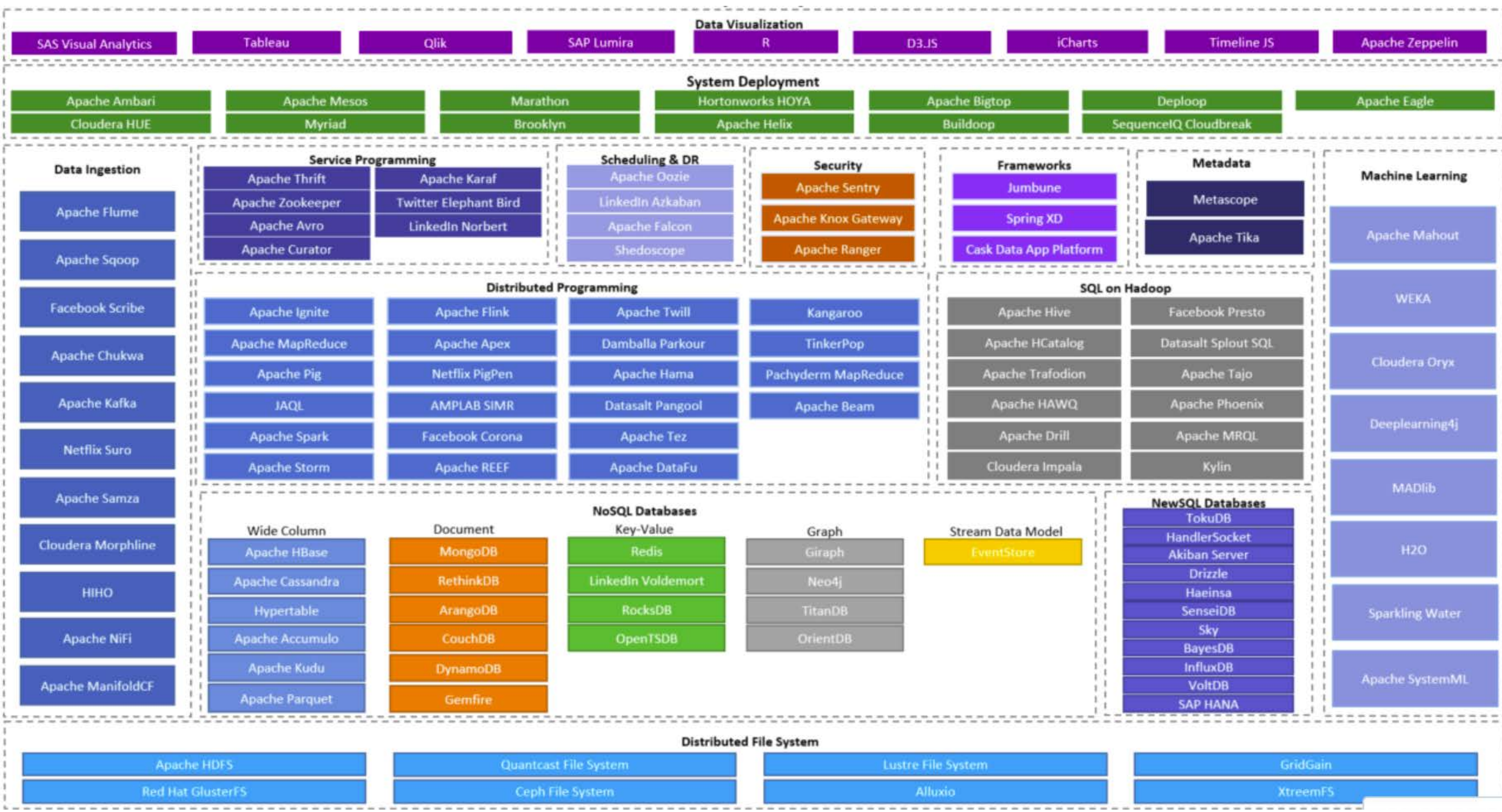
- Main catch: Only having Map and Reduce is too restricted
- More second-order functions: Map, reduce, group, co-group, union, join
- Focus on relational processing, but also support and optimization for UDFs
- Streaming: Data parallel parts of a query are executed tuple-by-tuple with exchange though sockets
  - Query can run on infinite input (stream of tuples)
  - Can produce instant answers despite changing inputs (to some degree)

Hadoop Ecosystem (small)



Source: https://www.mindtory.com/

Hadoop Ecosystem (large)



Source: https://mydataexperiments.com/

- Map/Reduce and Hadoop
- Big Data Processing Systems
- Example: HIVE

Exemplary System: HIVE

- DWH system build on top of Hadoop (Facebook)
  - Many successors: Cloudera, HortonWorks, Pentaho, ...
  - Today a popular Apache project
- Quite comprehensive (read-only) SQL support
- Focus: Optimization of batch-oriented MapReduce jobs
  - No index support
- Storage: All in files / directories in HDFS
- “At Facebook, a Hive warehouse contains tens of thousands of tables, stores over 700TB and is used for reporting and ad-hoc analyses by 200 Fb users.” (2017)

Motivation

- Fast data growth – from 15TB to 700TB in a few years
- Existing RDBMS became slower and slower and had no way to scale out to new hardware
- Only ingesting click-stream data was slower than its production
- Hadoop's API MapReduce is too low level – need for declarative data access

Storage

- Table = directory
- Partitions = subdirectories
  - Horizontal partitioning with range or equality partitioning
  - Used for partition pruning in scans
- Buckets = files
  - Hash or range partitioning
  - Used for bucket pruning during scans

- User can provide custom parsers to read special row formats

HiveQL

- Full set of primitive data types (float, string, int, ...)
- Nested collection types: Struct, sets, bags
- Subset of SQL: Select, join, aggregate, union-all, nested queries
  - No Theta-Joins
- Support for UDFs, embedded MapReduce scripts, and metadata queries
- No single-tuple insert, no delete, no update
- Table creation: Interpreting an existing file as a table
  - But SQL may create new tables = new files

Example – Word Count in Hive

Output from MAP phase

First-order UDFs

```
FROM (
  MAP doctext USING 'python  wc_mapper.py'
      AS (word, cnt)
  FROM docs
  CLUSTER BY word
) a
REDUCE word, cnt USING 'python wc_reduce.py';
```

Input table (with attribute doctext)

Sort-order for directing output-tuples

Limited Optimization

- Only rule-based (where should statistics come from?)
  - Predicate Push-Down, column pruning, partition pruning
- Joins: Broadcast smaller table to mapper for larger table
  - Map-side join
- Pre-aggregation (COMBINE phase)
- Users may provide hints
- Scheduling is completely delegated to Hadoop

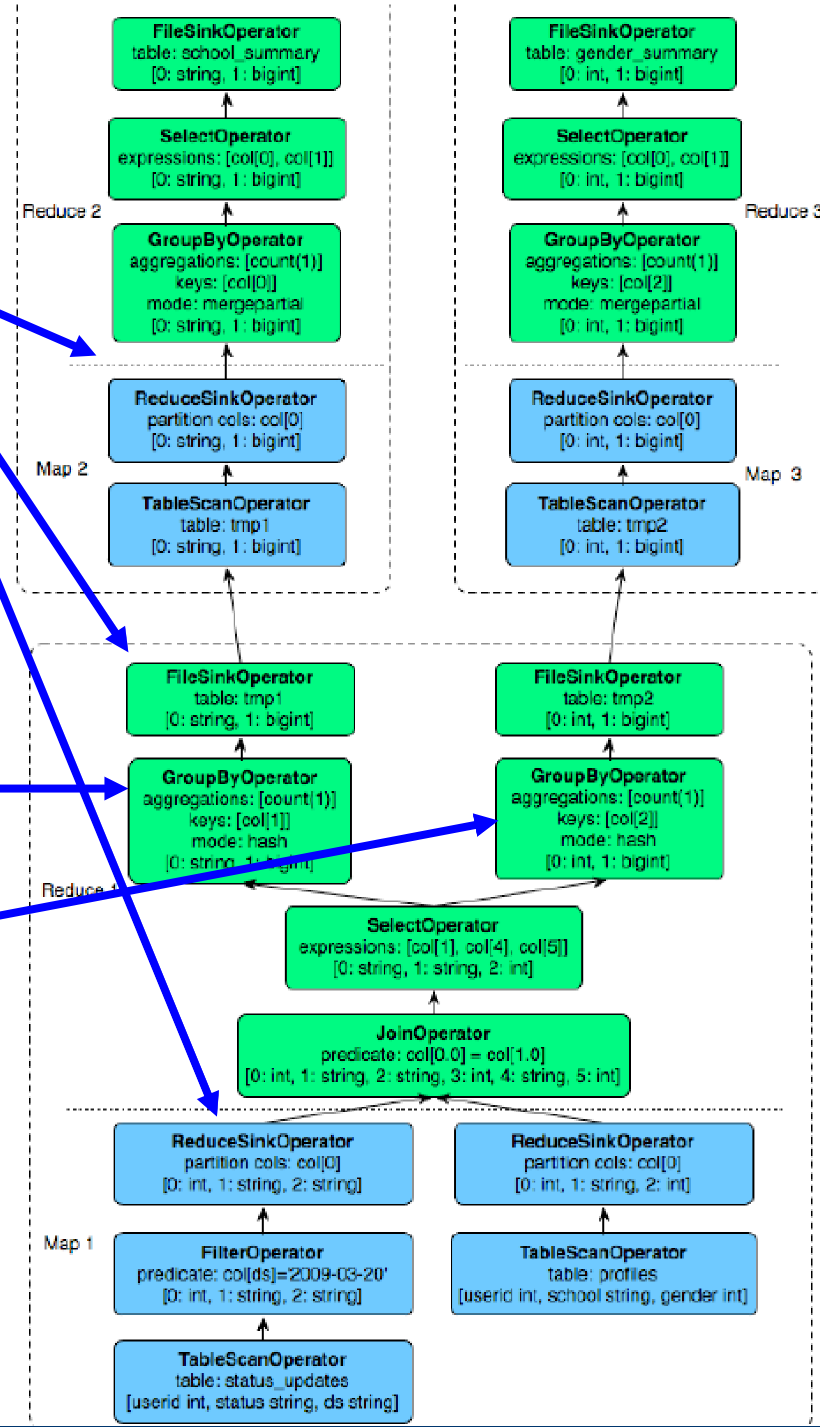
Example

Data exchange between phases through HDFS

```
FROM (SELECT a.status, b.school, b.gender
        FROM status_updates a JOIN profiles b
             ON (a.userid = b.userid
                 AND a.ds='2009-03-20' )) subq1

INSERT OVERWRITE TABLE gender_summary
                        PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1)
GROUP BY subq1.gender

INSERT OVERWRITE TABLE school_summary
                        PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1)
GROUP BY subq1.school
```



Other Systems

- HadoopDB
- Presto (Teradata)
- Cloudera Impala
- Spark SQL
- Apache Drill
- AsterixDB

References

- Dean, J. and Ghemawat, S. (2008). "MapReduce: Simplified Data Processing on Large Clusters " Communications of the ACM 51(1).
- Thusoo, A., Sarma, J., Jain, N., S, Z., Chakka, P., Z, N., Antony, S., L, H. and Murthy, R. (2010). "Hive - a petabyte scale data warehouse using Hadoop". Int. Conf. on Data Engineering, Long Beach, CA
- Alexandrov, A., Bergmann, R., Ewen, S., Freytag , J.-C., Hueske, F., Heise, A., Kao, O., Leich, M., Leser, U., Markl, V., et al. (2014). "The Stratosphere Platform for Big Data Analytics." VLDB Journal 23(6): 939-964.
- Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S. and Stoica, I. (2012). "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing". USENIX conference on Networked Systems Design and Implementation. San Jose, USA.

Self Assessment

- Describe the semantics of MAP and REDUCE functions
- Compare HDFS to a remote data access with NFS. What are pros / cons?
- What is the COMBINE phase of a mapreduce pipeline?
- Design a mapreduce program for the following problem: Given a set of market basket contents, find all pairs of items sold together more often than k times
- What is a single-point-of-failure? Where does Hadoop have spofs?