Distributed Data Processing

Gianluca Demartini

DATA7201 Data Analytics at Scale

Week 5

Last Week

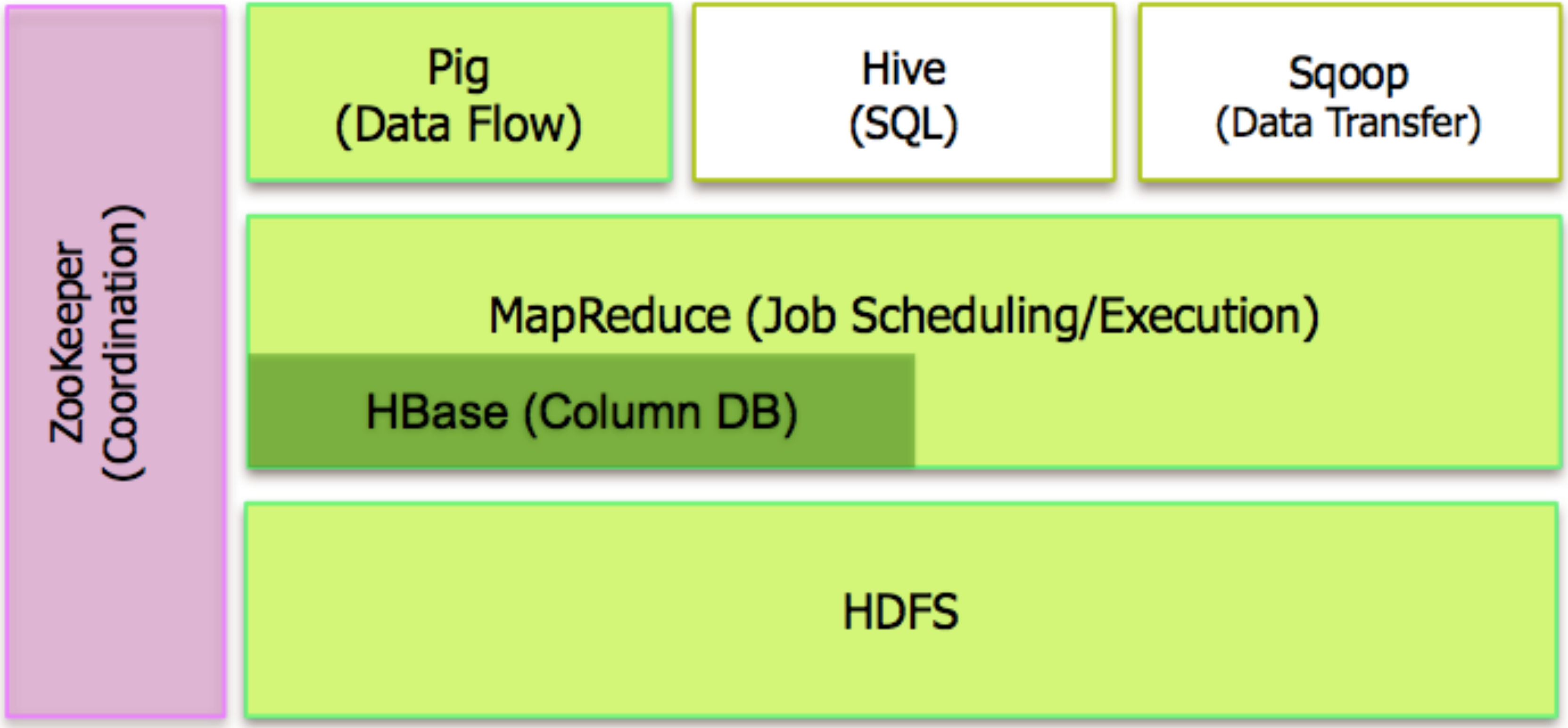
- OLAP vs OLTP
- Architectures for Distributed Databases
- Big Table
- Hbase and Hive
- PIG

Week	Date	Lecture	Prac	Assessment
1	21-Feb	Introduction to DATA7201 - Data Analytics at Scale	-	
2	28-Feb	Supporting Infrastructures and Use Cases	-	
3	6-Mar	Storage Infrastructures for Large Data Volumes	Intro to Cluster and HDFS	
4	13-Mar	Analytics Queries for Large Data Volumes	PIG(1)	
5	20-Mar	Distributed Data Processing	PIG (2)	
6	27-Mar	Processing Large Data Streams	PySpark (1)	**Quiz 1 Due (5)**
Semester Break				
7	10-Apr	Processing Large Graph Data (1) + use cases	PySpark (2)	
8	17-Apr	Processing Large Graph Data (2) + use cases	Project support	
9	24-Apr	Recommender Systems	Project support	**Quiz 2 Due (5)**
10	1-May	Opinion Mining + use cases	Project support	
11	8-May	Health Data Analytics (guest speaker)	Project support	
12	15-May	Large Language Models?	Project support	**Report Due (45)**
13	22-May	Course Revision	-	**Quiz 3 Due (5)**

Lecture Outline

- Zookeeper
- Map/Reduce jobs
- Hadoop, YARN, and Spark
- Python (PySpark), R

The Hadoop Stack



Zookeeper - Motivation

- In the past: a **single** program running on a **single** computer with a **single** CPU
- Today: applications consist of **independent** programs running on a **changing** set of computers
- Difficulty: **coordination** of those independent programs
- Developers have to deal with **coordination logic** and **application logic** at the same time

Zookeeper

- A distributed configuration service, synchronization service, and naming registry for large distributed systems.
  - High availability through redundant services
  - Scalability
  - Fast for read intensive workload

Zookeeper design

- Client requests are processed in FIFO order
- ZooKeeper service is an ensemble of servers that use replication (high availability)
- (meta)Data is cached on the client side
  - ID of datanodes, instead of probing ZooKeeper every time.
  - What if data changes?
    - Polling?
    - Watch mechanism: clients can watch for an update of a given object

Zookeeper design

- ZooKeeper data is replicated on each server that composes the service
  - Recovery from master failures
- Clients connect to exactly one server to submit requests
  - **read** requests served from the local replica
  - **write** requests are processed by an agreement protocol (an elected server leader initiates processing of the write request)

"ZooKeeper: Wait-free coordination for Internet-scale systems", Hunt et al., USENIX 2010

MapReduce (MR)

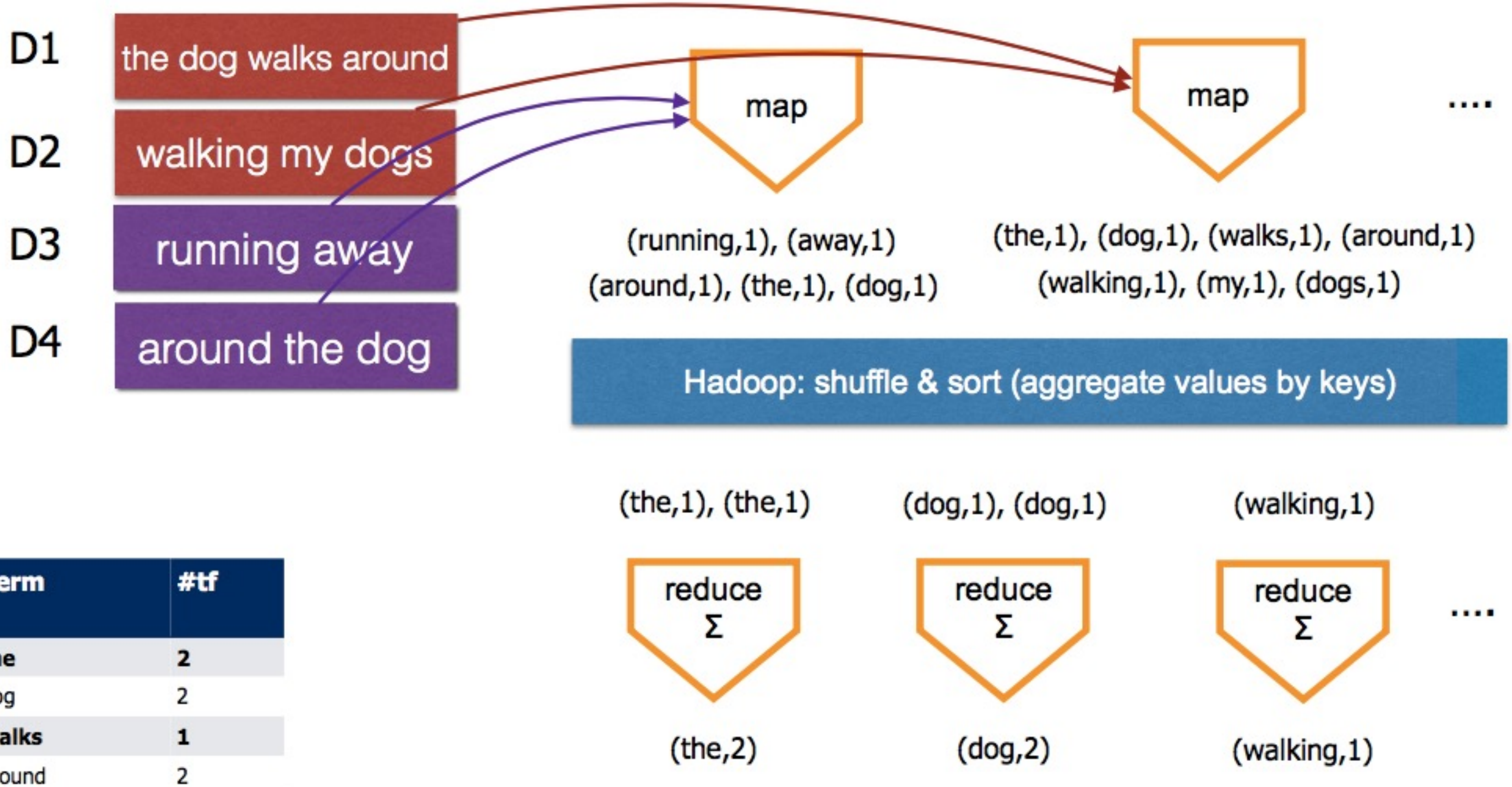
- High-level programming model and implementation for large-scale parallel data processing
- Commodity hardware
- Fault-tolerant
- **Divide & conquer**: partition a large problem into smaller sub-problems
  - **Independent sub-problems** can be executed in parallel by workers
  - Intermediate results from each worker are **combined** to get the final result

MR Data Model

- Files on HDFS!
- Each file a set of (**key,value**) **pairs**
- A map-reduce program:
  - Input: a set of (input key, value) pairs
  - Output: a set of (output key, value) pairs

k1 -> v1
k2 -> v2
k3 -> v3

MR: World count example



Term	#tf
the	**2**
dog	2
walks	**1**
around	2
walking	**1**
my	1
....	**...**

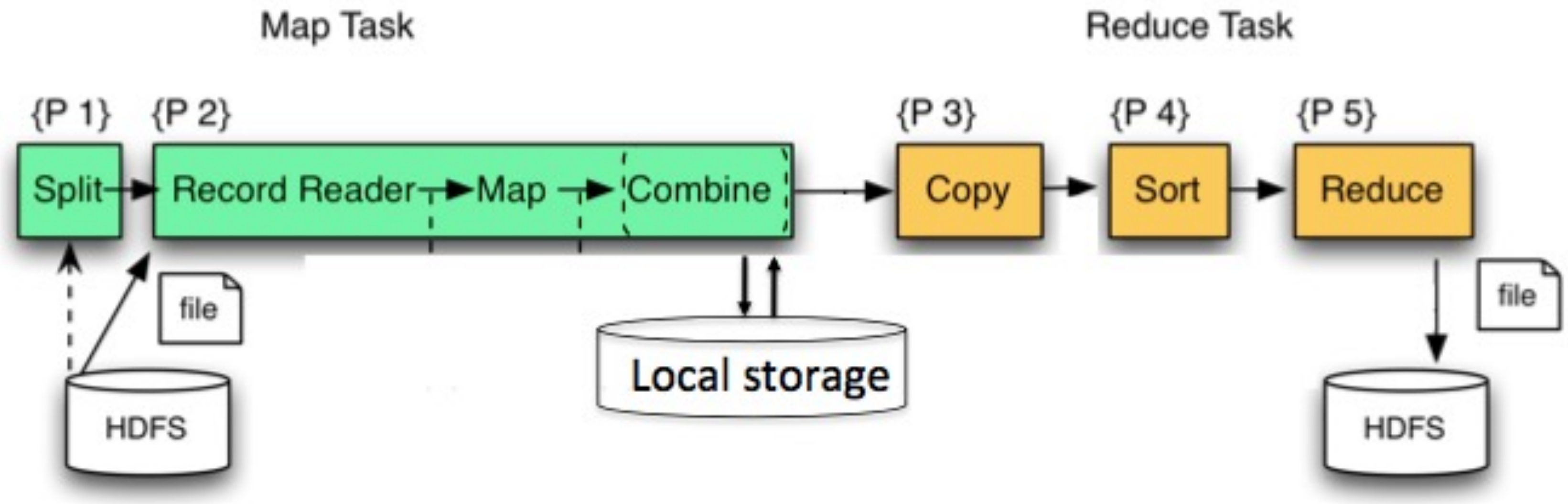
The Partitioner

- Responsible for dividing the intermediate key space and assigning intermediate key/value pairs to reducers
- Default key-to-reducer assignment:
  - hash(key) mod num_reducers

The Combiner

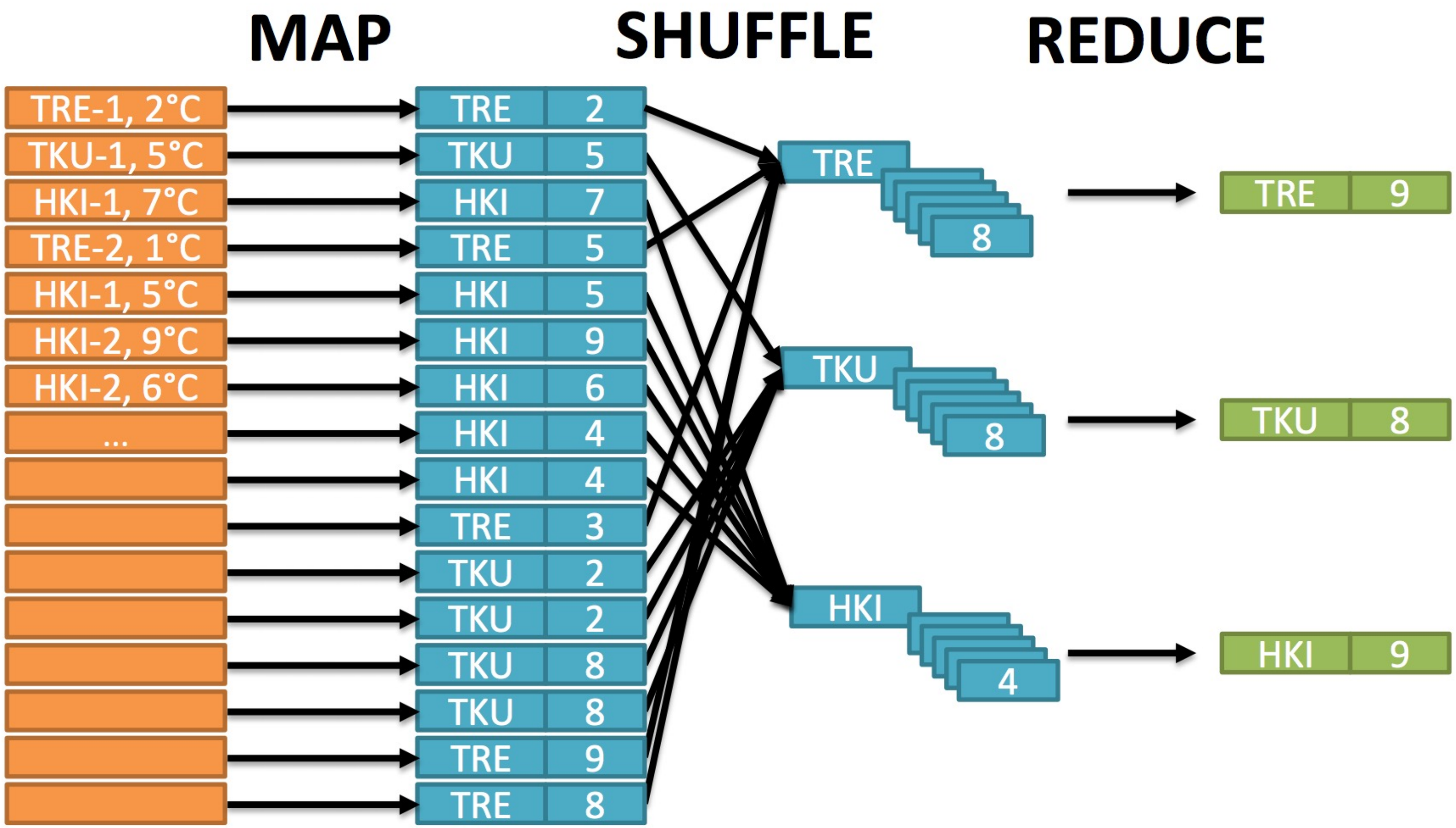
- Local aggregation of key/value pairs after map() and before the shuffle & sort phase (occurs on the same machine as map())
  - mini-reducer
- Instead of emitting 100 times (the,1), the combiner emits (the,100)
  - If it's too complex, it's not scaling-out
  - Has no access to other mapper's key/value pairs: avg does not work!
  - Most often, combiner code != reducer code

MR Phases

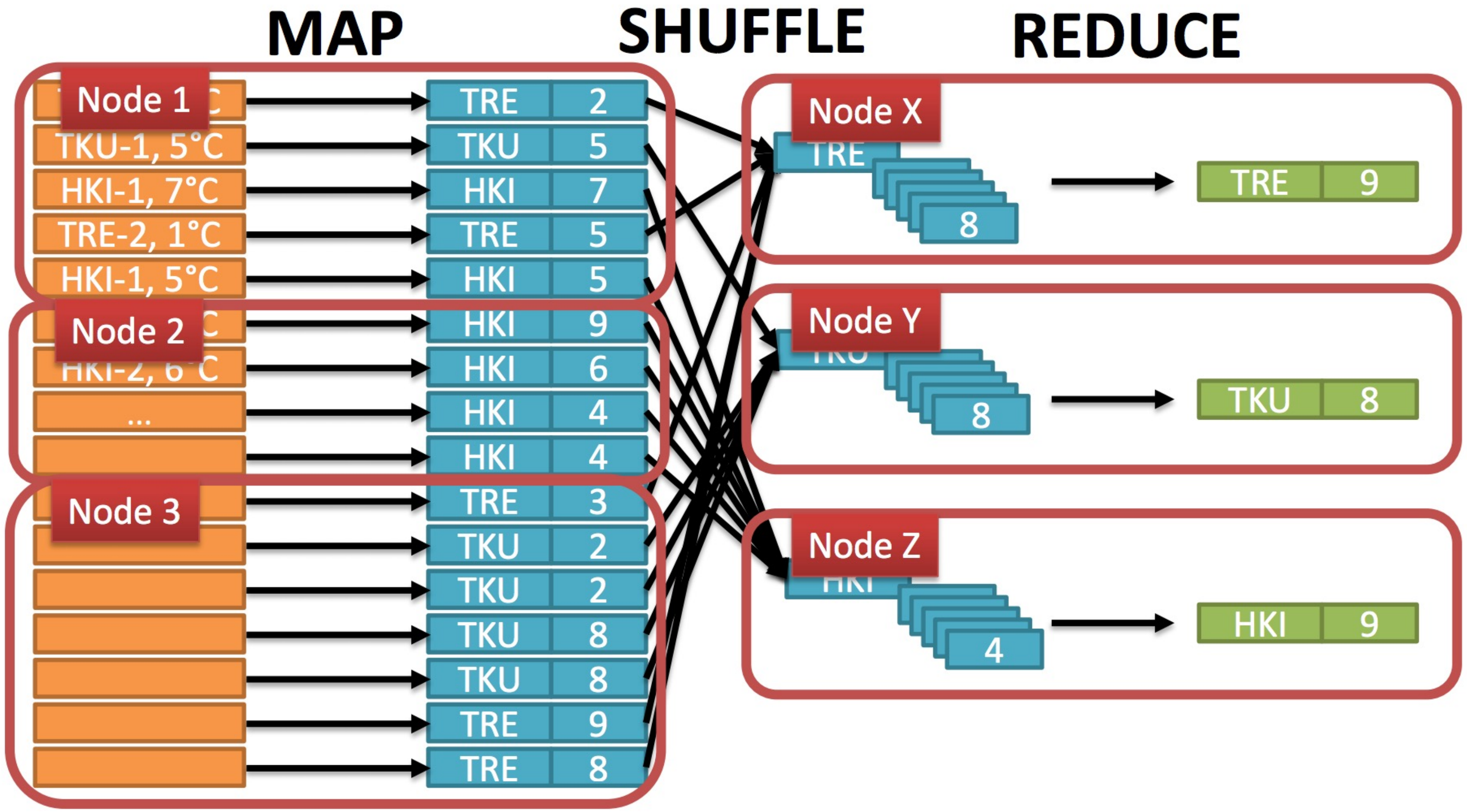


Combine: local aggregation of values with the same key
Reducers can only start calling reduce() **after all mappers** are finished
Shuffle: sorting of **intermediate key/value pairs**

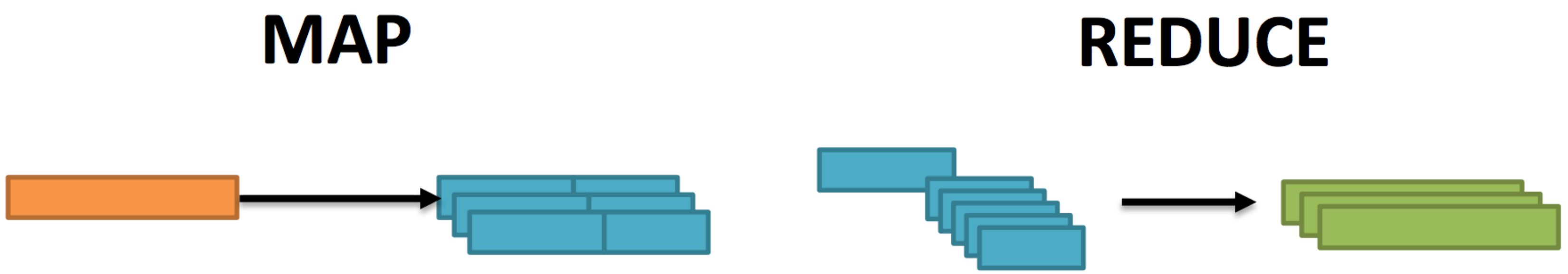
MR: Sensors data



MR: Sensors data



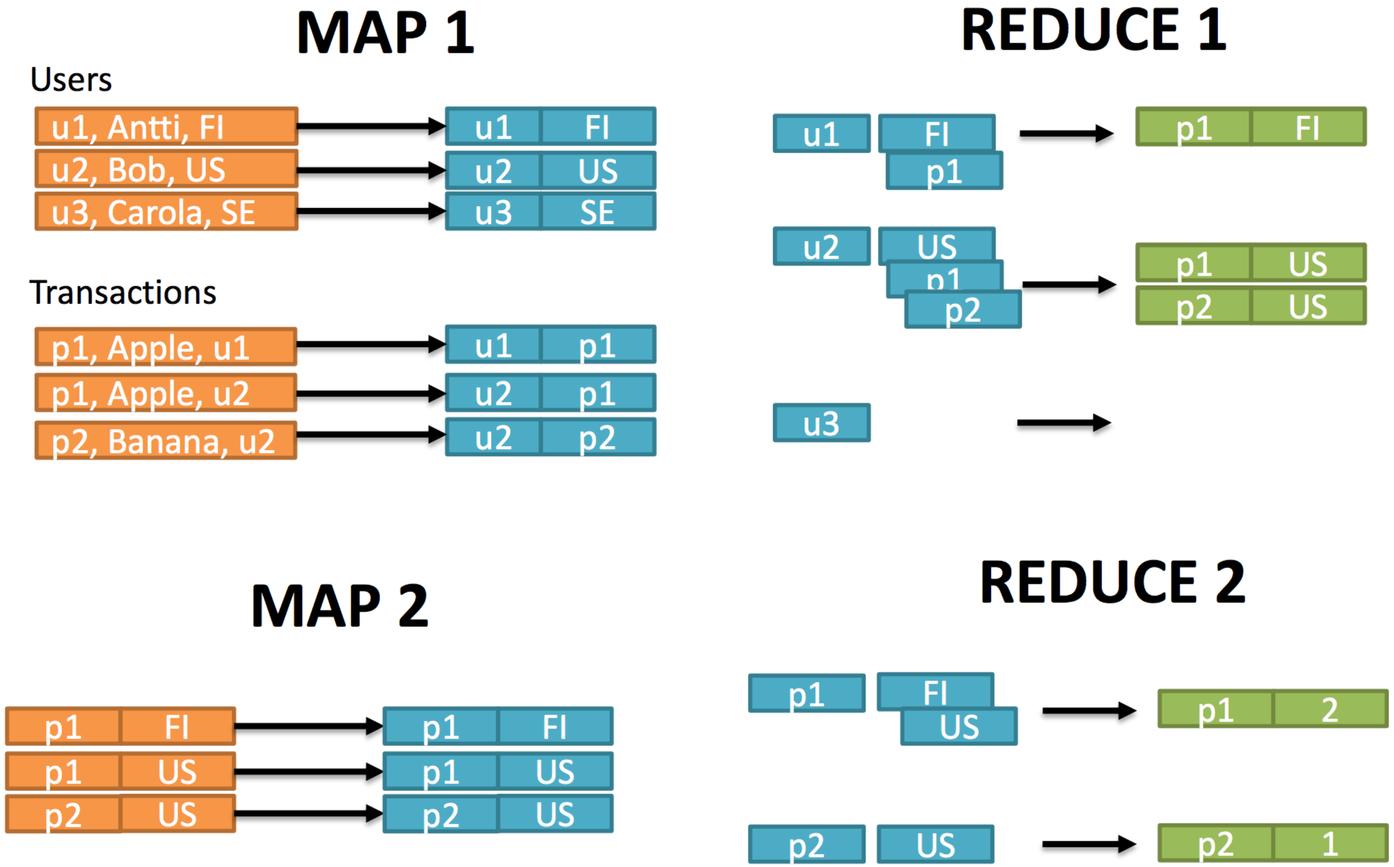
MR: Sensors data



Map(k1,v1) → list(k2,v2)

Reduce(k2, list(v2)) → list(v3)

Sequential MR jobs



Other MR application examples

- **Reverse web-link graph**
  - *Map*: Input is node-outgoing links.
    Output each link with the target as a key.
  - *Reduce*: Concatenate the list of all source nodes associated with a target.
- **Inverted index**
  - *Map*: Input is words for a document.
    Emit word-document pairs
  - *Reduce*: for the same word, sort the document IDs that contain this word; emits a pair.

Other MR application examples

- Reverse web-link graph
  - Map
    - I: Outgoing links of each node (A -> B, C -> B, B -> D)
    - O: Target as key and link as value (B,A)(B,C)(D,B)
  - Reduce
    - Concat all nodes for a target (B, (A,C)) (D, (B))

Other MR application examples

- Inverted Index
  - Map
    - I: words for a document (d1,“The lazy fox”) (d2,“The brown fox”)
    - O: word-document pairs (The,d1)(lazy,d1)(fox,d1)(The,d2)(brown,d2)(fox,d2)
  - Reduce
    - For each distinct word, sort doc ids with the word
    - (the, (d1,d2)) (lazy, (d1)) (fox, (d1,d2)) (brown, (d2))

Hadoop Design

- Centralized master
- "Pull" based communication model
  - Reduce tasks fetch files from mappers
  - Provides cheaper **fault recovery** and room for dynamic **scheduling of tasks**

MR Job Management

- Worker failure:
  - Master pings workers periodically
  - If worker down, then master reassigns the task to another worker
- Choice of M and R:
  - Larger is better for load balancing
  - Limitation: master needs $O(M \times R)$ memory

Job Scheduling

- Number of tasks (e.g., Map or Reduce operations from multiple users) can exceed number of tasks that can run concurrently on the cluster
- Scheduler maintains task queue and tracks progress of running tasks
- Waiting tasks are assigned nodes as they become available

- Scheduler starts tasks on node that holds a particular block of data needed by the task, if possible (“**move code to data**”)

Job Scheduling

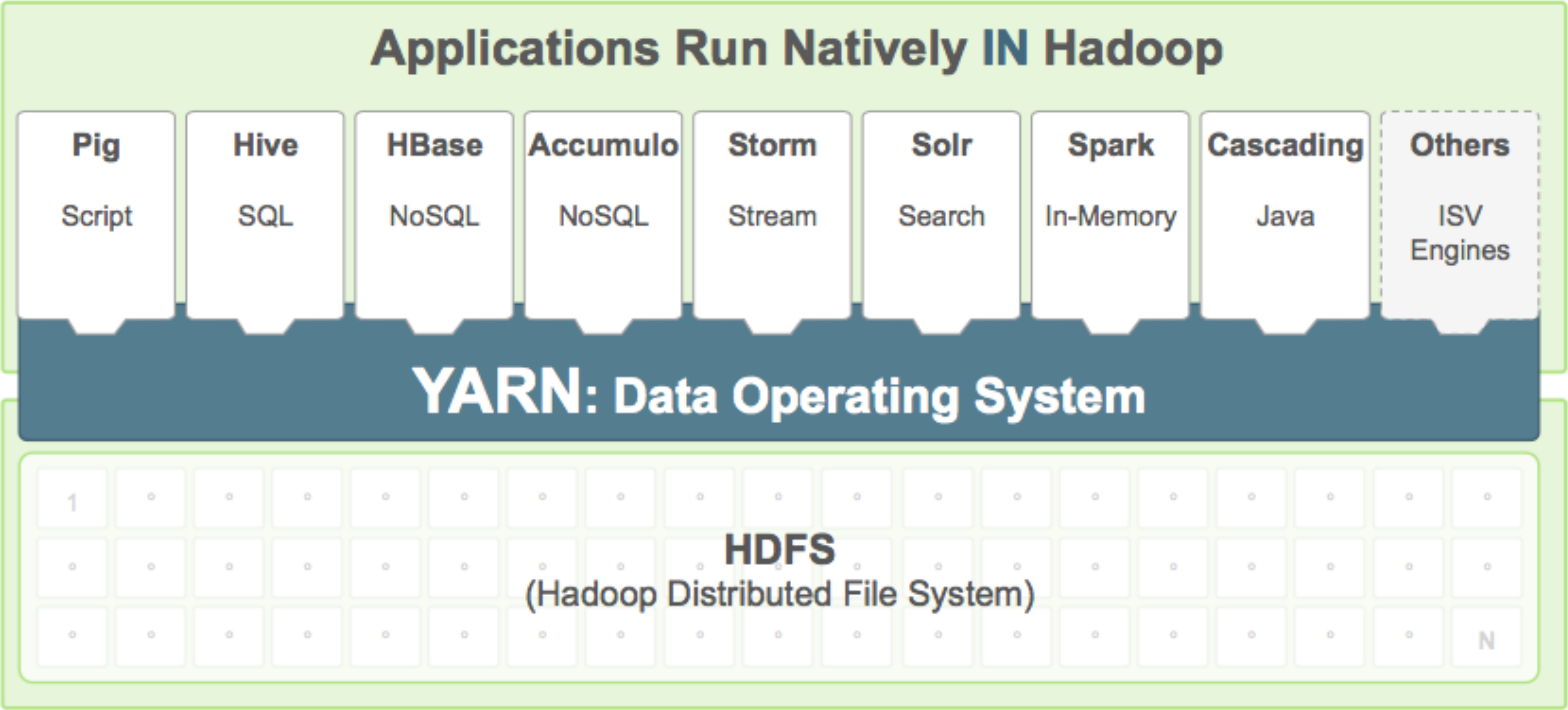
- FIFO (first-in-first-out) scheduler
- Consider Job priorities
  - Next job to be executed is the one with highest priority

- Fair scheduling
  - Every user receives a fair share of the cluster
  - Jobs run in parallel getting a share of the resources

Hadoop 2 - YARN

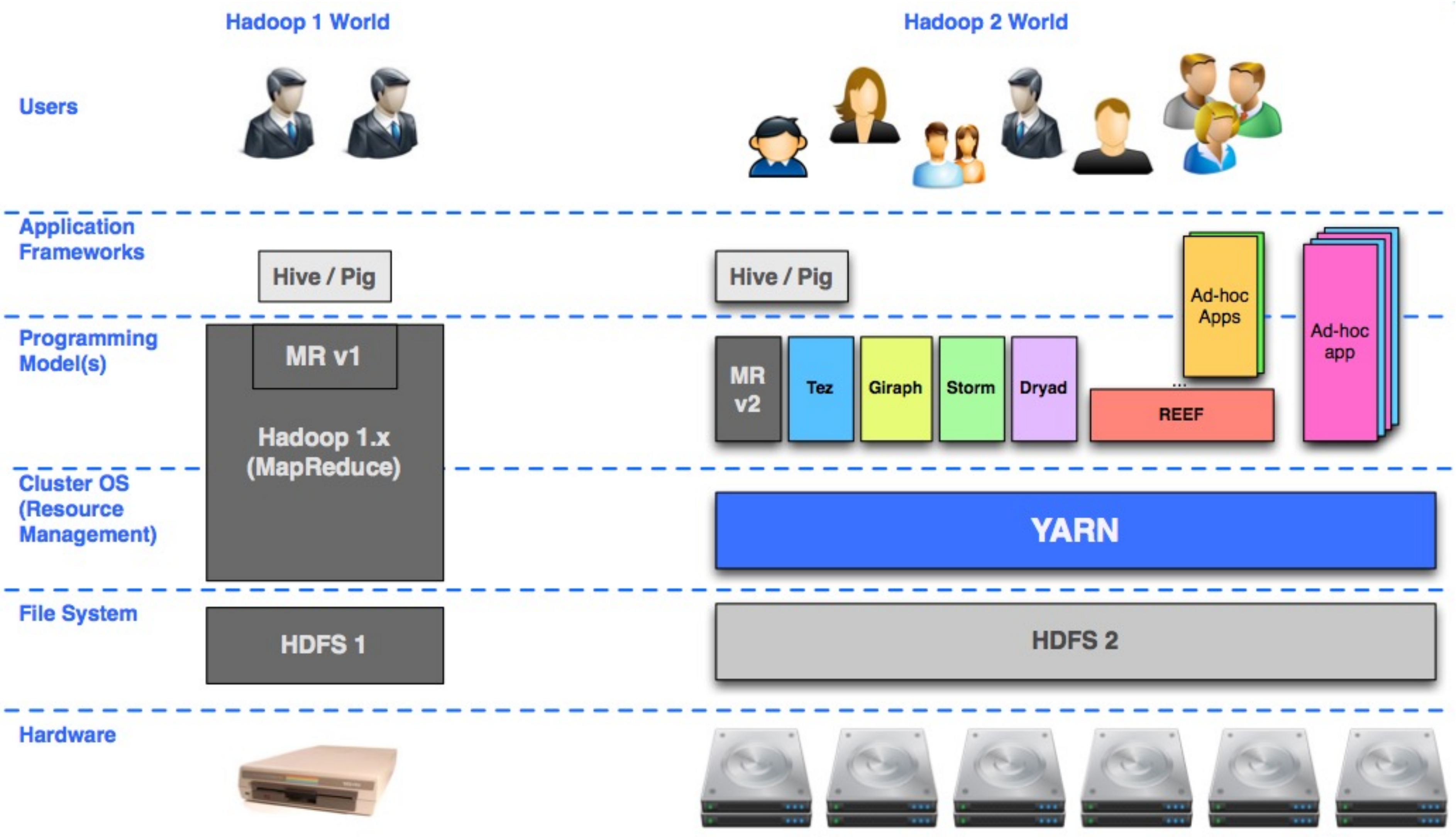
- YARN (Hadoop 2.x, used in production at Yahoo!)
- Cluster management technology
- The “operating system” on top of HDFS
- Map/Reduce is an application on top

YARN



By hortonworks.com

Hadoop 1 versus 2

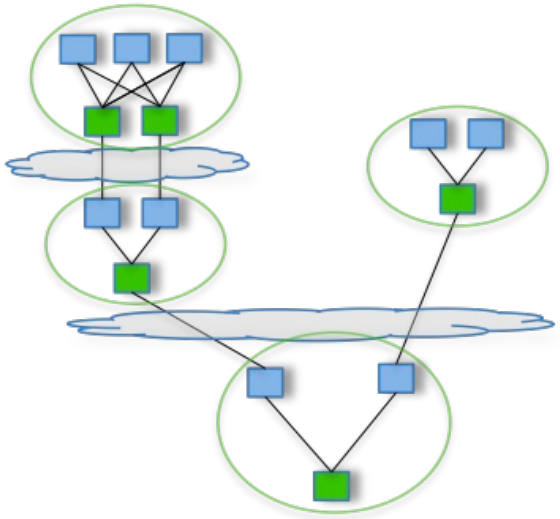


Hadoop 3

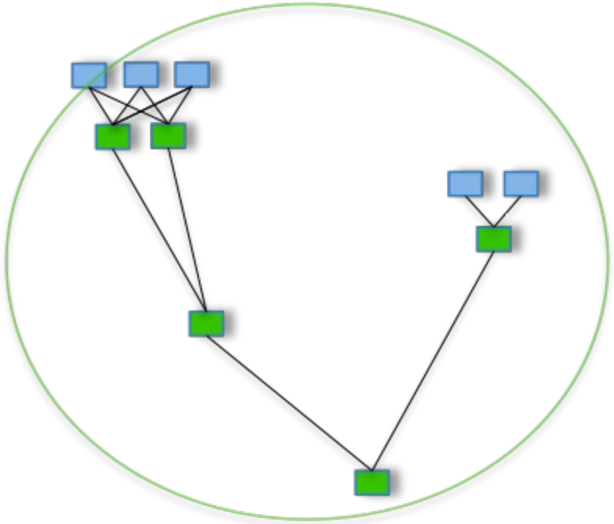
- Multiple Namenodes
- Better replica storage management (more scalable, less storage needed)
- Disk balancing for data storage
- Improved job scheduling (preemption)

Apache TEZ

- Complex execution of tasks
- Built on top of YARN



Pig/Hive - MR



Pig/Hive - Tez

- More efficient, Better data flow decisions
  - As compared to M/R over Hadoop

Spark

- Generalizes MapReduce while retaining its scheduling and fault tolerance benefits
- Main addition: efficient data sharing
  - Scheduler aims to put jobs where the data is
- Enables more applications
  - Iterative algorithms
  - Interactive queries
  - Stream processing

Spark implementation

- https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html

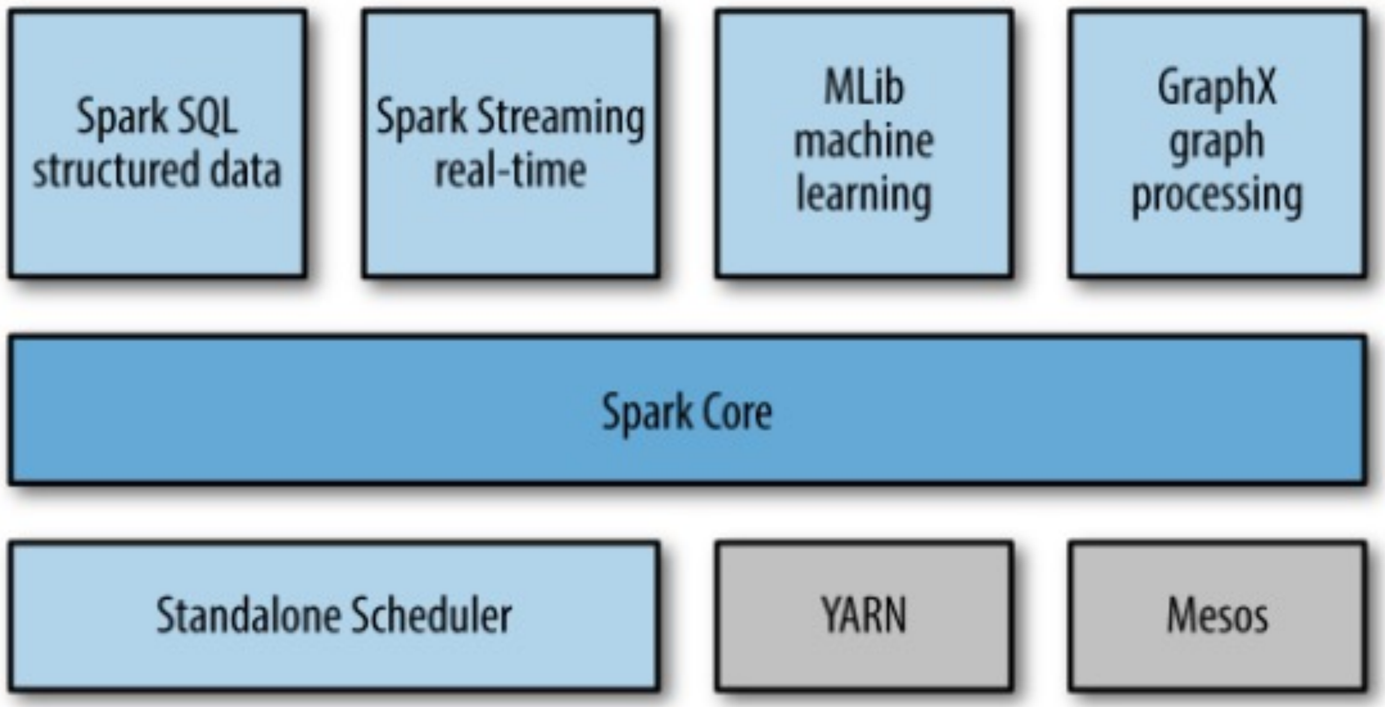


Figure 1-1. The Spark stack

Spark in Python

- PySpark package

```python
from pyspark import SparkConf, SparkContext
conf = (SparkConf()
         .setMaster("local")
         .setAppName("My app")
         .set("spark.executor.memory", "1g"))
sc = SparkContext(conf = conf)
```

https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html

Word Count M/R in Spark with Python

Take first element of the input

Input file

```
lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
counts = lines.flatMap(lambda x: x.split(' ')) \
              .map(lambda x: (x, 1)) \
              .reduceByKey(add)
output = counts.collect()
```

Split lines using
empty spaces to
get a list of words

Count each word
as appearing 1 time

Store results.
Because of lazy evaluation
it only computes the data now

Add as reduce function: sum of all word values

https://github.com/apache/spark/blob/master/examples/src/main/python/wordcount.py

Transformations

- **Map(function):** Return a new distributed dataset formed by passing each element of the source through a function
- **FlatMap(function):** each input item can be mapped to 0 or more output items (instead of one as for map)
- **reduceByKey(function):** When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function, which must be of type (V,V) => V.

https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-operations

SparkR

- Connect your R program to a Spark cluster from Rstudio
- SparkR package

```
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
sparkR.session(master = "local[*]", sparkConfig = list(spark.driver.memory = "2g"))
```

- convert a local R data frame into a SparkDataFrame (using "as.DataFrame()")

https://spark.apache.org/docs/latest/sparkr.html

SparkR
Machine Learning

https://spark.apache.org/docs/latest/sparkr.html

SparkR supports the following machine learning algorithms currently:

Classification

- `spark.logit`: Logistic Regression
- `spark.mlp`: Multilayer Perceptron (MLP)
- `spark.naiveBayes`: Naive Bayes
- `spark.svmLinear`: Linear Support Vector Machine
- `spark.fmClassifier`: Factorization Machines classifier

Regression

- `spark.survreg`: Accelerated Failure Time (AFT) Survival Model
- `spark.glm` or `glm`: Generalized Linear Model (GLM)
- `spark.isoreg`: Isotonic Regression
- `spark.lm`: Linear Regression
- `spark.fmRegressor`: Factorization Machines regressor

Tree

- `spark.decisionTree`: Decision Tree for Regression and Classification
- `spark.gbt`: Gradient Boosted Trees for Regression and Classification
- `spark.randomForest`: Random Forest for Regression and Classification

Clustering

- `spark.bisectingKmeans`: Bisecting k-means
- `spark.gaussianMixture`: Gaussian Mixture Model (GMM)
- `spark.kmeans`: K-Means
- `spark.lda`: Latent Dirichlet Allocation (LDA)
- `spark.powerIterationClustering (PIC)`: Power Iteration Clustering (PIC)

Collaborative Filtering

- `spark.als`: Alternating Least Squares (ALS)

Frequent Pattern Mining

- `spark.fpGrowth` : FP-growth
- `spark.prefixSpan` : PrefixSpan

Statistics

- `spark.kstest`: Kolmogorov-Smirnov Test

SparkSQL

- Query structured data as a distributed dataset (RDD) in Spark
- Load and query data from a variety of sources
  - Hive table
  - JSON files
- Hive queries
- Spark SQL - DataFrames

SparkSQL – Example commands

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val df = sqlContext.read.json("employee.json")

df.show()
df.select("name").show()
df.groupBy("age").count().show()
```

As a Python library (https://spark.apache.org/docs/2.4.0/api/python/pyspark.sql.html)

https://www.tutorialspoint.com/spark_sql/spark_sql_quick_guide.htm

System comparison (so far)

- Query/Analysis languages
  - SQL, Pig, R (sparkR), Python (pyspark), SparkSQL
- Databases
  - Relational DB systems (e.g., Oracle)
  - OLAP: Hive
  - Column-stores
    - OLTP: HBase
- Data
  - HDFS / GFS

Summary

- Zookeeper
- Map/Reduce jobs
- Hadoop, YARN, and Spark
- Python (PySpark), R

References

- **ZooKeeper: Wait-free coordination for Internet-scale systems**. Hunt et al., USENIX 2010
- **MapReduce: Simplified Data Processing on Large Clusters.** Jeffrey Dean and Sanjay Ghemawat. OSDI 2004. Sec. 1 - 4.

What’s next (Part II)

- Data Streams (week 6)
  - Apache Storm and Apache Kafka
  - Spark Streaming
- Graph data / network data (weeks 7-8)
  - (Social) Network data analytics at scale
  - Modularity, community detection
  - Link Analysis

Scenarios

- Scenario Assumptions
  - What is the data we expect?
  - Who are the users? What do they know?
  - What are the queries we expect?
  - What is the timeline?

Scenarios - Which tool is best for which use case?

1. Data generated by the Large Hadron Collider is stored and analysed by researchers
2. All pages crawled from the Web are stored by a search engine and served to clients via its search interface
3. Data generated by the Australian taxation office is used to send warning letters to tax payers ("you are late with your taxes")

- Options
    A. Relational DBMS
    B. Hadoop (HDFS, M/R, PIG, HBase etc.)
    C. Spark (HDFS, RDDs, MLLib, etc.)
    D. Streaming solution (Storm, Kafka, SparkStreaming, etc.)
    E. Graph data solution (Pregel, Giraph, Spark GraphX, etc.)