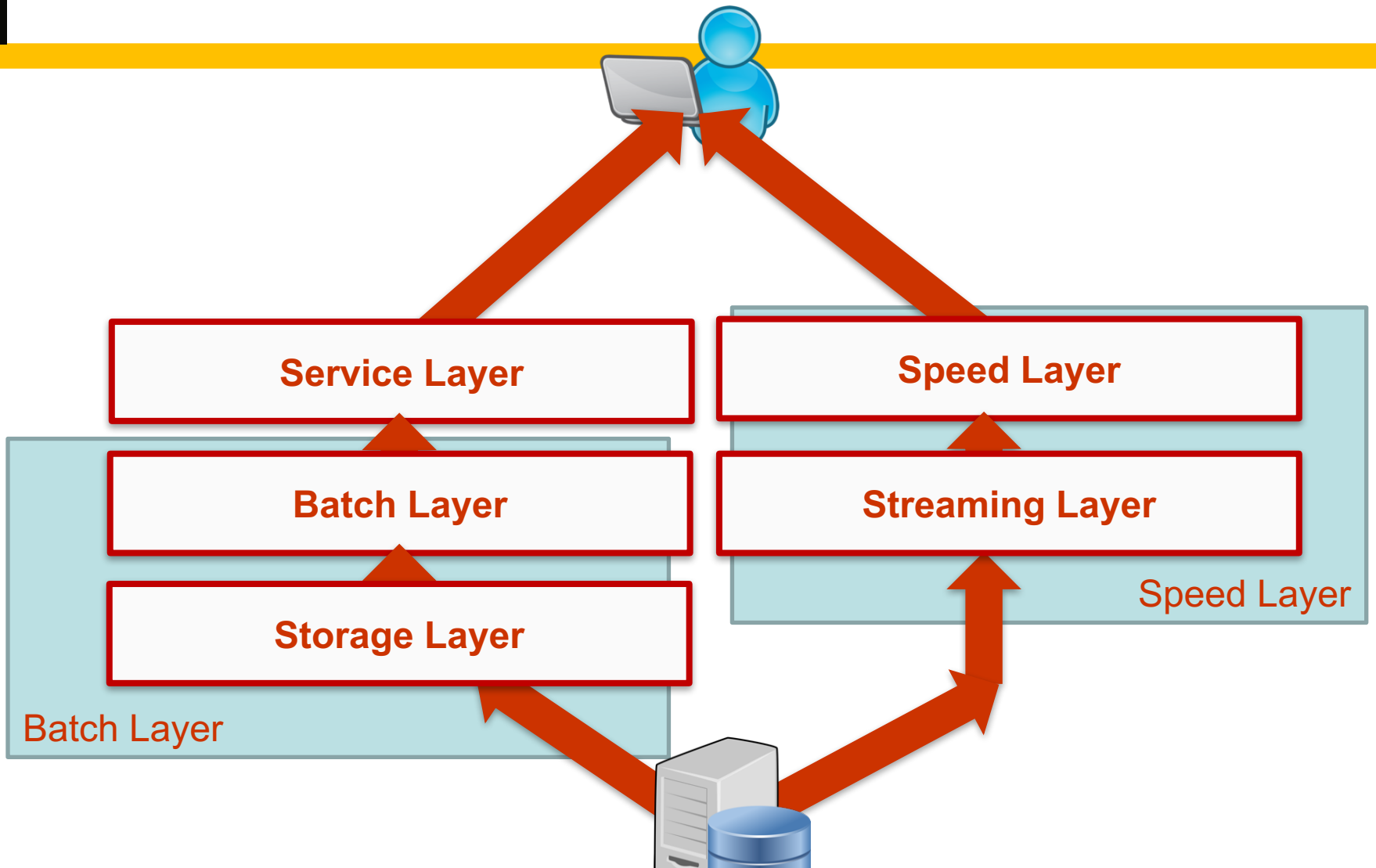
IT UNIVERSITY OF COPENHAGEN

Big Data Management
Batch Processing Tools

Björn Þór Jónsson

Framework
Lambda Architecture



Batch Layer

Input

- Distributed storage
  - HDFS (or similar)
  - Redundancy (local vs. remote)
  - Rack awareness
- Distributed shared-nothing processing

Goals

- High throughput
  - Linear scale-out
  - Efficient resource usage
  - Limited network traffic
  - Computations follow data
- Programmer productivity
  - Effective abstractions
- ~~Low latency~~

Main Ecosystems

Apache Hadoop Ecosystem

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop Data Exchange

Flume Log Collector

Zookeeper Coordination

Oozie Workflow

Pig Scripting

Mahout Machine Learning

R Connectors Statistics

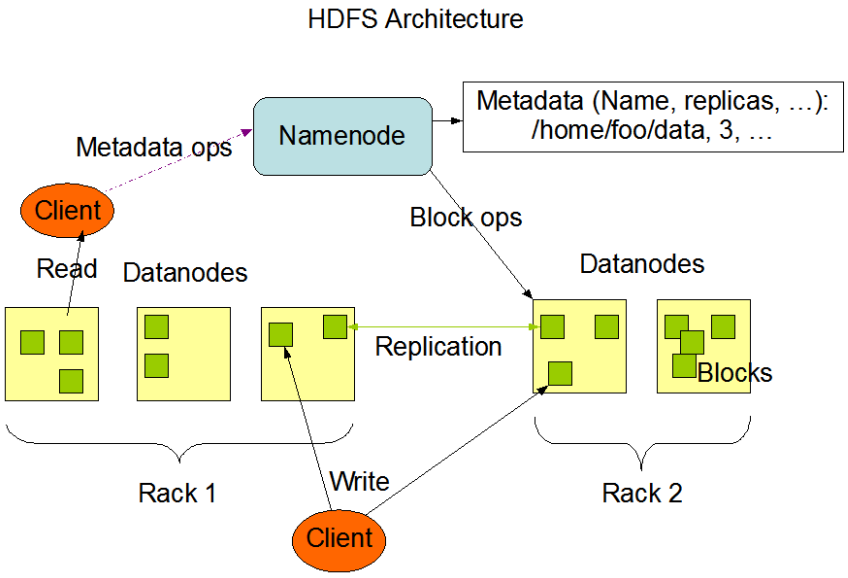
Hive SQL Query

Hbase Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System

HDFS



- Namenode [master]
  - Inode records
  - Mapping of blocks to datanode
  - Sends instructions to datanode
- Datanode [slave]
  - Inodes

A file is split into blocks (128 MB)

Each block is replicated across datanodes

Blocks are stored on local file system

http://hortonworks.com/apache/hdfs

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

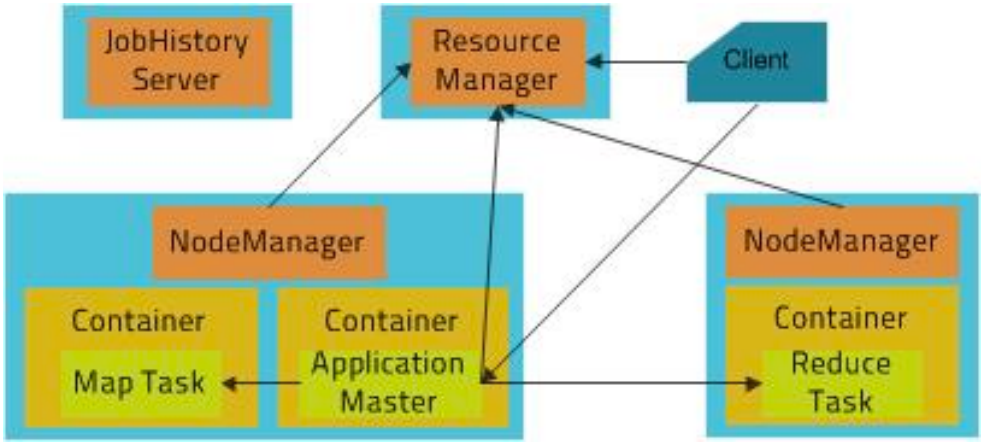
HDFS Efficiency and Availability

Feature	Description
Rack awareness	Considers a node's physical location when allocating storage and scheduling tasks
Minimal data motion	Hadoop moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides, which significantly reduces network I/O and provides very high aggregate bandwidth.
Utilities	Dynamically diagnose the health of the file system and rebalance the data on different nodes
Rollback	Allows operators to bring back the previous version of HDFS after an upgrade, in case of human or systemic errors
Standby NameNode	Provides redundancy and supports high availability (HA)
Operability	HDFS requires minimal operator intervention, allowing a single operator to maintain a cluster of 1000s of nodes

https://hortonworks.com/apache/hdfs/

YARN

Yet Another Resource Negotiator

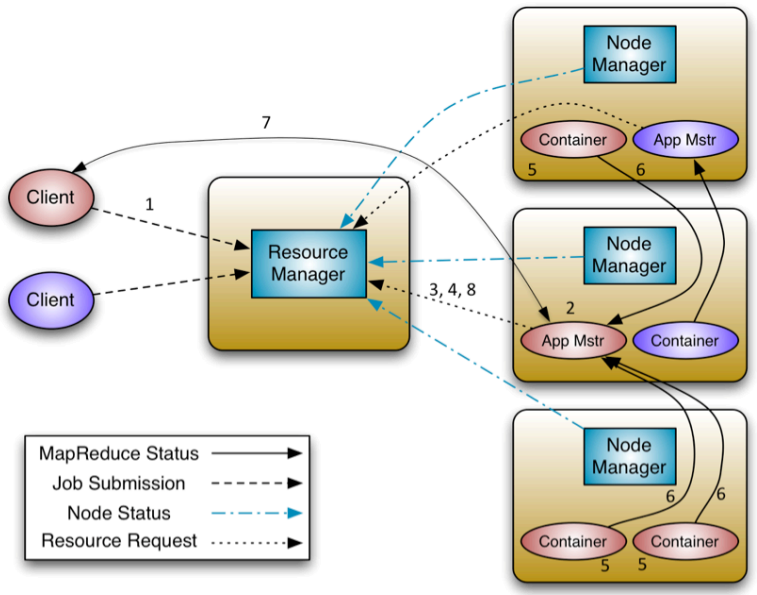


Application = AM + tasks

- Resource Manager [master]
  - A scheduler (capacity, fairness) for resource management across nodes
- Node Manager [slave]
  - Resource management on one nod
    - Vcores, memory
  - Executes instructions from resource manager and Applications Master
- Container
  - Unit of execution
  - Consumes resources
- Application Master (AM)
  - Negociates resource utilisation with Node manager
  - Responsible for application execution across containers (on several nodes)

http://dl.acm.org/citation.cfm?id=2523633
http://hortonworks.com/blog/introducing-apache-hadoop-yarn/

YARN Application Execution



1. Client submits application
2. RM launches AM
3. AM registers with RM
4. Resource negotiation between AM and RM
5. AM gives NM instructions for Container launch
6. Application executes and gives feedback to AM
7. Client communicates with AM to get status (app specific)
8. Once application completes, AM deregisters from RM. AM shuts down. RM tells NM to repurpose containers.

Might repeat

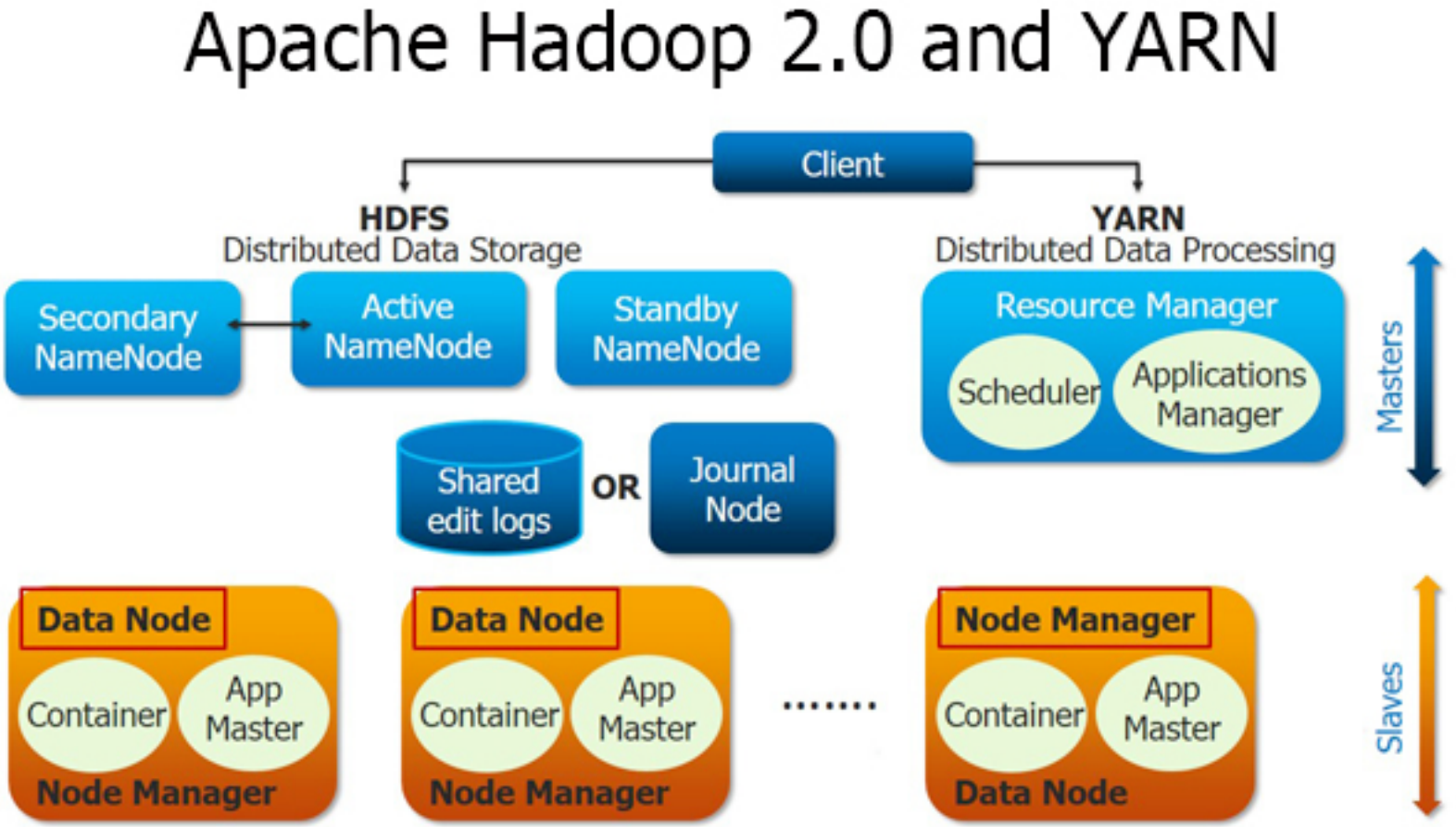
YARN Failure Handling

- Container failures are handled by NM
  - If Application master fails
    - Report to RM
    - RM starts new application master in a new container
  - Otherwise, restart in a new container
- NM failure
  - Handled by RM (heartbeat)
  - Application master *might* recover tasks running on containers on the failed node (application specific)
- RM failure
  - Single point of failure
  - Blocking clients
  - Solution:
    - RM state is persistent (through HDFS or Zookeeper) and can be recovered in case of failure
    - Multiple RM shared same state (redundancy)

Typical Hadoop Cluster

- Keeping two types of machines separated:
  - Masters: HDFS namenode and YARN resource manager
  - Slaves: HDFS data node and YARN node manager
- Provides co-location of task and data
- Possibility to easily decommission slave machines

Hadoop

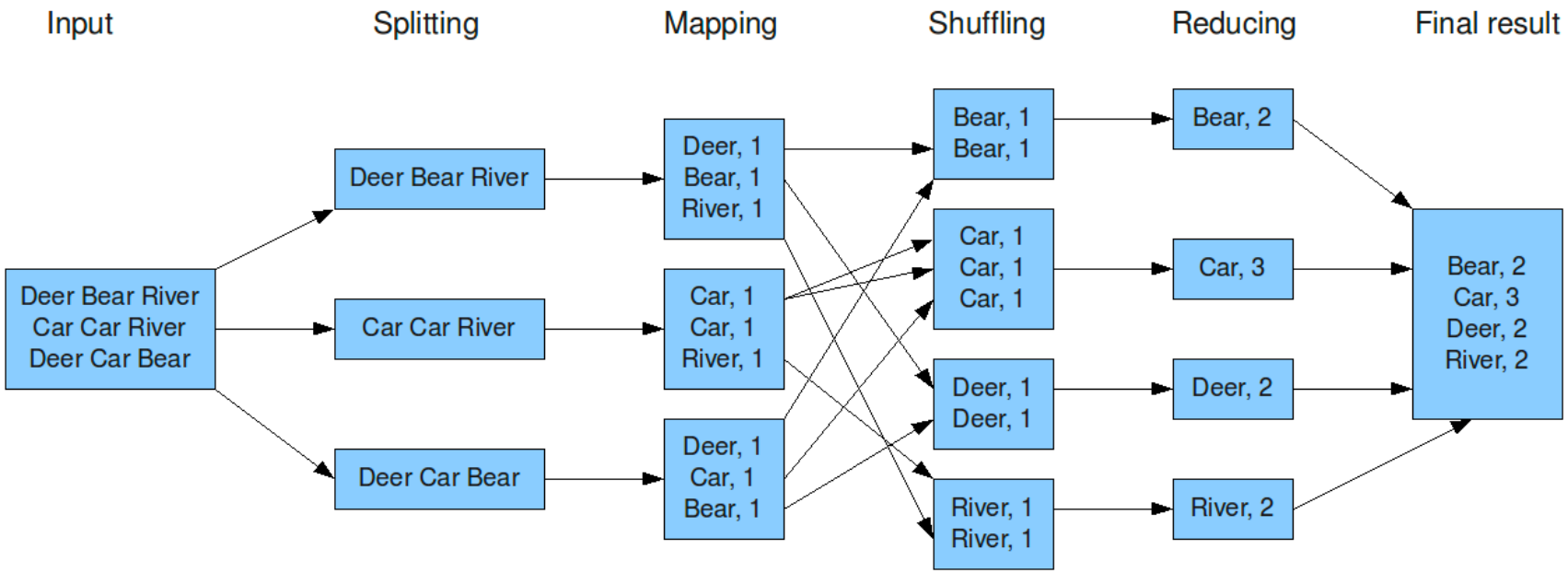


http://bigdataanalyticsnews.com/introduction-apache-hadoop-big-data/

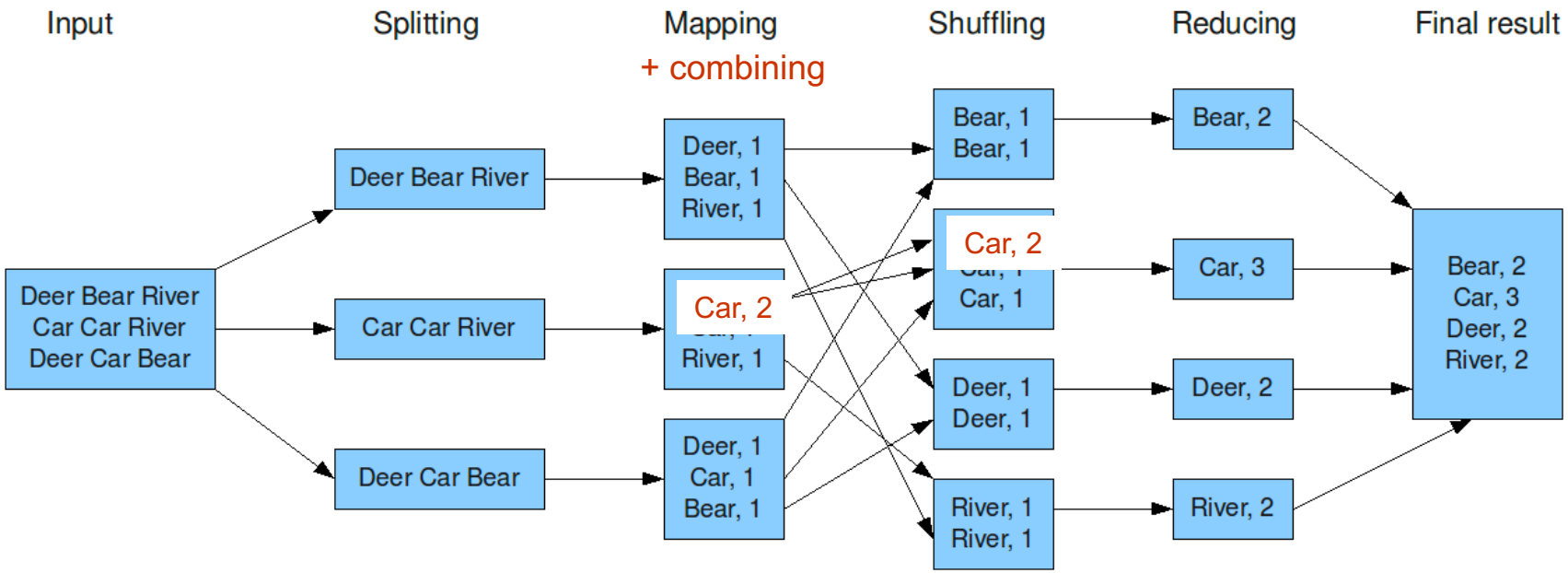
Word Count Example

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  @Override
  public void map(LongWritable key, Text value,
                  Mapper.Context context) throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }
  }
}

public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable value : values) {
      sum += value.get();
    }

    context.write(key, new IntWritable(sum));
  }
}
```

MapReduce

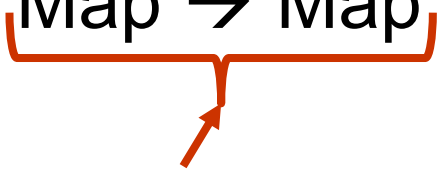

Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

MapReduce (optimized)



Hadoop Problems

- Containers prevent sharing resources
  - Large data structures → one per thread!
- Rigid MapReduce pipeline
  - Low-level abstraction
  - Map → Map → Reduce → Map → Reduce



Complex two-step Map operation

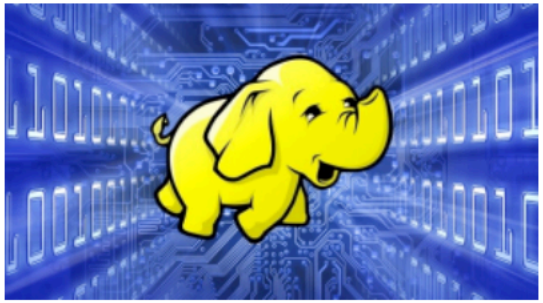
Must write results to disk

- Limited successes

March 13, 2017

Hadoop Has Failed Us, Tech Experts Say

Alex Woodie



The Hadoop dream of unifying data and compute in a distributed manner has all but failed in a smoking heap of cost and complexity, according to technology experts and executives who spoke to *Datanami*.

"I can't find a happy Hadoop customer. It's sort of as simple as that," says Bob Muglia, CEO of Snowflake Computing, which develops and runs a cloud-based relational data warehouse offering. "It's very clear to me, technologically, that it's not the technology base the world will be built on going forward."

Hadoop's strengths lie in serving as a cheap storage repository and for processing ETL batch workloads, Johnson says. But it's ill-suited for running interactive, user-facing applications, he says.

Tez: Multistep Dataflow

Algebraic model

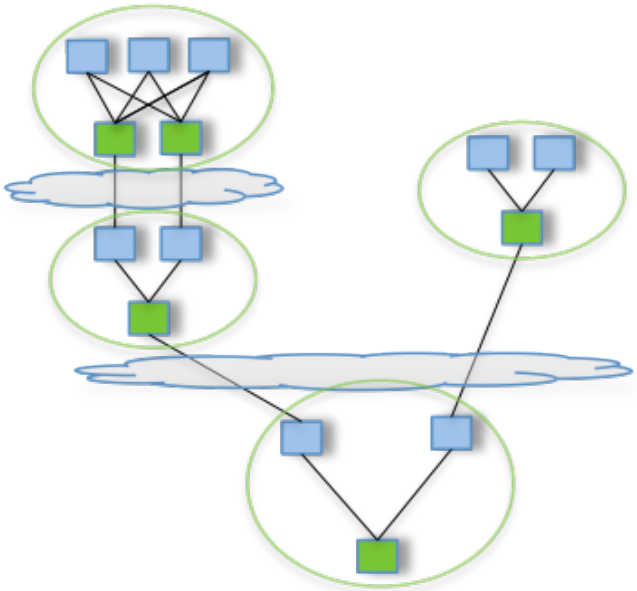
- Task accepts input(s) and generates output(s)
- Tasks can be composed into a DAG
  - Vertices: tasks
  - Edges: data movement
    - 1-1, broadcast, scatter-gather
    - Sequential/concurrent
    - Persisted/reliable/ephemeral
- No storage on HDFS across tasks

Programming

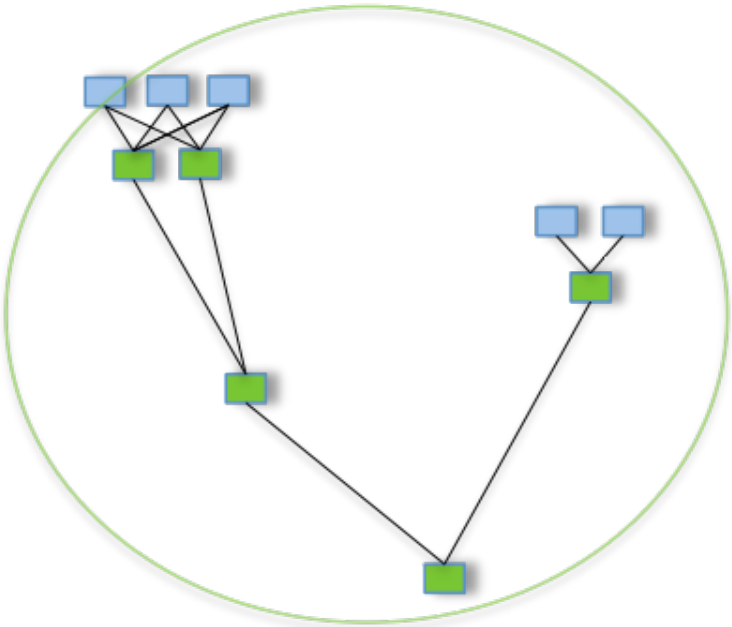
- DAG API
  - Define tasks (vertices) and data movement (edges)
- Run-time API
  - Task programming and library utilisation
    - Input / Processor /Output
    - Built-in: HDFS Input/Output, KV Input/Output, SortedGroupedPartitioned KV Input/Output

https://tez.apache.org

Map-Reduce vs. Tez



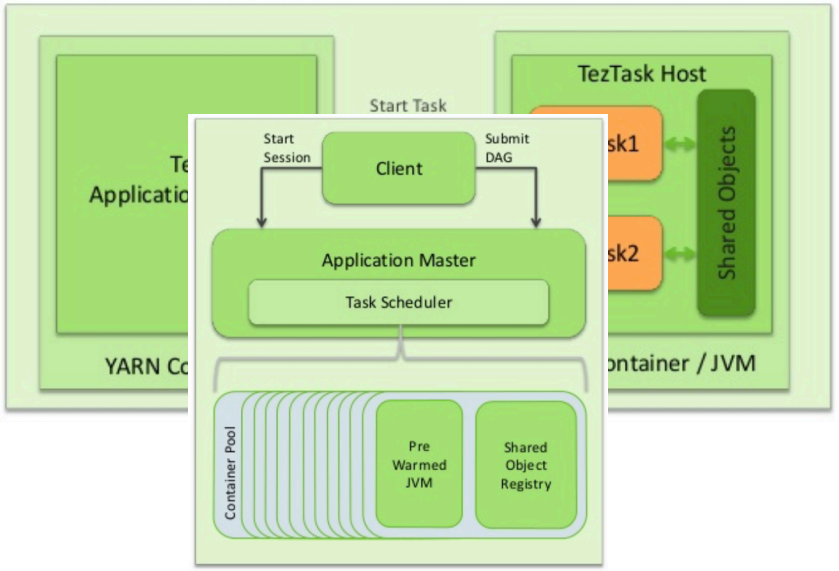
Pig/Hive - MR



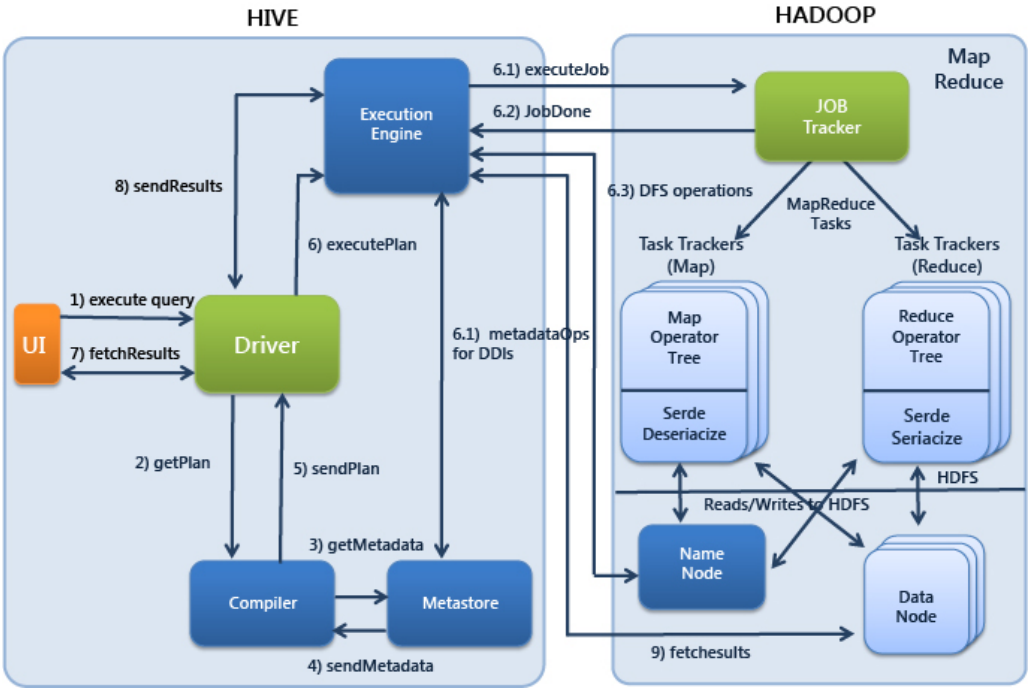
Pig/Hive - Tez

Tez Advantages

- No HDFS storage across tasks in a DAG
- Re-use of resources
  - Reuse YARN containers to launch new tasks
  - Shared in-memory data across tasks (within a container)
- DAG modifications at run-time
  - Vertex manager
  - DAG scheduler
  - Task scheduler

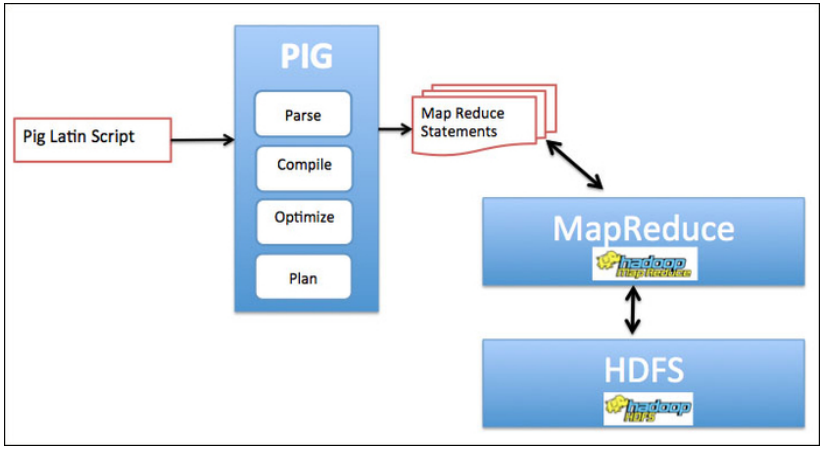


Hive



https://hive.apache.org

Pig



Pig-latin example

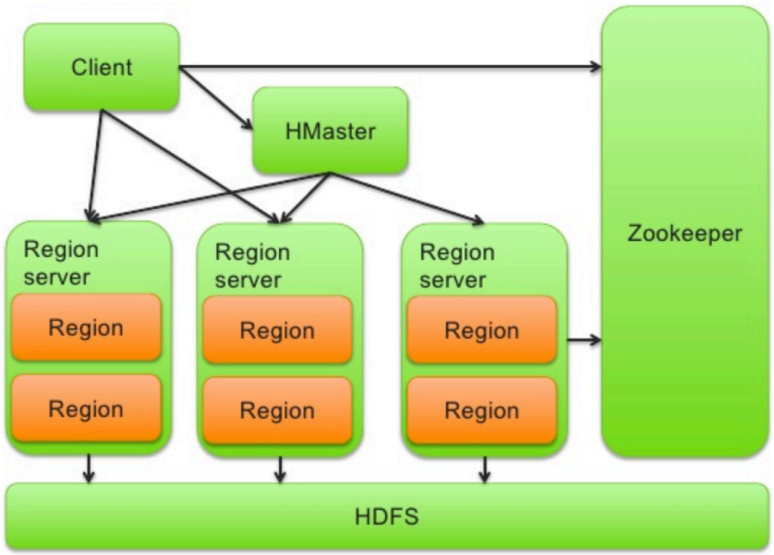
- Query : Get the list of pages visited by users whose age is between 20 and 25 years.

```
users = load 'users' as (name, age);

users_18_to_25 = filter users by  age > 20 and age <= 25;

page_views = load 'pages' as (user, url);

page_views_u18_to_25  = join users_18_to_25 by name,
page_views by user;
```

Pig vs. Hive

- Both originally designed for Map-Reduce, now moved to Tez
- Hive:
  - Declarative (SQL)
  - Structured data (requires schema meta-data)
- Pig:
  - Procedural (latin – algebraic operators)
  - No assumptions about the data
    - Might require custom load function (defaults include JSON, Hbase, Text)

HBase



- BigTable data model
  - Sparse, multi-dimensional map
    - Row key, column key, timestamp
  - Each value is an array of bytes
- Hbase tables are split into region
- Each region is managed by a region manager
- Region managers are slaves
- Hmaster is master
  - Control path through master to create, modify, delete table
  - Data path bypasses master. Client is in direct contact with region server when reading/writing data.

http://hbase.apache.org

Mahout
Machine Learning


```
DataModel model = new FileDataModel(new File("/path/to/dataset.csv"));
UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
UserNeighborhood neighborhood =
            new ThresholdUserNeighborhood(0.1, similarity, model);
UserBasedRecommender recommender =
            new GenericUserBasedRecommender(model, neighborhood, similarity);
List recommendations = recommender.recommend(2, 3);
for (RecommendedItem recommendation : recommendations) {
  System.out.println(recommendation);
}
```

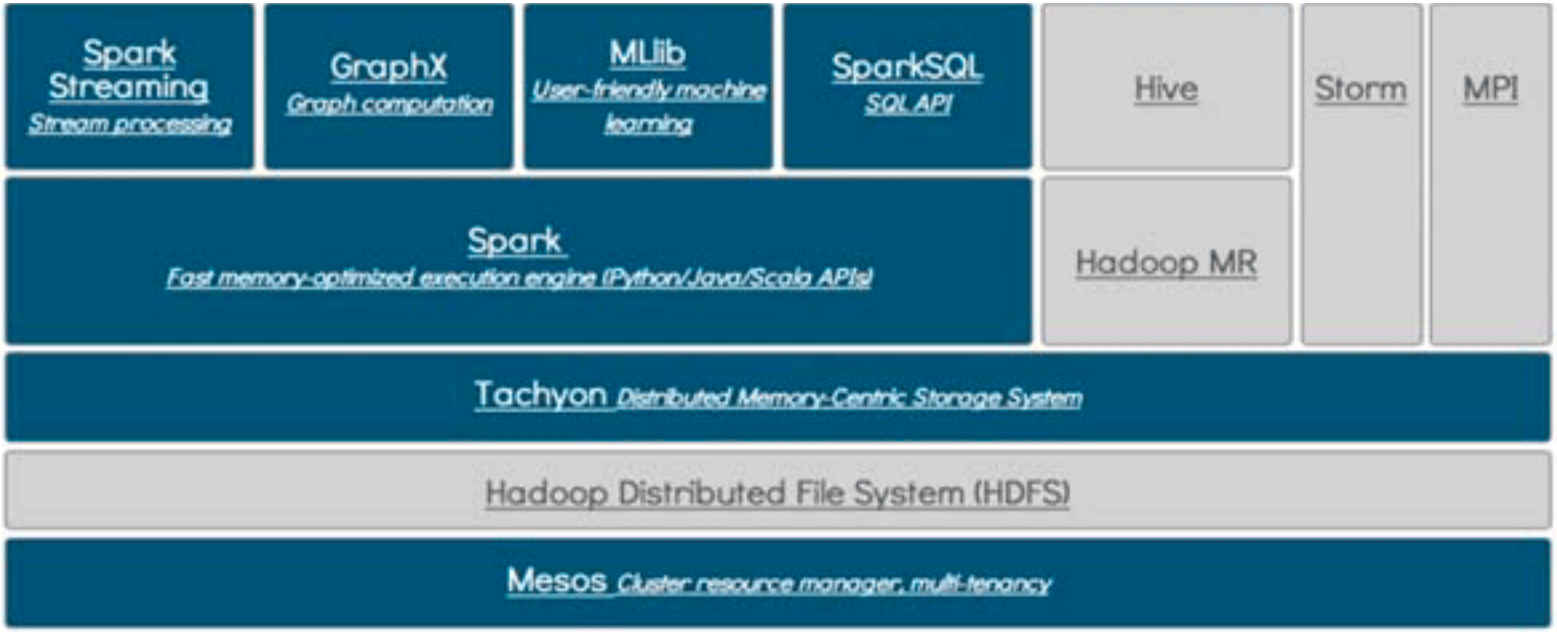
https://mahout.apache.org

https://mahout.apache.org/users/recommender/userbased-5-minutes.html

Hadoop Ecosystem

- Hadoop: Doug Cutting's son's toy **elephant** name
- YARN: Yet Another Resource Negotiator
- HDFS: Hadoop File System
- Tez: Hindi for "speed"
- Hive
- Pig
- HBase
- Mahout: A person who drives an **elephant**

Spark Ecosystem



Spark RDDs

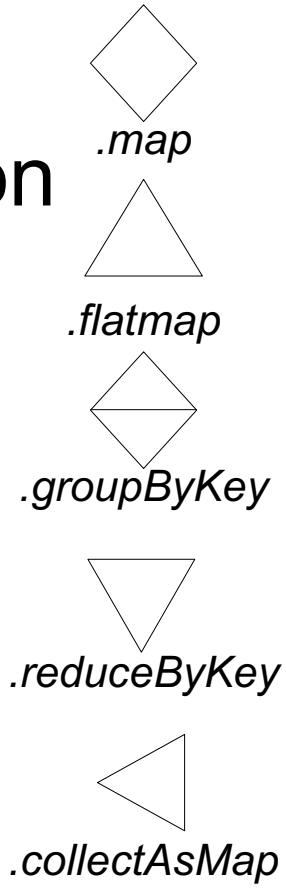
- Resilient Distributed Datasets (RDDs)
  - Transform one RDD to another via operators
  - Lazy execution – optimizations
- Supports deep pipelines
- Supports worker's memory sharing

Spark Pipeline Symbols

- **.map** = one-to-one transformation
- **.flatmap** = one-to-any transformation

- **.groupByKey** = Hadoop's Shuffle
- **.reduceByKey** = Hadoop's Reduce

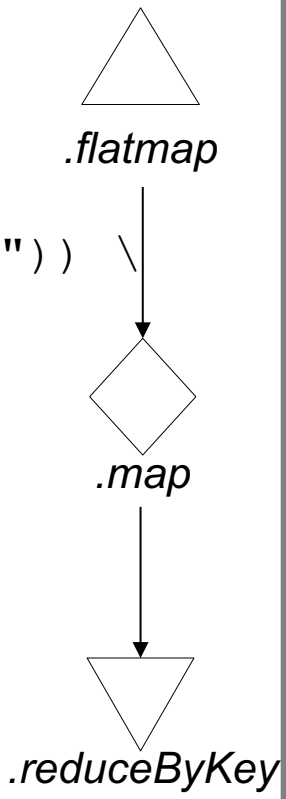
- **.collectAsMap** = collect to Master
- …



http://www.itu.dk/people/bjth/Papers/mmsys2017.spark.pdf

Spark Word Count

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```



http://spark.apache.org/examples.html

Spark SQL

- Structured data within Spark programs
  - DataSet/DataFrame as data representation
  - SQL as query language
  - Improved data parsing

```
import org.apache.spark.sql.types._

// Create an RDD
val peopleRDD = spark.sparkContext.textFile("examples/src/main/resources/people.txt")

// The schema is encoded in a string
val schemaString = "name age"

// Generate the schema based on the string of schema
val fields = schemaString.split(" ")
  .map(fieldName => StructField(fieldName, StringType, nullable = true))
val schema = StructType(fields)

// Convert records of the RDD (people) to Rows
val rowRDD = peopleRDD
  .map(_.split(","))
  .map(attributes => Row(attributes(0), attributes(1).trim))

// Apply the schema to the RDD
val peopleDF = spark.createDataFrame(rowRDD, schema)

// Creates a temporary view using the DataFrame
peopleDF.createOrReplaceTempView("people")

// SQL can be run over a temporary view created using DataFrames
val results = spark.sql("SELECT name FROM people")

// The results of SQL queries are DataFrames and support all the normal RDD operations
// The columns of a row in the result can be accessed by field index or by field name
results.map(attributes => "Name: " + attributes(0)).show()
// +-------------+
// |        value|
// +-------------+
// |Name: Michael|
// |   Name: Andy|
// | Name: Justin|
// +-------------+
```

http://spark.apache.org/docs/latest/sql-programming-guide.html

Spark vs. Hadoop

Resource Management

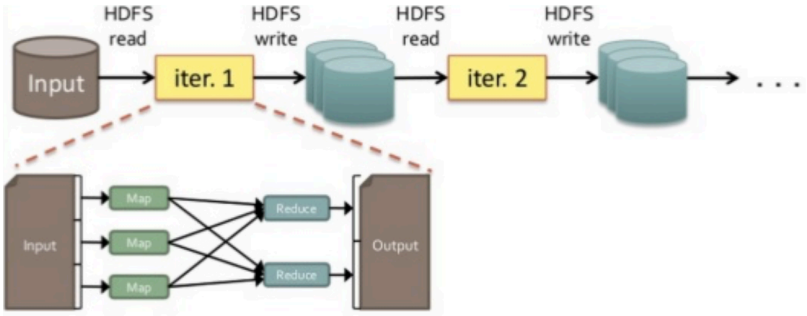
- Mesos vs. YARN:
  - C++ vs. Java
  - Both memory and CPU resources
  - Mesos uses Linux container groups, YARN uses Linux processes as containers
  - In Mesos, the scheduler proposes schedules that the client accepts/rejects, while in YARN, the client gives preferences to the scheduler (e.g., locality).

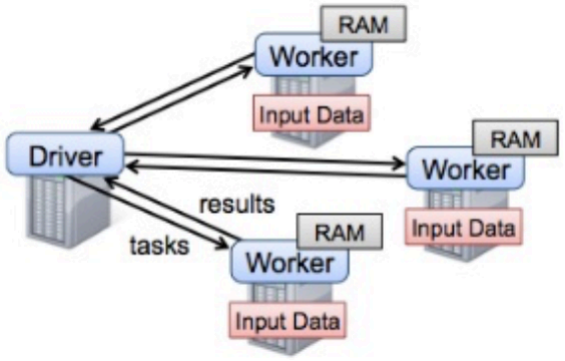
Efficient Data Sharing

- Tachyon (now Alluxio) vs. Tez (sharing data within a container)
- Tachyon/Alluxio is a memory-centric file system
  - Single namespace over cluster
  - Memory caching of persistent data through Resilient Distributed Data sets (RDD)
  - RDDs provide efficient fault tolerance by (i) constraining the possible transformations on partitions (map, filter and join), and (ii) managing lineage for each RDD (the set of transformations that led to the current state of a partition).
  - Each RDD is immutable.

https://www.mapr.com/blog/apache-mesos-vs-hadoop-yarn-whiteboard-walkthrough

Spark vs. Hadoop





https://people.eecs.berkeley.edu/~haoyuan/papers/2013_ladis_tachyon.pdf

Spark vs. Tez

- Master-slave architecture
- Executor/Container
- Caching across tasks within an executor: Memory vs. storage
- Scala/Python/Java vs. Java

MLIB vs. Mahout

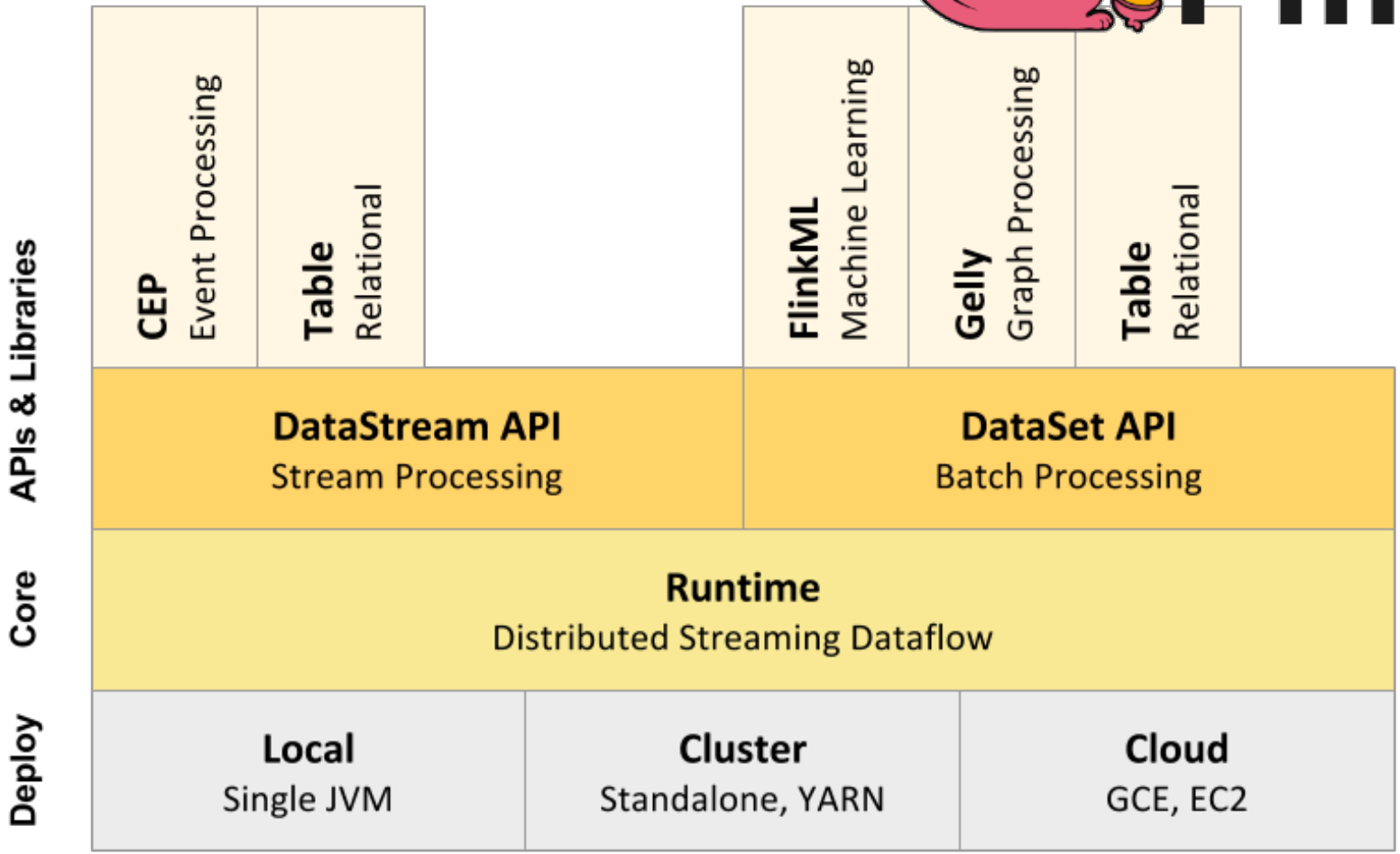
Model Type	Algorithm
Binary Classification	Linear SVM
Regression & Binary Classification	Gradient-Boosted Trees
Binary & Multiclass Classification*	Logistic Regression, Naïve Bayes
Regression & Binary, Multiclass Classification	Decision Trees, Random Forests
Regression	Linear Least Squares (Lasso, Ridge), Isotonic Regression
Recommender Engine	***Collaborative Filtering** (Alternating Least Squares)
Clustering	K-means, Gaussian Mixture, Power Iteration, Latent Dirichlet Allocation
Dimension Reduction	Principal Component Analysis, Singular Value Decomposition
Itemsets	Frequent Pattern Mining: FP-growth
Graph Algorithms	***Page Rank**, Connected Components, Triangle Counting

https://spark.apache.org/docs/1.2.0/mllib-linear-methods.html

http://spark.apache.org/docs/latest/mllib-guide.html

Flink

Flink



https://flink.apache.org/introduction.html

Take Away Points

- Layered architecture
  - Storage management
  - Cluster resource management
  - Dataflow programming
- Hadoop and Spark = complete ecosystems
  - Complementary
  - **Spark more versatile, more efficient**
  - Both constitute very heavy stacks