Recommendation with Matrix Factorization

By Stark

Objective

- Understand how recommendation systems work and how is matrix factorization is used in this.
- Learn about MapReduce and Spark architecture.
- Learn about Spark usages.
- Understand how to implement matrix factorization using SGD. Learn about how to implement this in Spark programming.

Agenda

- **About Matrix Factorization**
- HDFS & MapReduce & Spark
- Spark Usage Overview
- Prototype your program
- Optimization

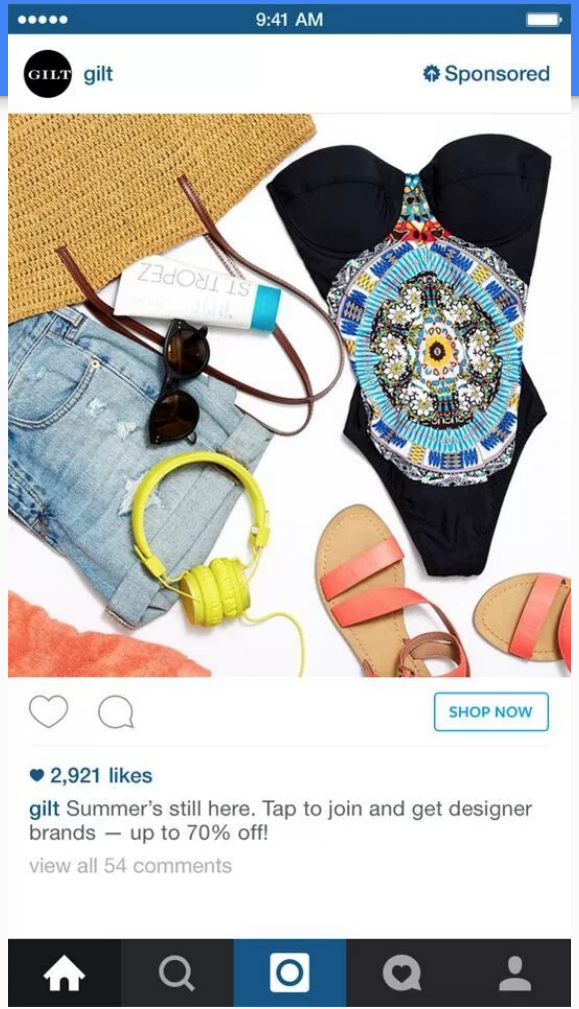
For What

ADVERTISING

Instagram gets serious about ads, opening platform to all

With Instagram's advertising kicking into high gear, analysts at Kenshoo, a marketing software company, predict the social network could make as much as $1 billion in annual revenue in the next three to four years. It would be enough to

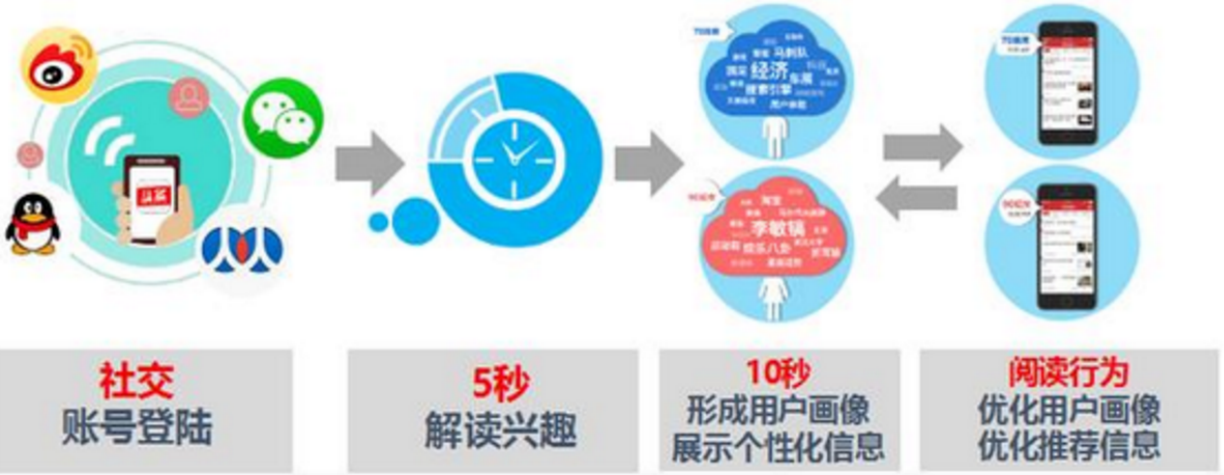
财报显示，Facebook广告收入在过去两年里增长超过3倍。研究机构eMarketer的数据更显示，2014年Facebook在移动广告市场的份额已经增长到18.4%，谷歌的市场份额则从46.6%下滑至40.5%。



（1）信息流广告：信息流广告和新闻资讯长得一模一样，当用户习惯性阅读新闻标题时，你的slogan或活动信息就会自然地融入其中，被用户接受。在推荐信息流中，所有涉及图片的新闻都以小图呈现，是今日头条的一种原生广告产品；大图模式可以让用户在一整屏信息中率先看到你。

（2）详情页广告：在资讯详情页中出现的广告，位于资讯全文结尾的下方。广告展现形式有三种：一句简单文字介绍；或一张小图＋一个标题＋一句简单的介绍，或放置一张图片banner。通过点击这些图示，可直接跳转至广告页面。

3.今日头条依据个性化推荐机制精准的将广告分发给用户（如下图的用户内容推荐流程）



Recommendation Systems

1. Search engine help us selectively find informations. But it is better to have something recommends interesting things to us automatically without asking.

1. Recommendations can be generated by a wide range of algorithms. While user-based or item-based **content/**collaborative filtering methods are simple and intuitive, **matrix factorization** techniques are usually more effective because they allow us to discover the **latent** features underlying the interactions between users and items.

Background

What is content filtering approach ?

Create a profile for each user and product to characterize its nature.

User: age, gender, answers from a questionnaire
Movies: Genre, actors, year, story ..

A program to associate user characteristics to products’

Problems:
Many information hard to collect.
Many implementations are domain specific.
Unable to find latent features.

Background

What is collaborative filtering approach ?

Assumption is, a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

More accurate than content filtering

Problems:
Suffer from “Cold Start”. -- inability to address the system’s new products and users

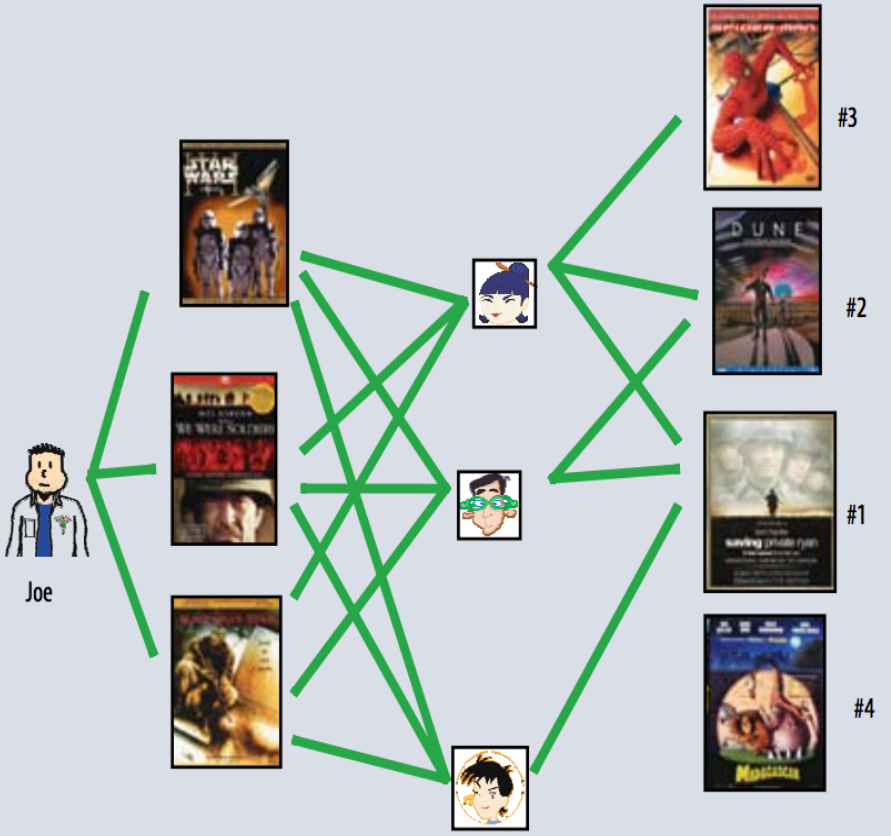


Figure 1. The user-oriented neighborhood method. Joe likes the three movies on the left. To make a prediction for him, the system finds similar users who also liked those movies, and then determines which other movies they liked. In this case, all three liked *Saving Private Ryan*, so that is the first recommendation. Two of them liked *Dune*, so that is next, and so on.

Matrix Factorization

What is it ?

Proposed by Yahoo Research: Matrix Factorization Techniques for Recommendation Systems [https://goo.gl/B1lvVE]

Simply a mathematical tool for playing around with matrices. But also because of this, it is applicable in many scenarios.

Advantages:

1. Can be used to discover latent features underlying the interactions between two different kinds of entities.
2. No Cold Start. Superior than CF from this perspective.

Problems:

1. Heavy Computation.
2. Hard to explain the results.

Matrix Factorization

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

In a recommendation system such as MovieLens, there is a group of users and a set of items. Given that each users have rated some items in the system, we would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users.

Matrix Factorization

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

In a recommendation system such as MovieLens, there is a group of users and a set of items. Given that each users have rated some items in the system, we would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users.

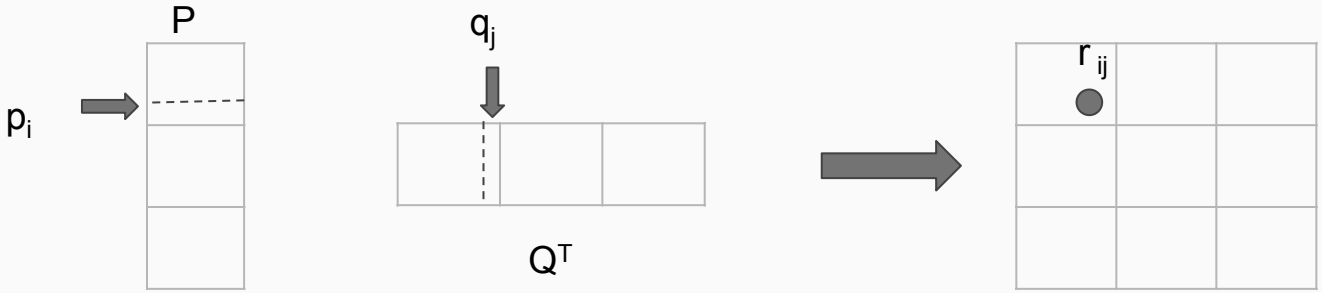
Matrix Factorization - Mathematical

Suppose we have U users and D items, let the original matrix R be the size of U * D.
Also assume we are going to discover K latent features.
Then, we are going to find two matrices P (U * K) and Q (D * K),
Such that their product approximate R

$R \sim= P * Q^T = R''$

Each row of P represents association between a user and K features.
Each row of Q represents association between an item and K features.

Matrix Factorization - SGD



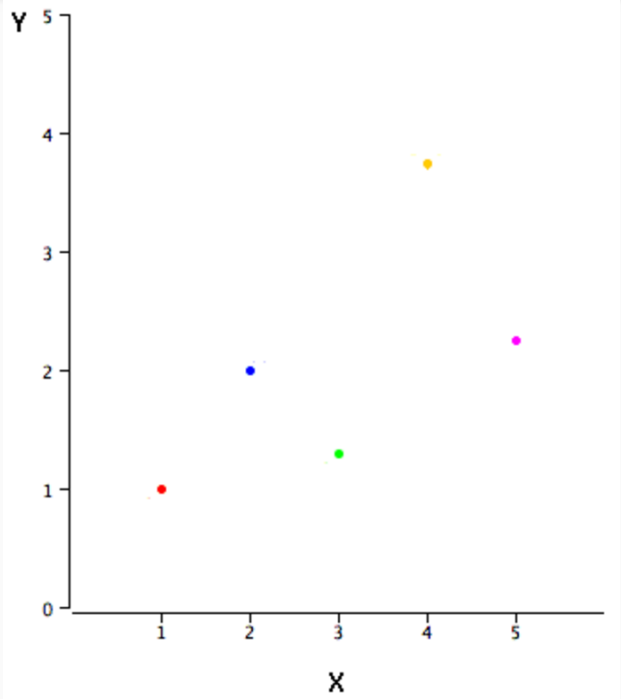
$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

? what is k here ?

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik}$$

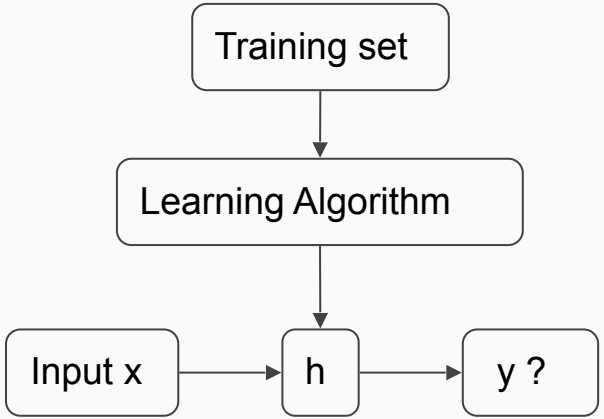
Simple Regression -- SGD explanation

X	Y
1.00	1.00
2.00	2.00
3.00	1.30
4.00	3.75
5.00	2.25



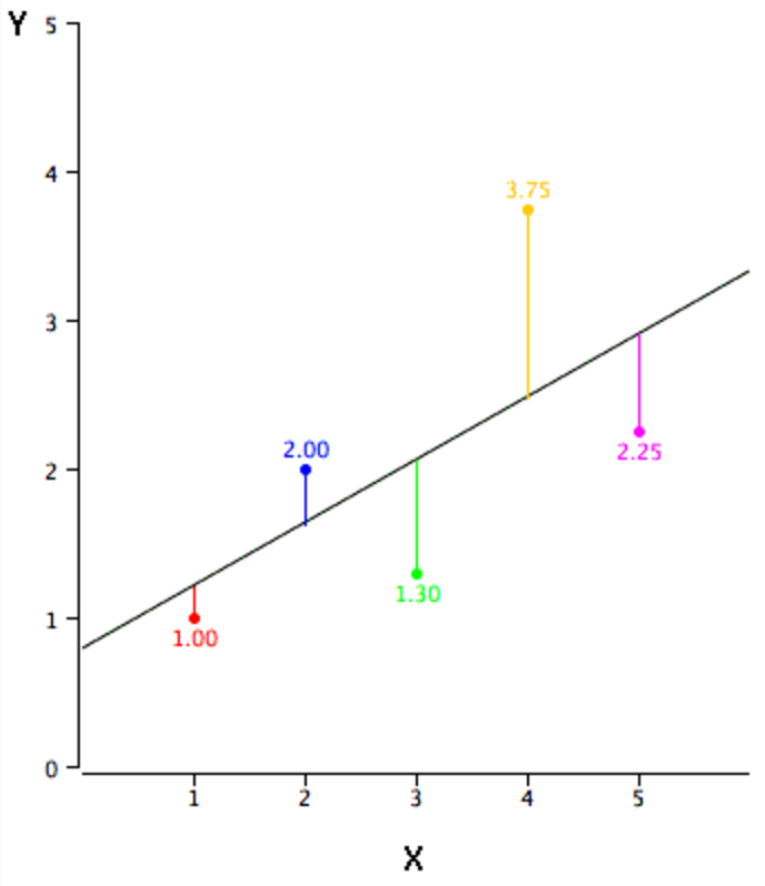
If x = 6, y = ?

Simple Regression -- SGD explanation



The variable we are predicting: *criterion variable* (referred to as Y).
The variable we are basing our predictions: *predictor variable* (referred to as X).

When there is only one predictor variable, the prediction method is called *simple regression*.



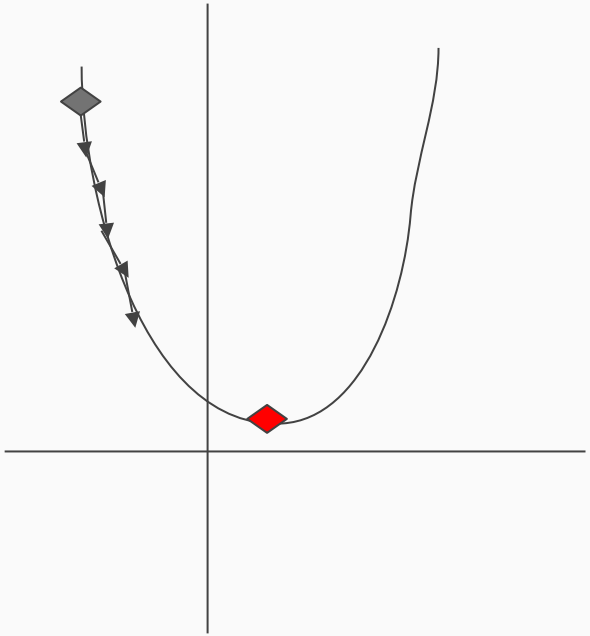
To fine a line $h_\Theta(x) = \Theta_0 + \Theta_1 x$
That the sum of the square errors is minimum
⇒ try to minimize

$$J(\Theta_0, \Theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\Theta_1 x_i + \Theta_0 - y)^2$$

⇒

Cost Function

Stochastic gradient descent: SGD tries to find minima or maxima by iteration.
Used to minimize this Cost Function.



When $\theta_0 = 0$, Cost function will be:

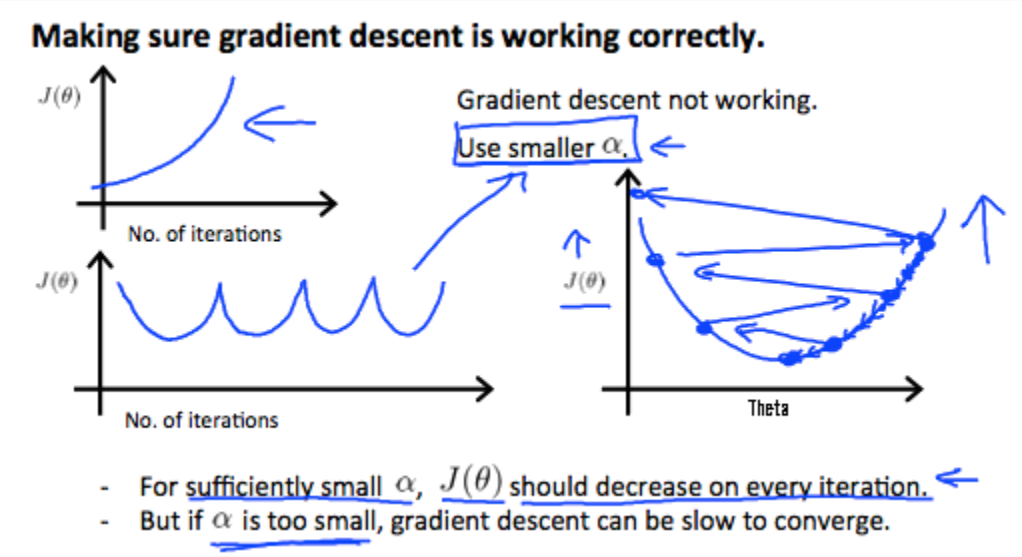
$$J(\Theta_1) = \frac{1}{2m}\sum_{m}^{i=1}(\Theta_1 x_i - y)^2$$

$$\Longrightarrow \quad \frac{\mathrm{d}J(\Theta_1)}{\mathrm{d}\Theta_1} = \frac{\sum r_i x_i}{m}$$

$$step = \alpha\frac{\sum r_i x_i}{m}$$

SGD when α too large

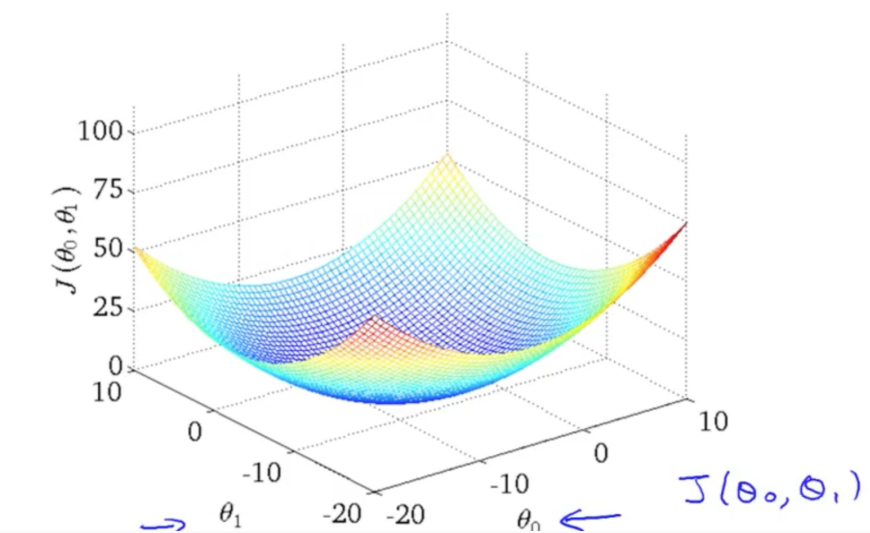
Suppose for one variable

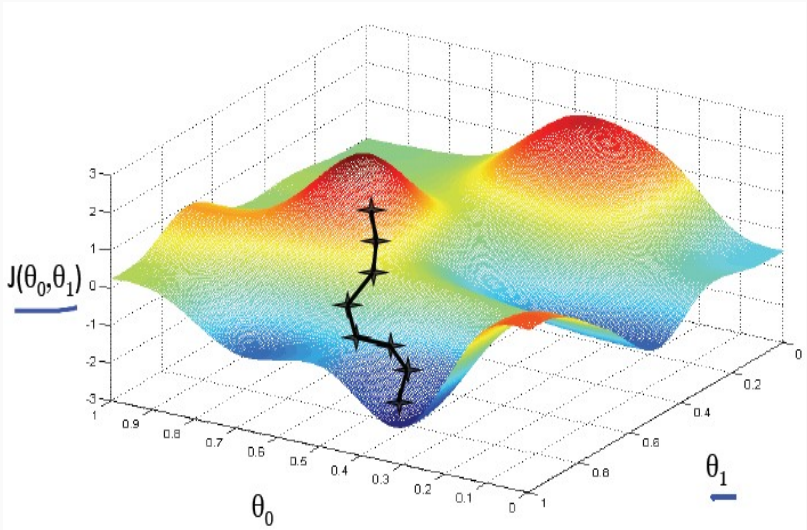


$\theta_0 != 0$

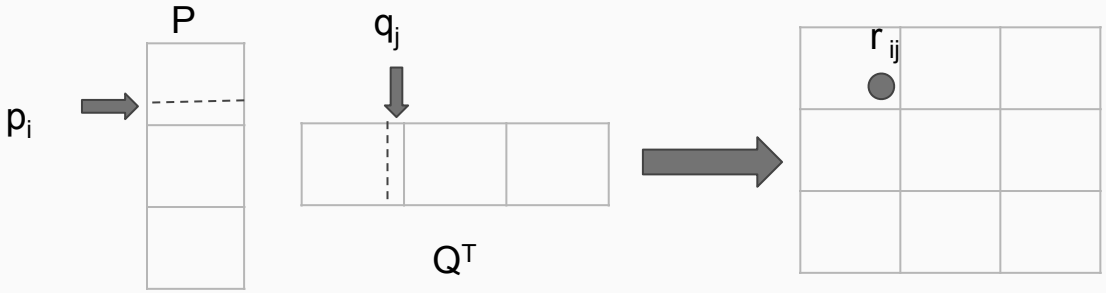
When $\theta_0 != 0$, Cost function will be:

$$J(\Theta_0, \Theta_1) = \frac{1}{2m} \sum_{m}^{i=1} (\Theta_1 x_i + \Theta_0 - y)^2$$





Matrix Factorization - SGD



Stochastic gradient descent: SGD tries to find minima or maxima by iteration.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik}$$

Matrix Factorization - SGD

Having obtained the gradient, we can formulate the rule for both p_{ik} and q_{kj}

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

α is a constant whose value determines the rate of approaching the minimum. Usually we will choose a small value for **α**, say 0.0002. This is because if we make too large a step towards the minimum we may run into the risk of missing the minimum and end up oscillating around the minimum.

Sample implementation

```
"""
@INPUT:
    R     : a matrix to be factorized, dimension N x M
    P     : an initial matrix of dimension N x K
    Q     : an initial matrix of dimension M x K
    K     : the number of latent features
    steps : the maximum number of steps to perform the optimisation
    alpha : the learning rate
@OUTPUT:
    the final matrices P and Q
"""
def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002):
    Q = Q.T
    for step in xrange(steps):
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j])
                    for k in xrange(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k])
        eR = numpy.dot(P,Q)
        e = 0
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
                    if e < 0.001:
            break
    return P, Q.T
```

For every i, j, calculate R"[i] [j],
Adjust P[i , :] and Q[: , j] according to the difference e_{ij}

Number of predictors : K

Matrix Factorization

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

	D1	D2	D3	D4
U1	4.97	2.98	2.18	0.98
U2	3.97	2.40	1.97	0.99
U3	1.02	0.93	5.32	4.93
U4	1.00	0.85	4.59	3.93
U5	1.36	1.07	4.89	4.12

Agenda

- About Matrix Factorization
- **HDFS & MapReduce & Spark**
- Spark Usage Overview
- Prototype your program
- Optimization

Before goes to implementation

Why do we need Spark ?

Spark was invented to do THINGs MapReduce cannot do but still based on MapReduce.

-- Every Spark job will be converted to run on MapR

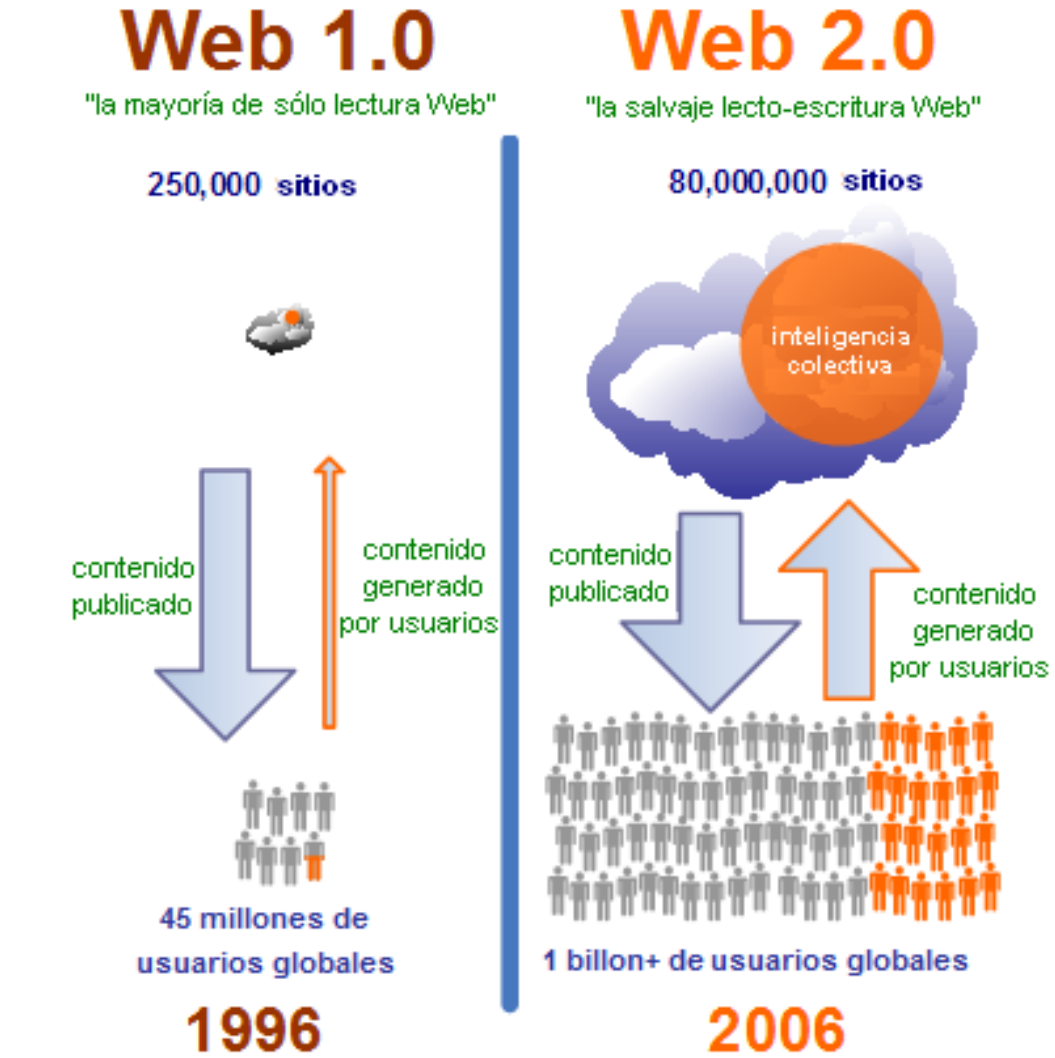
So, what is MapReduce ? What can it do or it cannot do ?

What's the premise of both MapReduce and Spark ?

Let's have a look at some background knowledge..

History of Big Data

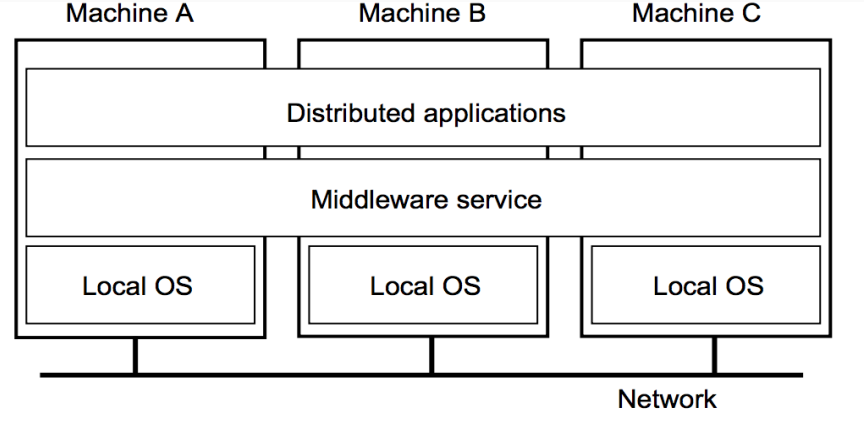
- Data Explosion
  - Web 1.0 to Web 2.0
- Large and growing data requires
  - Huge storage and back up
  - Data analysis, intensive computation
  - Larger system capacity and load balancing.
  - Still fast data accessing



So it comes the Distributed System

A distributed system is a piece of software that ensures that:

A collection of independent computers that appears to its users as a single coherent system



Key Component: Middleware service

1. Connect and coordinate resources
- CPUs – Computation resources on all the machines
- Memory – Faster access data (eg. Spark jobs)
- **Storage – Disk && SSD**
- Network – Data transmission. Some jobs require faster network connection
2. Scalability
3. Distribution transparency
4. Fault tolerance

Hadoop File System

Spark runs on HDFS (today it support variety of FS as underlying infrastructure). It load and output data to HDFS.

So, what is HDFS ?

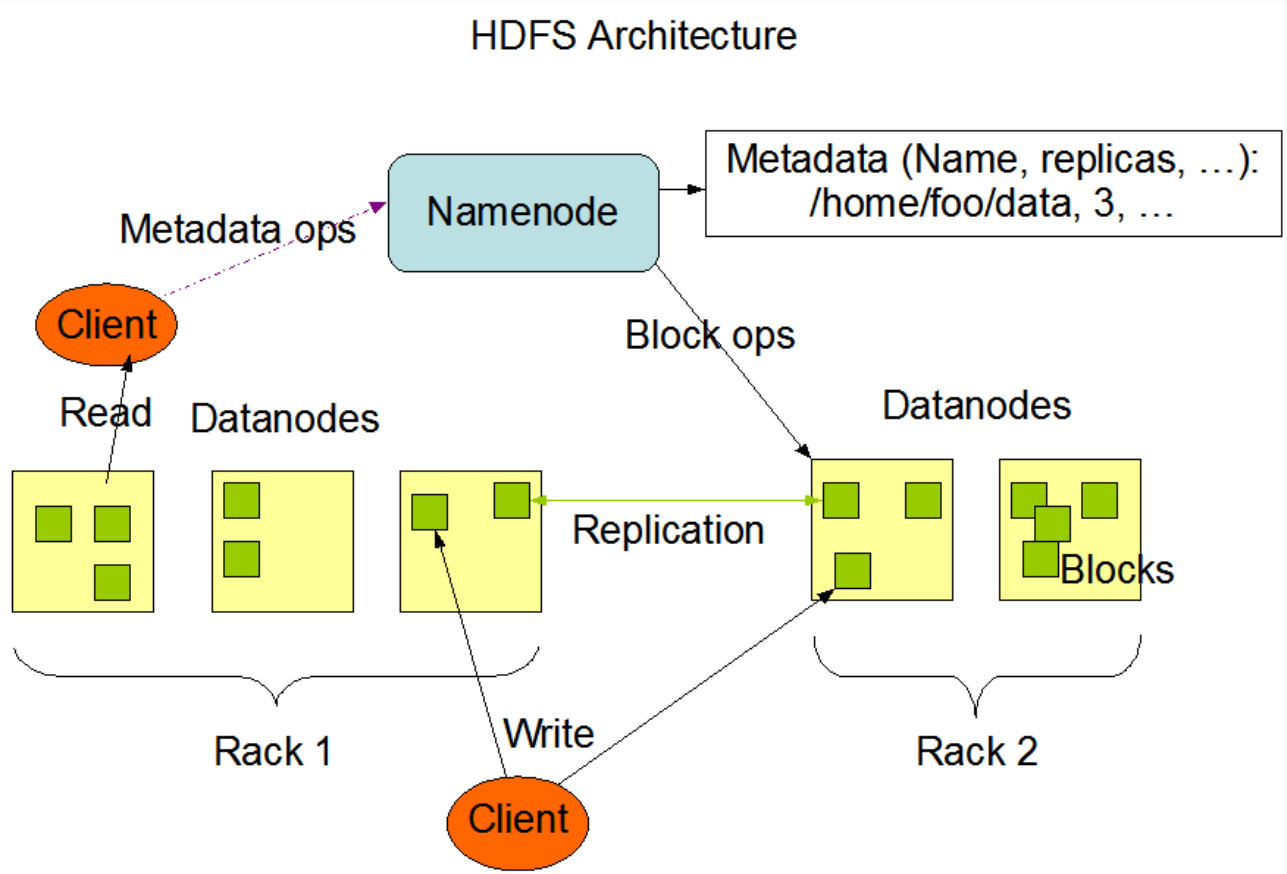
A distributed file system designed to run on commodity hardware.

1. Transparency: As if you are using local FS, create a file with “path/to/file”, file opened. But you have no idea where the file physically sits.

2. Fault tolerant: data corruption. → Replication

3. High throughput → Clients talks directly to Data Node after get address from NameNode

HDFS architecture



Master-Slave

NameNode: All the metadata.

1. User namespace. (accounts)
2. Directories, file address, size.. etc.
3. Replication factor.
4. Responsible for file system namespace operation: open, delete, touch, rename...

DataNode:

1. Files are split into data blocks stored in a set of DataNodes.
2. Responsible for read and write requests from clients.

Hadoop MapReduce

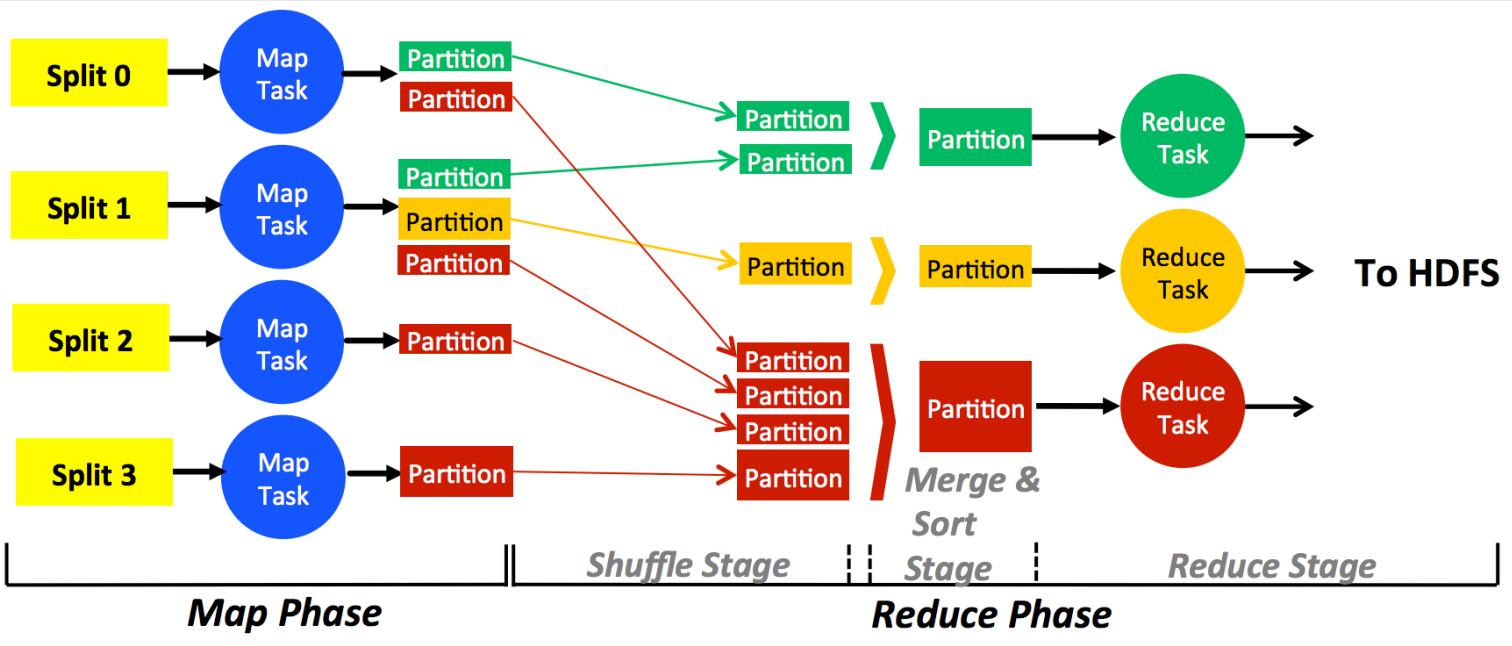
Data parallel framework for processing big data on large commodity hardware.

Each MapReduce processing has two phases. **Mapper** phase and **Reduce** phase.

Transparently tolerance.

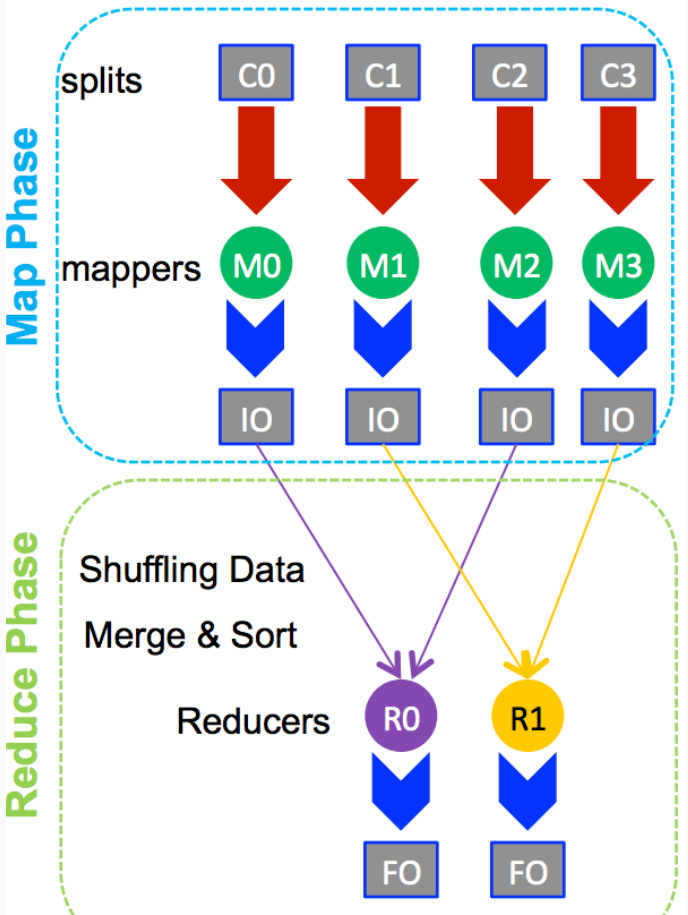
1. Data corruption → rely on HDFS
2. Computation failure → retry

Hadoop MapReduce



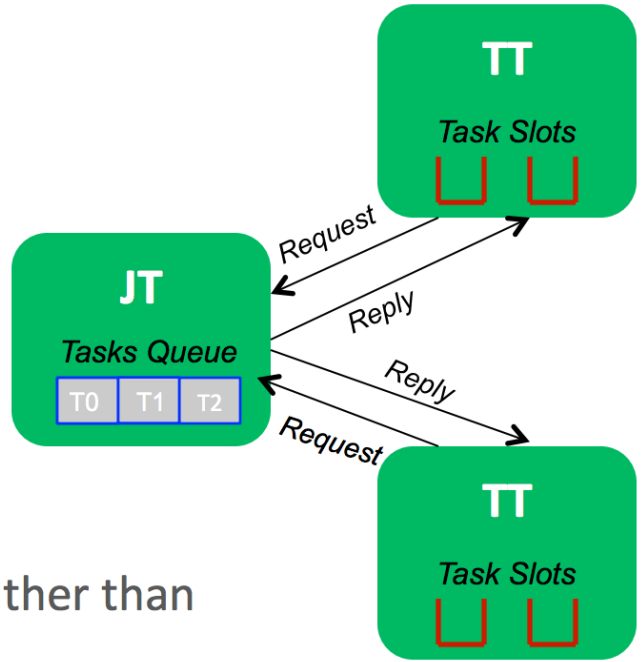
1. Large data is splitted and store in different nodes by HDFS
2. These splits will first be input into Mappers. MapReduce will try to feed splits to local Mappers. [Locality]
3. Output of Mappers are intermediate and will soon been read into second tasks, called Reducers.
4. The process of reading intermediate output into reducers, and in the meantime, merge & sort them all called Shuffle.
5. Each Reducer will get complete data for each key it's responsible for, and generate final output.

Hadoop MapReduce



Task Scheduling in MapR

- MapReduce adopts a *master-slave architecture*
- The master node in MapReduce is referred to as *Job Tracker* (JT)
- Each slave node in MapReduce is referred to as *Task Tracker* (TT)
- MapReduce adopts a *pull scheduling* strategy rather than a *push one*
  - I.e., JT does not push map and reduce tasks to TTs but rather TTs pull them by making requests



Task Scheduling in MapR

- Every TT sends a *heartbeat message* periodically to JT encompassing a request for a map or a reduce task to run

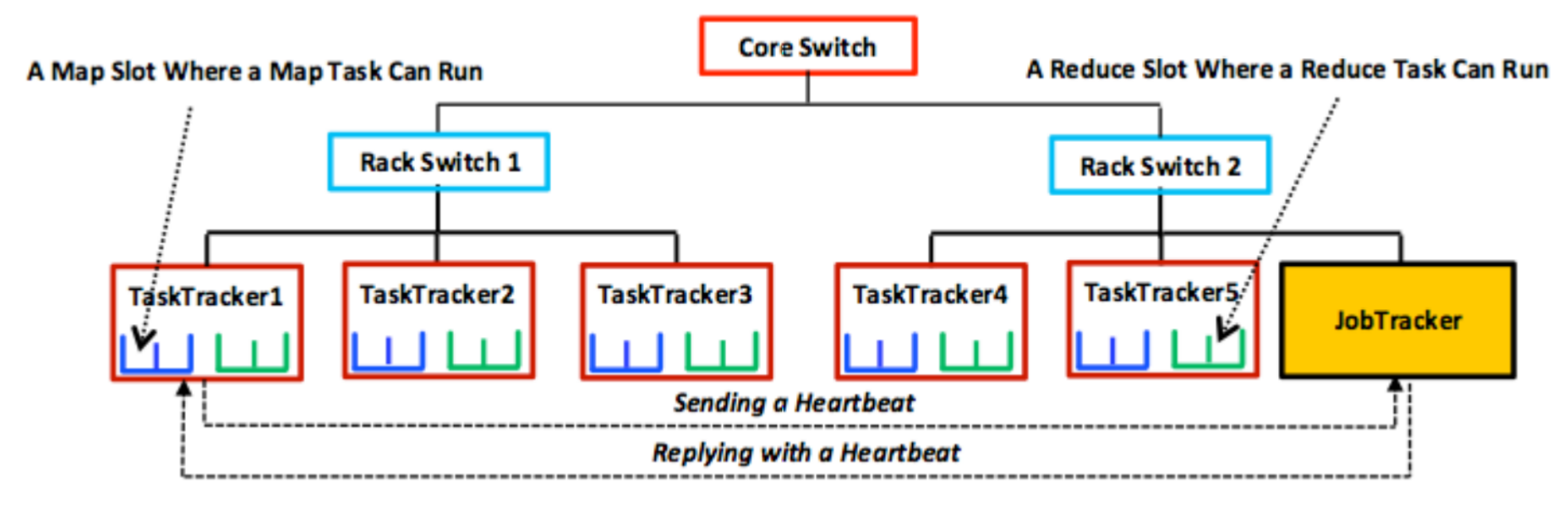
I. Map Task Scheduling:

- JT satisfies requests for map tasks via attempting to schedule mappers in the *vicinity* of their input splits (i.e., it considers locality)

II. Reduce Task Scheduling:

- However, JT simply assigns the next yet-to-run reduce task to a requesting TT regardless of TT's network location and its implied effect on the reducer's shuffle time (i.e., it does not consider locality)

Task Scheduling Network Topology



When Task Tracker polls for a mapper job in its empty Mapper slots, Job Tracker will try to assign him a Mapper that has input local to him.

Fault Tolerance

1. Data corruption: Rely on HDFS to do data replication.

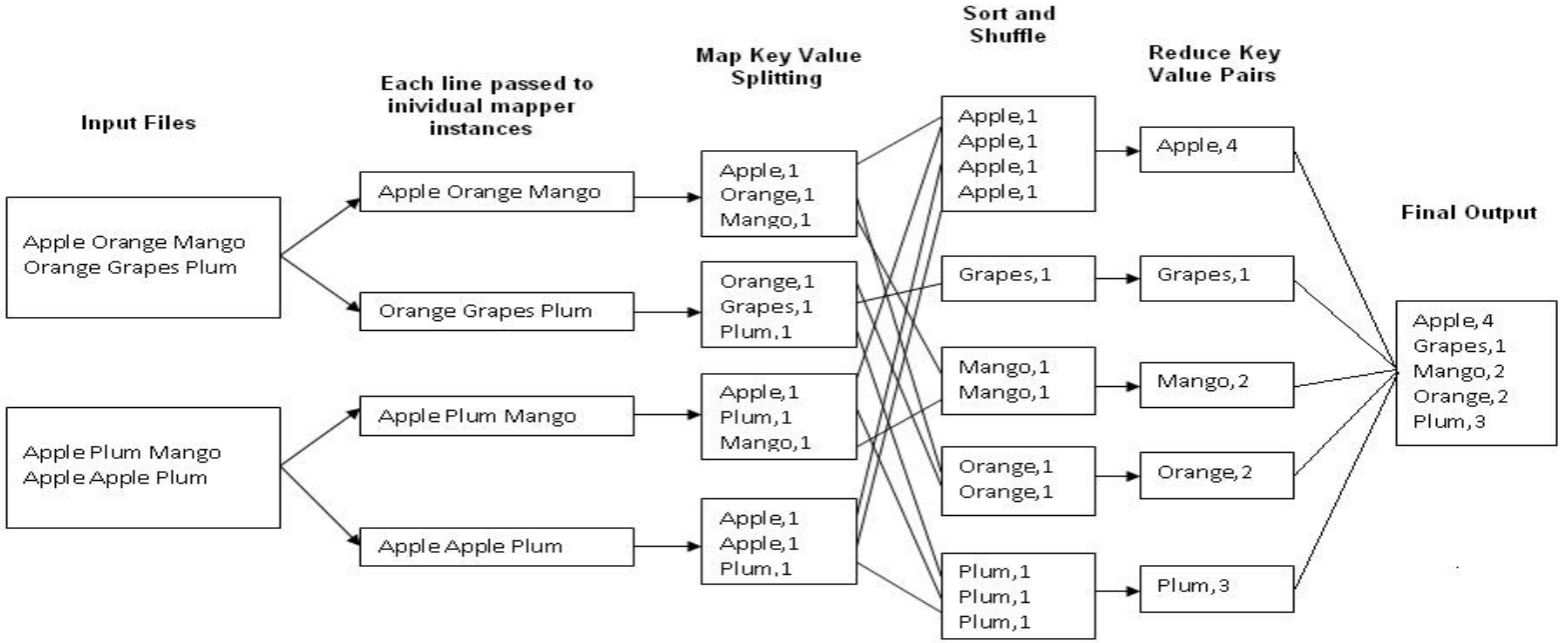
1. Task Tracker will heartbeat Job Tracker. If Job Tracker fails to receive heartbeat from some Task Tracker for a periode of time.
- Re-run all the Mappers including previously ran ones on this machine if job is in Mapper phase.
- Re-run only the on-going Reducer jobs.

Word Count Example

Task: Count the occurrence of each word in the text.

```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

Word Count Example



Apache Spark

Spark was invented to do THINGs MapReduce cannot do but still based on MapReduce.

So, What are the THINGs?

Iterative jobs: Many common machine learning algorithms apply a function repeatedly to the same dataset to optimize a parameter. (SGD)

Interactive jobs: Clients want to issues queries on datasets and immediately see them work. Queries in MR usually take long periode.

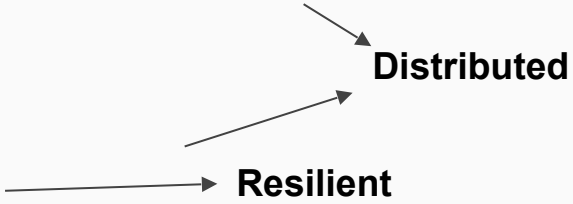
Spark Quick Look - Word Count

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

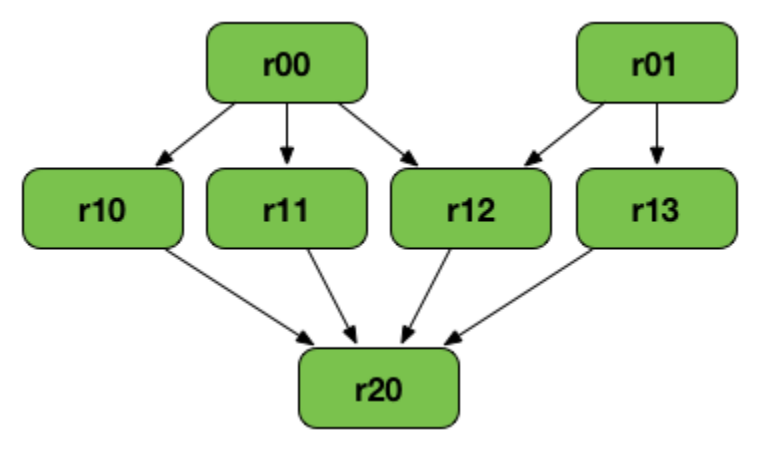
Spark -- RDD

Resilient distributed dataset (RDD). Computation unit.

1. Represents a read-only collection of objects. **Immutable**.
2. **partitioned** across a set of machines
3. Parallel, i.e. process data in parallel. Based on **partitions**
4. RDD can be rebuilt if a partition is lost.
5. In-Memory, i.e. data inside RDD is stored in memory as much (size) and long (time) as possible.
6. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations. But Lazy cache()
7. Lazy evaluated, i.e. the data inside RDD is not available or transformed until an action is executed that triggers the execution.



RDD - Resilience



a handle to an RDD contains enough information to compute the RDD starting from data in reliable storage.

```
val r00 = sc.parallelize(0 to 9)
val r01 = sc.parallelize(0 to 90 by 10)
val r10 = r00 cartesian r01
val r11 = r00.map(n => (n, n))
val r12 = r00 zip r01
val r13 = r01.keyBy(_ / 20)
val r20 = Seq(r11, r12,
r13).foldLeft(r10)(_ union _)
```

A RDD lineage graph is hence a graph of what transformations need to be executed after an **action** has been called.

RDD.toDebugString() can reveal lineage information

Word count lineage transformation

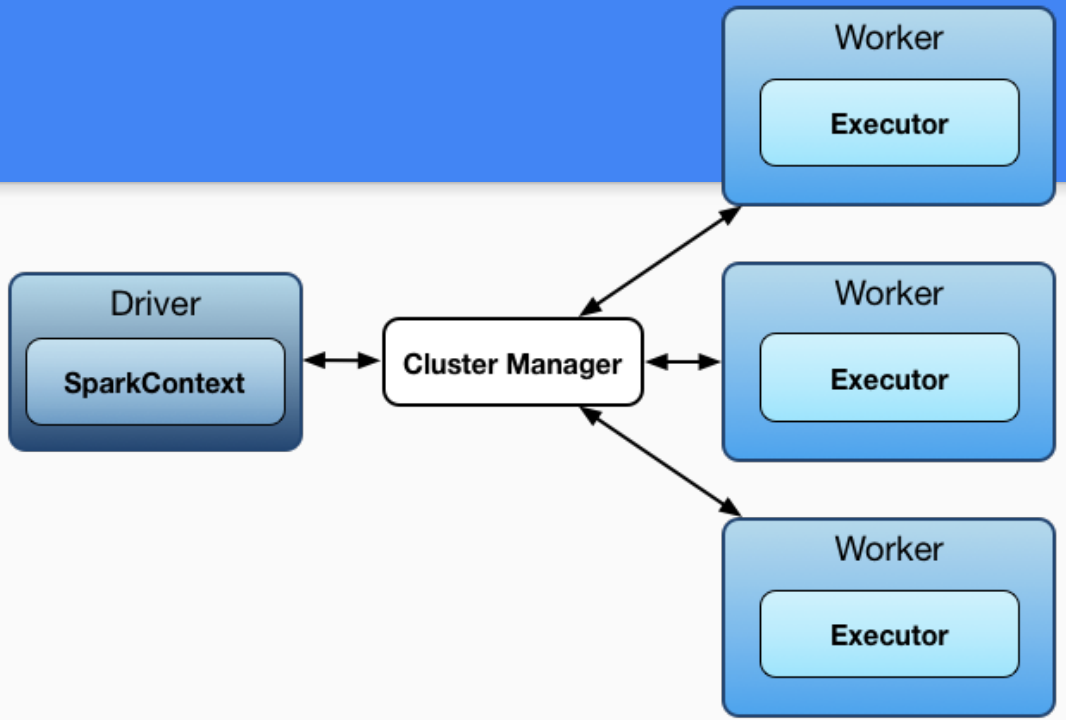
```
scala> val wordCount = sc.textFile("README.md").flatMap(_.split("\\s+")).map((_, 1)).reduceByKey(_ + _)
wordCount: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[21] at reduceByKey at <console>:24

scala> wordCount.toDebugString
res13: String =
(2) ShuffledRDD[21] at reduceByKey at <console>:24 []
 +-(2) MapPartitionsRDD[20] at map at <console>:24 []
    |  MapPartitionsRDD[19] at flatMap at <console>:24 []
    |  README.md MapPartitionsRDD[18] at textFile at <console>:24 []
    |  README.md HadoopRDD[17] at textFile at <console>:24 []
```

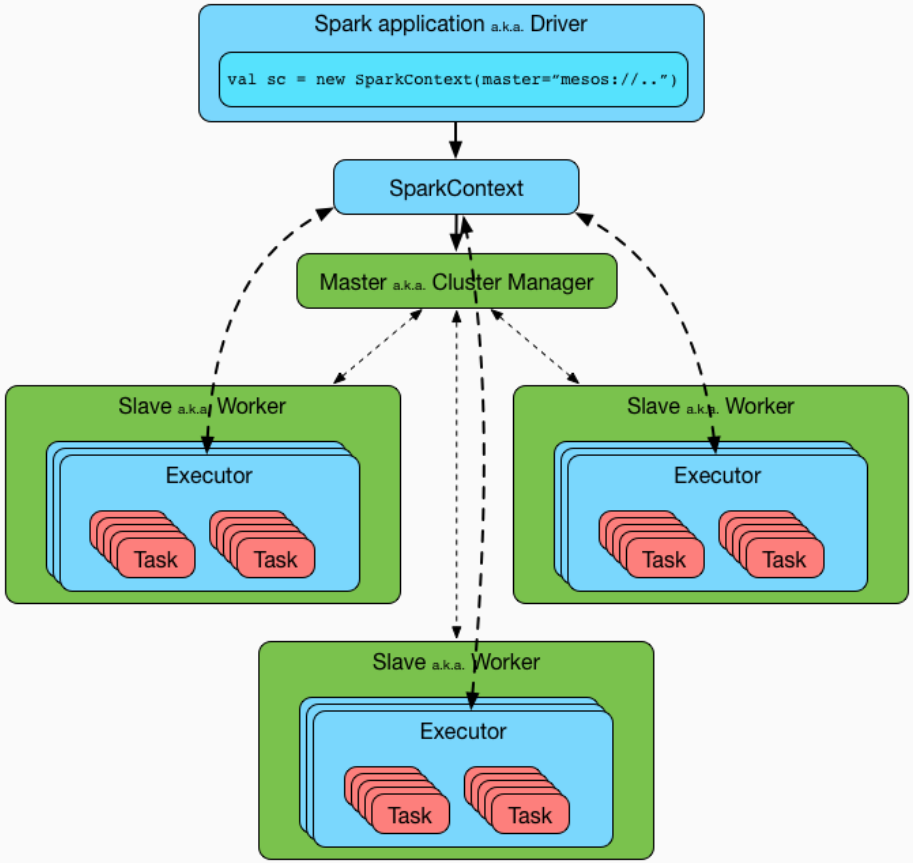
Spark Architecture

1. Spark uses a master/worker architecture

1. a driver that talks to a single coordinator called master that manages workers in which executors run

1. Your Spark program runs in driver





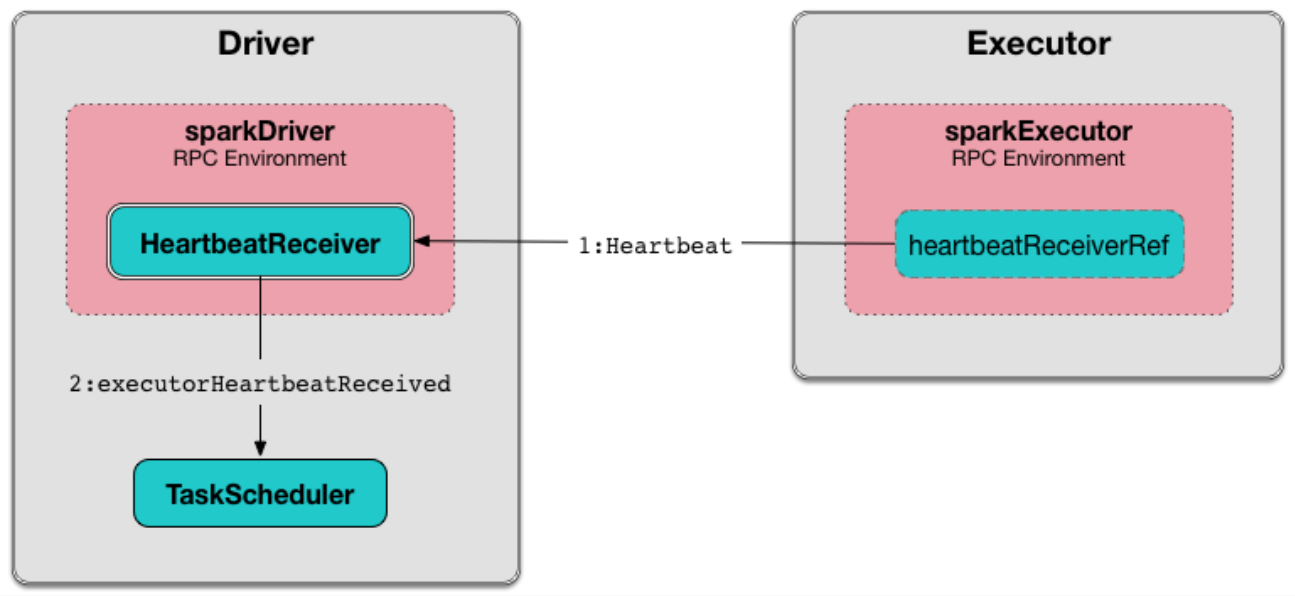
Spark Driver:

1. Where hosts a SparkContext for a Spark application.

1. It splits a Spark application into tasks and schedules them to run on executors.

1. A driver is where the task scheduler lives and spawns tasks across workers.

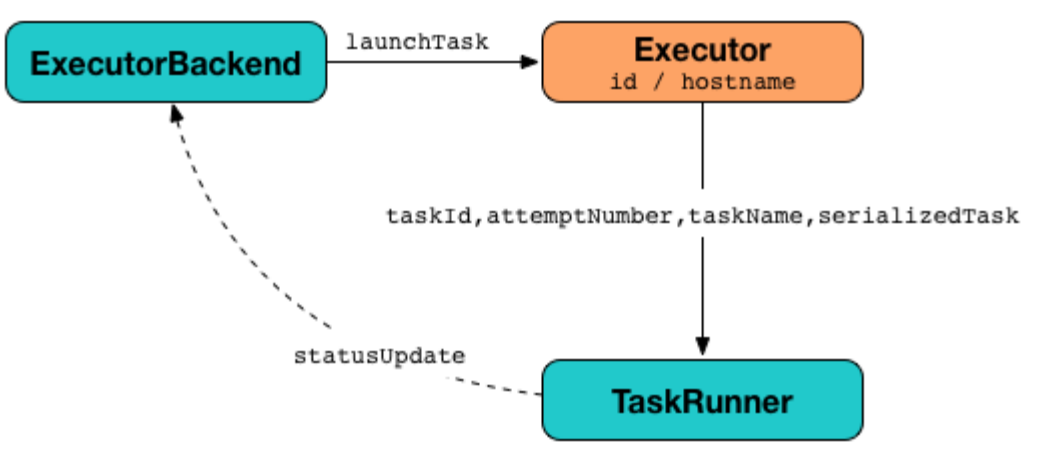
Executor



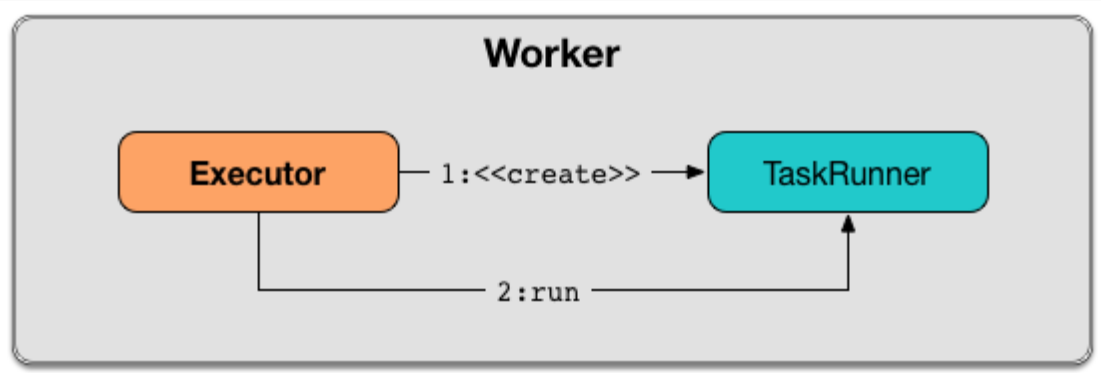
Executor is a distributed agent that is responsible for executing tasks.

1. Executor *typically* runs for the entire lifetime of a Spark application
2. Executors **provide in-memory storage** for RDDs that are cached in Spark applications
3. When an executor starts it first registers with the driver and communicates **directly** (with other executors) to execute tasks.
4. Executors reports **heartbeat** and partial metrics for active tasks the driver. (RPC heart beat)

Task Runner



TaskRunner is a thread of execution that manages a single individual task.



Scheduler

1. DAGScheduler:

After an **action** has been called, SparkContext hands over a **logical** plan to DAGScheduler that it in turn **translates** to a set of stages that are submitted as TaskSets

DAGScheduler stages for each job, keeps track of which RDDs and stage outputs are **materialized(?)**, and finds a minimal schedule to run jobs. It then submits stages to TaskScheduler.

2. TaskScheduler: TaskScheduler is responsible for **submitting** tasks for execution in a Spark application

Agenda

- About Matrix Factorization
- HDFS & MapReduce & Spark
- **Spark Usage Overview**
- Prototype your program
- Optimization

Spark Usage - var

Closures:

- Defines the scope and life cycle of variables and methods when executing code across a cluster.

1. Spark tasks will be executed on remote machines.

1. Closure contains copy of variables and functions, will be computed before Spark tasks, then sent to remote executor.

1. Directly printing and rdd using rdd.foreach() is WRONG! Function “increment_counter” is run on remote machines, thus print things on *stdout* of this remote machine.

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
def increment_counter(x):
    global counter
    counter += x
rdd.foreach(increment_counter)

print("Counter value: ", counter)
```

Spark Usage - var

1. RDDs: *resilient distributed dataset*
   a collection of elements **partitioned** across the nodes of the cluster that can be operated on in parallel.

1. Two types of *shared variables* that can be used in parallel operations.
- *broadcast variables:* keep a **read-only** variable cached on each machine

```
>>> broadcastVar = SparkContext.broadcast([1, 2, 3])
```

- *Accumulators:* variables that are **only "added"** to through an associative and commutative operation and can therefore be efficiently supported in parallel

```
>>> accum = sc.accumulator(0)
>>> SparkContext.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
```

Spark Usage - functions

Spark's API relies heavily on **passing functions** in the driver program to run on the cluster.

1. Lambda expression.

1. Local defs of a function calling into Spark.

1. Lambda expressions, for simple functions that can be written as an expression.

```
def doStuff(self, rdd):
    field = "bittiger"
    return rdd.map(lambda s: field + s)
```

Spark Usage - functions

2. (1) Local defs inside a function.

```
def doStuff (rdd):
    def myFunc(s):
        return “bittiger”+s
    sc = SparkContext(...)
    rdd.map(myFunc)
```

2. (2) class member functions. (requires sending the object that contains that class along with the method)

```
class MyClass(object):
    def func(self, s):
        return “bittiger”+s
    def doStuff(self, rdd):
        return rdd.map(self.func)
```

Spark Usage - Initialize & loading

SparkContext object:

```
conf = SparkConf().setAppName(appName).setMaster(master)
sc = SparkContext(conf=conf)
```

Contains information and methods about how to access the cluster. For example..

1. You want to load data from HDFS. Convert file data into an RDD, with each line as a separate row.

```
>>> distFile = sc.textFile("data.txt")
```

1. You want to load data from memory.

```
>>> data = [1, 2, 3, 4, 5]
>>> distData = sc.parallelize(data)
```

1. You want to broadcast a variable or create an Accumulator. (see previous slides)

Spark Usage - Key-Value pairs

While most Spark operations work on RDDs containing any type of objects, a few special operations are only available on RDDs of key-value pairs.

But actually, we should think of Spark programing as playing with matrix with the first column elements as keys.

```
lines = sc.textFile("data.txt")
pairs = lines.map(lambda s: (s, 1))
counts = pairs.reduceByKey(lambda a, b: a + b)
```

⬅ Load data. Every line is a row in *lines*

⬅ Convert each line to a pair of (line, 1)

⬅ Aggregate using *lines* as keys.

Spark Usage - Transformation & Action

Transformation:
Transformations **create new RDD** from existing RDD like map, reduceByKey we just saw. They are computed **lazily**. We will see lazy evaluations more in details in next part.

Action:
Actions return final results of RDD computations. Actions triggers **execution** using lineage graph to load the data into original RDD, carry out all intermediate transformations and **return** final results to Driver program or write it out to file system.

Spark Usage - Transformation & Action

Transformation:

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).

There are many kinds of Transformation and Actions, please check http://spark.apache.org/docs/latest/programming-guide.html#initializing-spark
For more information.

Spark Usage - Transformation & Action

Action:

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.

Agenda

- About Matrix Factorization
- HDFS & MapReduce & Spark
- Spark Usage Overview
- **Prototype your program**
- Optimization

Let’s do coding.

Preparation

1. Get **data source**. MovieLens. We will be predicting movie ratings.
   Wget http://files.grouplens.org/datasets/movielens/ml-1m.zip
   1 million ratings.

   Wget http://files.grouplens.org/datasets/movielens/ml-10m.zip
   10 million ratings.

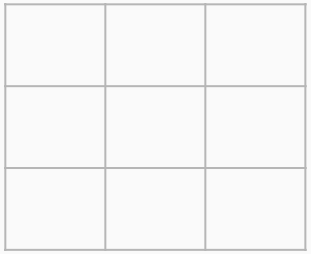
1. Sanitize your data.
   unzip ml-1m.zip ; cut -d':' -f1,3,5 ml-1m/ratings.dat | sed 's/:/,/g' > 1M.csv
   unzip ml-10m.zip ; cut -d':' -f1,3,5 ml-1m/ratings.dat | sed 's/:/,/g' > 10M.csv

1. After cluster is started. Put data into HDFS
   hadoop fs -dfs.block.size=1048576 -put 1M.csv /
   hadoop fs -dfs.block.size=2031616 -put 10M.csv /

How is the data look like

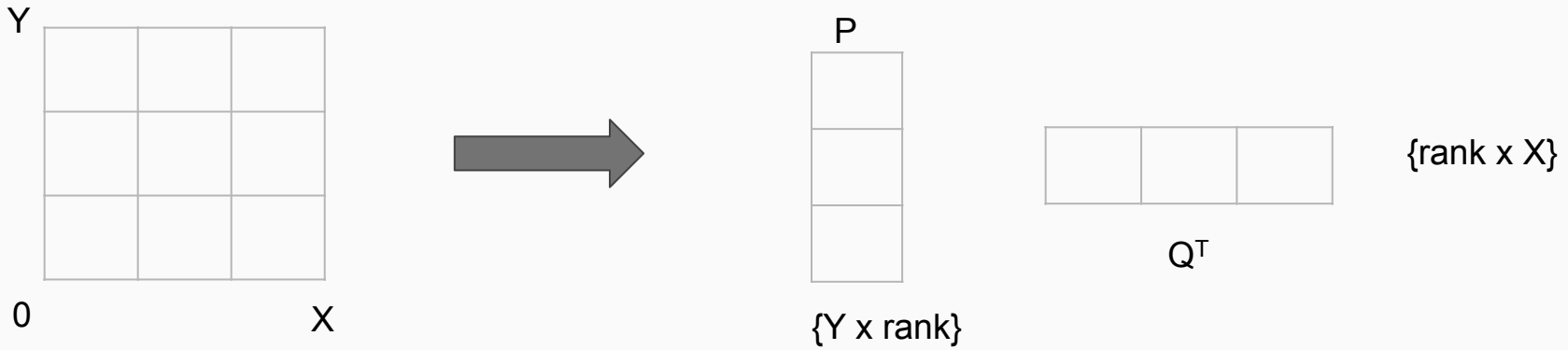
```
1,1193,5
1,661,3
1,914,3
1,3408,4
1,2355,5
1,1197,3
1,1287,5
1,2804,5
1,594,4
1,919,4
1,595,5
1,938,4
1,2398,4
1,2918,4
1,1035,5
1,2791,4
1,2687,3
1,2018,4
1,3105,5
1,2797,4
1,2321,3
```

UserId, MovieId, Rating



Which means, if we put ratings into a matrix,
x axis (or y) will be userid.
y axis (or x) will be movieid.

Matrix Factorization

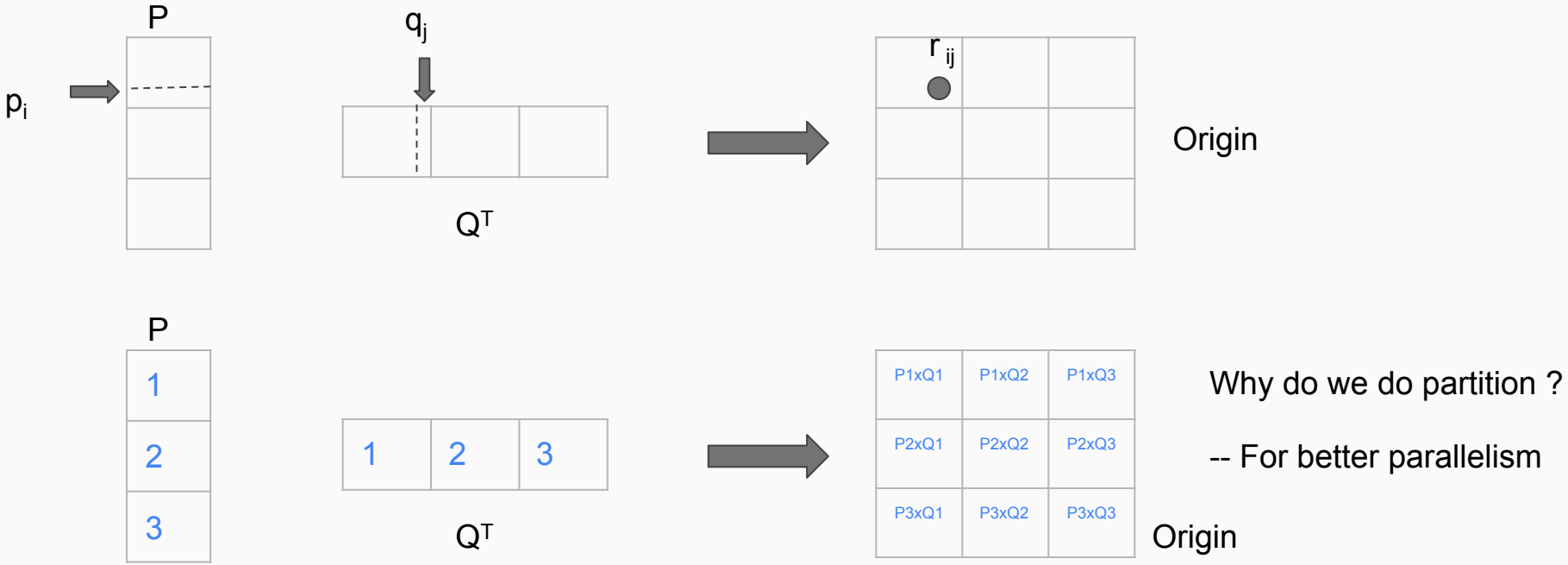


1. First, we load original matrix into memory.

```
matrix_rdd = sc.textFile("path/to/file").persist()
```

Then, Y = max(userid) X = max(movieid) N = 3

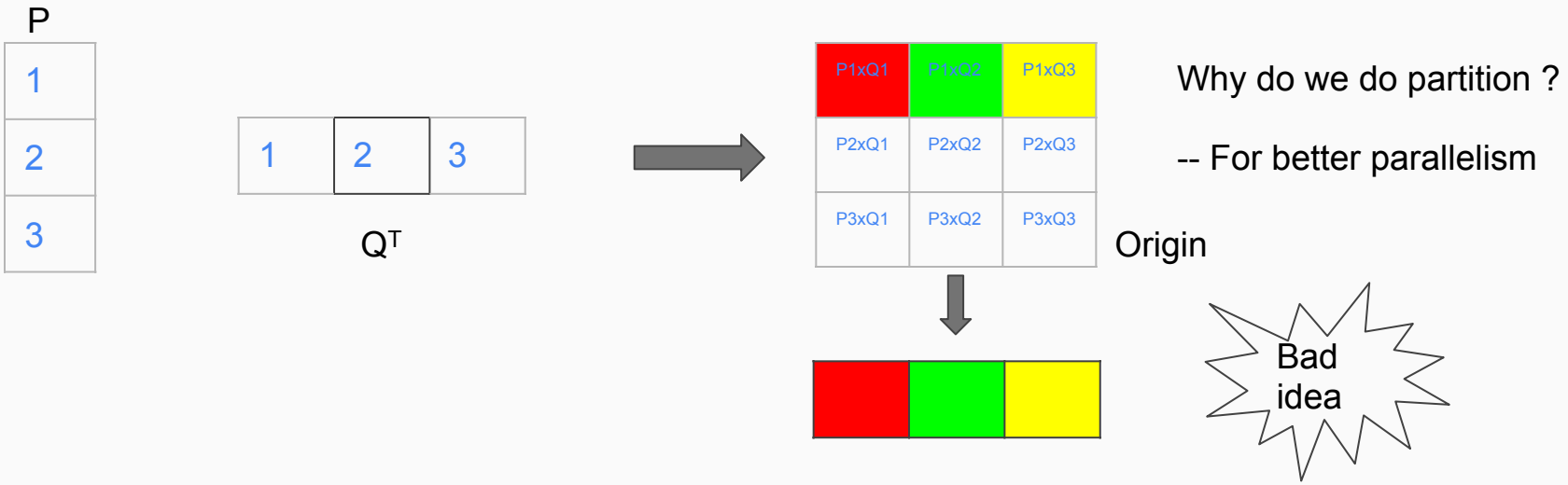
Matrix Factorization



Why do we do partition ?

-- For better parallelism

Note 1: Don’t compute in sequential order



Their computation all repy on P1, thus unable to run in parallel

Matrix Factorization

P

1
2
3

1	2	3

Q^T

P1xQ1	P1xQ2	P1xQ3
P2xQ1	P2xQ2	P2xQ3
P3xQ1	P3xQ2	P3xQ3

Why do we do partition ?

-- For better parallelism

Origin

2. Second, we initialize matrix P and Q, filled with random values.

RDDp =

1	P1
2	P2
3	P3

RDDq =

1	Q1
2	Q2
3	Q3

np.random.rand(len_x, len_y) will generate a random

Len_x * len_y matrix

Note 2: Give ID to each block

we initialize matrix P and Q, filled with random values.

RDDp =

1	P1
2	P2
3	P3

RDDq =

1	Q1
2	Q2
3	Q3

```
RDDq = sc.parallelize(range(N)).map(
  lambda x :
      (x,    np.random.rand( y_block_dim.value, rank))
)
.partitionBy(N)
```

Note 2: Give ID to each block

How to GROUP original data into N * N blocks

```
matrix_rdd = matrix_rdd.map(
   lambda x: (
      (x[0]/x_block_dim.value) + (x[1]/y_block_dim.value)*N, x[0],  x[1], x[2]
   )
).groupBy(
      lambda x:x[0],numPartitions=N*N
 ).cache()
```

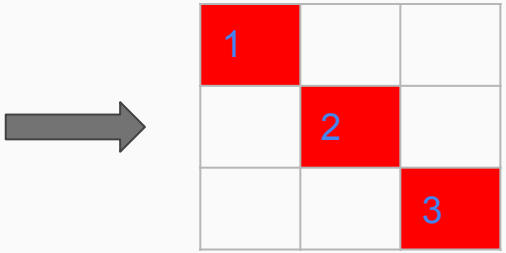
Matrix Factorization

P

1
2
3

1	2	3

Q^T



SelectedOrigin =

1	O1
2	O5
3	O9

RDDp =

1	P1
2	P2
3	P3

RDDq =

1	Q1
2	Q2
3	Q3

RDDp.Join(RDDq).Join(SelectedOrigin) =

1	(P1, Q1),O1
2	(P2, Q2),O5
3	(P3, Q3),O9

Note 3. Blocks need to be joined

Spark doesn’t APIs to work on multiple RDDs together.

All transformations happen on one RDD.

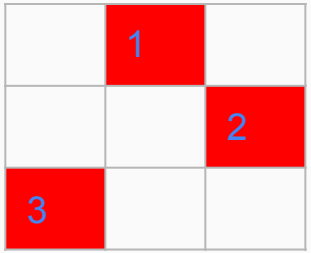
Matrix Factorization

P

1
2
3

3	1	2

Q^T



SelectedOrigin =

1	O2
2	O6
3	Q7

RDDp =

1	P1
2	P2
3	P3

RDDq =

3	Q1
1	Q2
2	Q3

RDDp.Join(RDDq).Join(SelectedOrigin) =

1	(P1, Q2),O2
2	(P2, Q3),O6
3	(P3, Q1),O7

Matrix Factorization

P

1
2
3

2	3	1

Q^T

		1
2		
	3	

SelectedOrigin =

1	O3
2	O4
3	Q8

RDDp =

1	P1
2	P2
3	P3

RDDq =

2	Q1
3	Q2
1	Q3

RDDp.Join(RDDq).Join(SelectedOrigin) =

1	(P1, Q3),O3
2	(P2, Q1),O4
3	(P3, Q2),O8

Note 4 Block ids need to change

Block IDs need to change in order, when it comes to a next iteration, different blocks could be joined together.

For original matrix: delta is the shift value.

```
matrix_rdd.filter(lambda line:
  ((line[0]%N - line[0]/N == delta) or (line[0]/N - line[0]%N == N-delta))
).map(lambda line:
  (line[0]/N, line[1])
).partitionBy(N)
```

0	1	2
3	4	5
6	7	8

Note 4 Block ids need to change

RDDq =

0	Q1
1	Q2
2	Q3

RDDq =

2	Q1
0	Q2
1	Q3

RDDq =

1	Q1
2	Q2
0	Q3

```
W_rdd.map(lambda x: ((x[0] - delta)%N, x[1])).partitionBy(N)
```

Note 5. Partition your RDDs

Spark will load load data chunks into executors based on the unit of partitions.

Otherwise the whole matrix will be loaded for each Spark task, other tasks need to wait until the task who is using conflict data to finish. Parallelism is undermined.

```
matrix_rdd.filter(lambda line:
  ((line[0]%N - line[0]/N == delta) or (line[0]/N - line[0]%N == N-delta))
).map(lambda line:
  (line[0]/N, line[1])
).partitionBy(N)
```

```
RDDq = sc.parallelize(range(N)).map(
  lambda x :
      (x,    np.random.rand( y_block_dim.value, rank))
)
.partitionBy(N)
```

SGD on one block in Spark

```
H_row = H_one[y - H_offeset]
W_row = W_one[x - W_offeset]
diff = val - np.dot(W_row, H_row)
W_gradient = -2 * diff * H_row
W_row -= step_size * W_gradient
H_gradient = -2 * diff * W_row
H_row -= step_size * H_gradient
```

1. Loop through all the dots in the block of the original matrix, x, y denotes their co-ordinates.

1. Find the corresponding lines W_row and H_row in W and H.

1. Calculate the diff.

1. Adjust W_row and H_row

Let’s read through codes to connect them together.

Matrix Factorization

1. As we can see, we need **N** rounds to finish one iteration of the adjustment to P and Q.
2. We can do more iterations to improve accuracy. Reduce standard error as much as it can do.
3. The computation among the three blocks, (like. [(P1, Q1), O1]), it's same as I have talked before.

For more information...

1. Matrix Factorization Techniques for Recommendation Systems
   https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf

1. Collaborative filtering:

   https://en.wikipedia.org/wiki/Collaborative_filteringhttps://en.wikipedia.org/wiki/Collaborative_filtering

1. SGD overview

   http://sebastianruder.com/optimizing-gradient-descent/