Spark

Part 2:
More on transformations and actions.

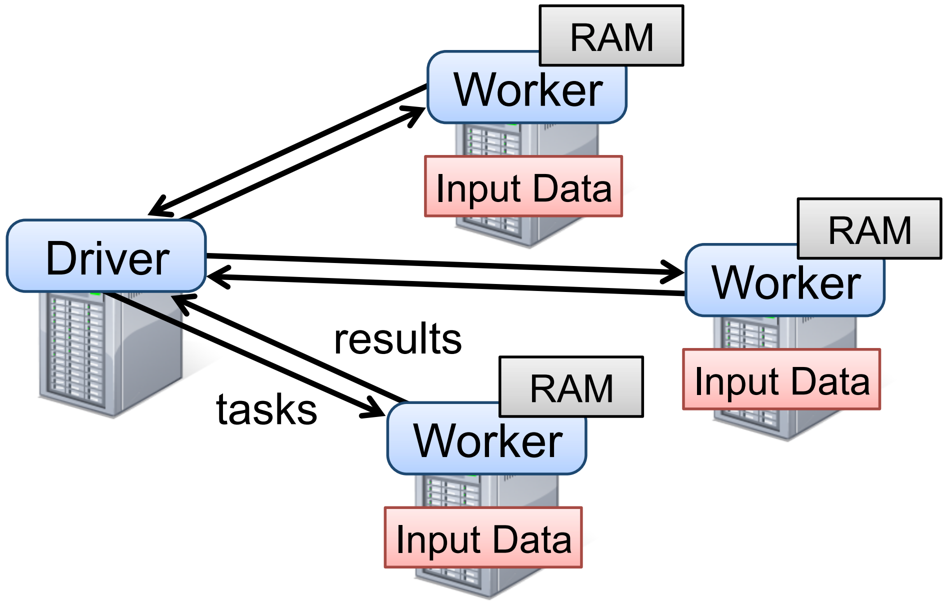
Execution model.

Dario Colazzo

Credits: Amir H. Payberah

Spark programming model

- RDDs : collection of element values distributed over the cluster, mainly in main-memory (RAM)
- Transformations : lazy operators that create new RDDs from RDDs.
- Actions : lunch a computation and return a value to the program driver or write data to the external storage



Transformations

- Already seen
  - map, flatMap, filter, reduceByKey, groupByKey, cogroup
- And made some practice with them.

Set operators

- Union: merges two RDDs and returns a single RDD using bag semantics, i.e., duplicates are not removed.

- Intersection : performs intersection, using set semantics, i.e. duplicates are eliminated.

```
rdd1.union(rdd2)     rdd1.intersection(rdd2)
```

- Attention: RDDs do not need to be homogeneous in order to be unioned/intersected.

- Difference: can be done by means of subtractByKey on RDDs of key-value pairs

```
rdd1.subtractByKey(rdd2)
```

Sampling and repartitioning

- Sample: returns a sample of the input RDD, takes as argument a boolean indicating whether the same element can be re-sampled, the percentage of the sample, and a seed for random number generation

```
>>> rdd = sc.parallelize(range(100), 4)
>>> 6 <= rdd.sample(False, 0.1, 81).count() <= 14
True
```

- Repartition: performs RDD repartitioning by possibly lowering/increasing the number of parts. Attention: shuffle is used. So you may want to use coalesce in case of lowering number of parts.

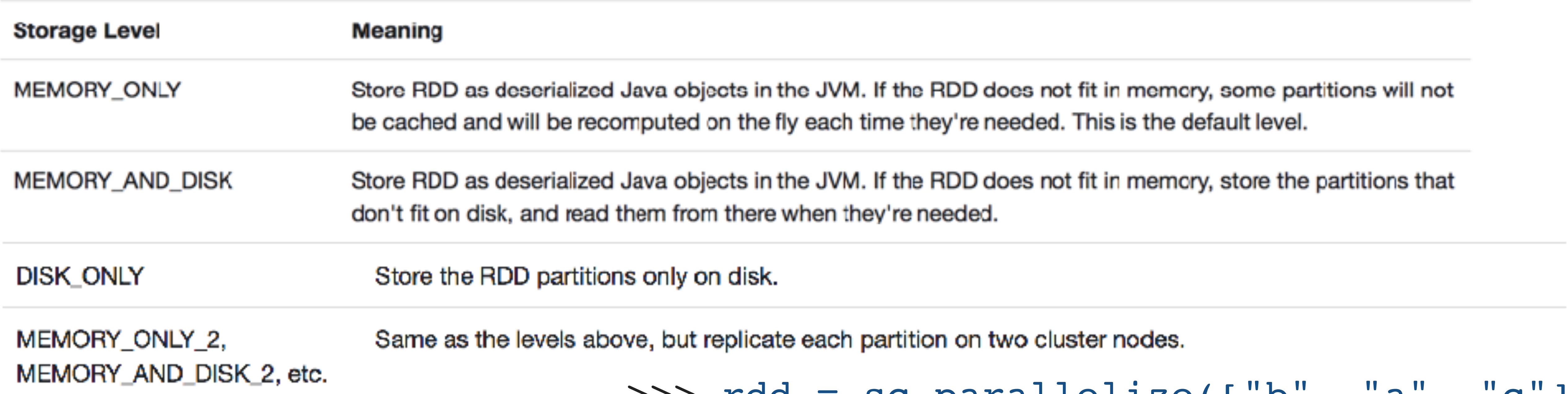
```
>>> rdd.repartition(10)
```

- Glom: return an RDD created by coalescing all elements within each partition into a list (below note the use of take(n), returning the first n elements of an RDD)

```
>>> rdd.glom().take(1)
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24]]
```

Making RDDs persistent

- Crucial for fast iterative and interactive data processing
- They allow for persisting an RDD in a memory level (RAM, DISK): the RDD is computed once and re-used many times.



Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.

```
>>> rdd = sc.parallelize(["b", "a", "c"])
>>> rdd.persist().is_cached
True
```

Actions

- We have already seen collect and take
- We also have count(), with the obvious meaning.
- Reduce can also be performed as an action. The passed binary operation must be associative and commutative: so it is first evaluated locally on each partition and then globally (local aggregation).

```
>>> from operator import add
>>> sc.parallelize([1, 2, 3, 4, 5]).reduce(add)
15
```

- saveAsTextFile is used to save an RDD as a text file

```
>>> rdd1.saveAsTextFile(file: or hdfs: path …)
```

What else?

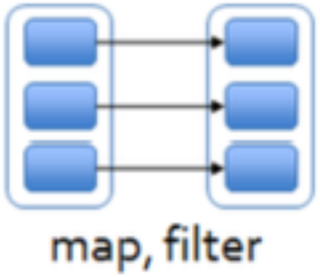
- Many other operations
- Well done documentation:

http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD

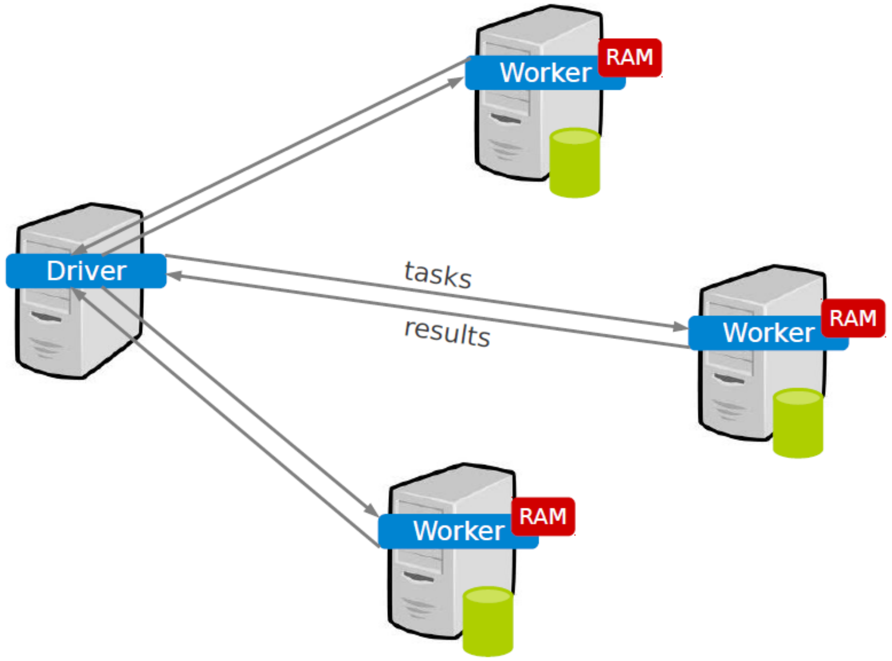
Spark execution model

Process distribution

- A Spark application consists of a driver program that runs the user's main function and executes various parallel tasks on a cluster.
- A task is a transform/action operation performed on a single partition (e.g., the image below shows three map or filter tasks).



- In the case of shell-based use the driver role is played by the shell itself (we will see next class how to write and submit Spark programs to a cluster).

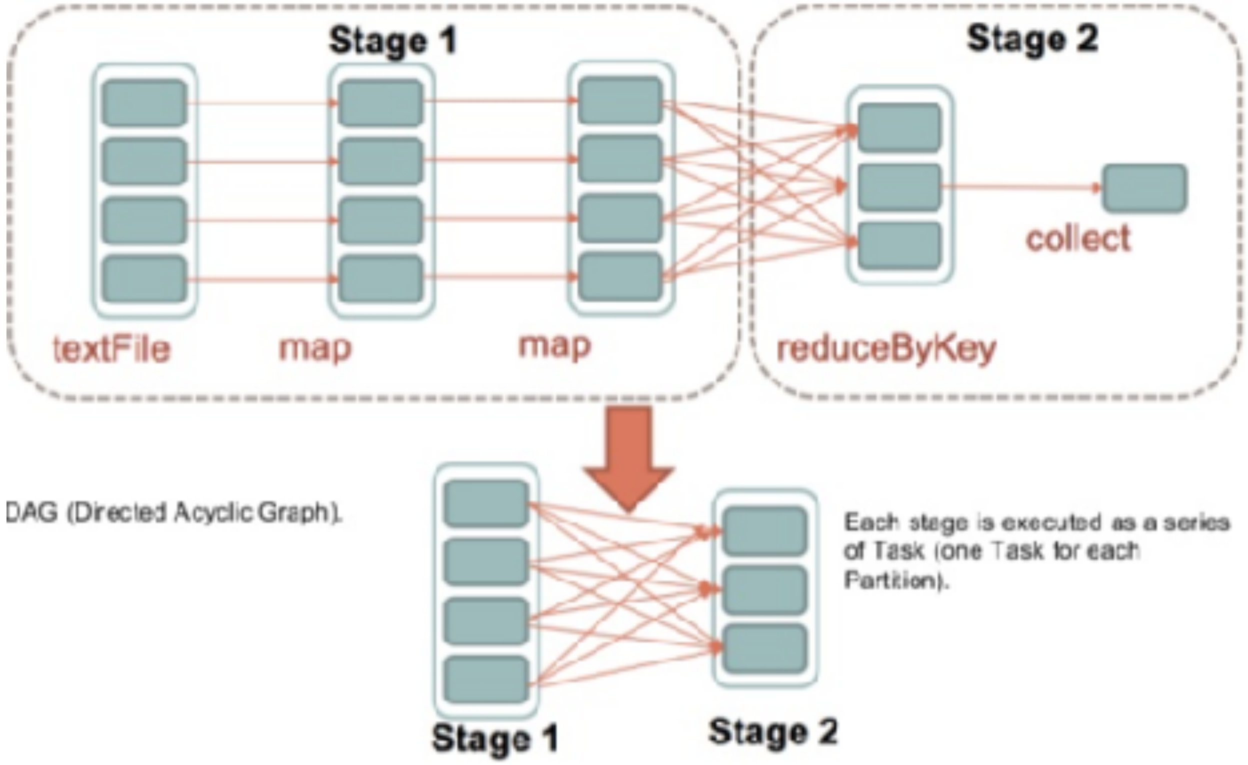


Stages and scheduling

- When a user runs an action on an RDD: the scheduler builds a DAG of stages from the RDD lineage graph.

- A stage contains as many as possible pipelined transformations with narrow dependencies

- Between stages we have wide dependencies : those involving a shuffle operation over the cluster.

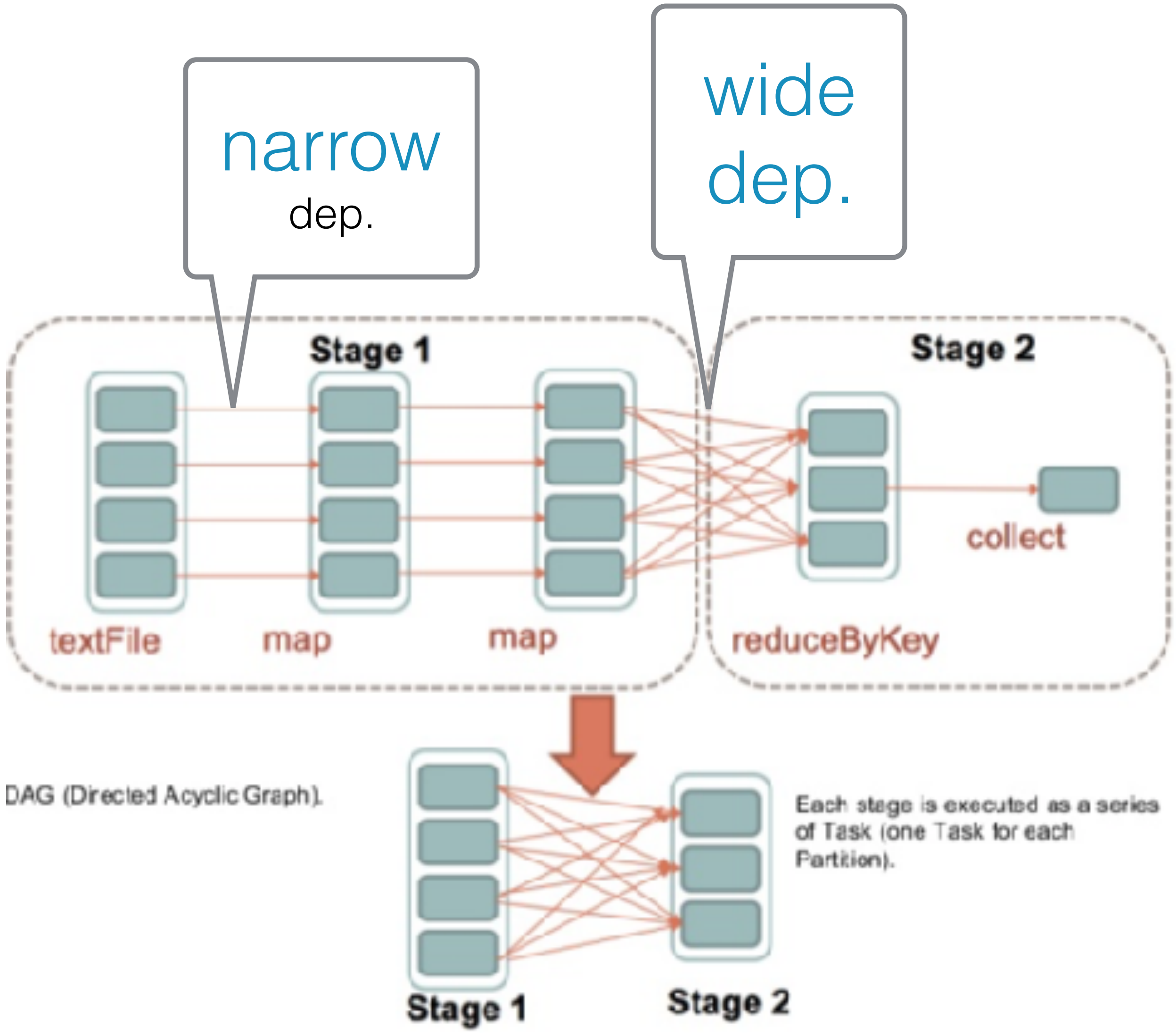


Each stage is executed as a series of Task (one Task for each Partition).

Stages and scheduling

- The scheduler launches tasks to compute missing partitions from each stage until it computes the target RDD.

- Tasks are assigned on machines based on data locality: if a task needs a partition, which is available in the memory of a node, the task is sent to that node.

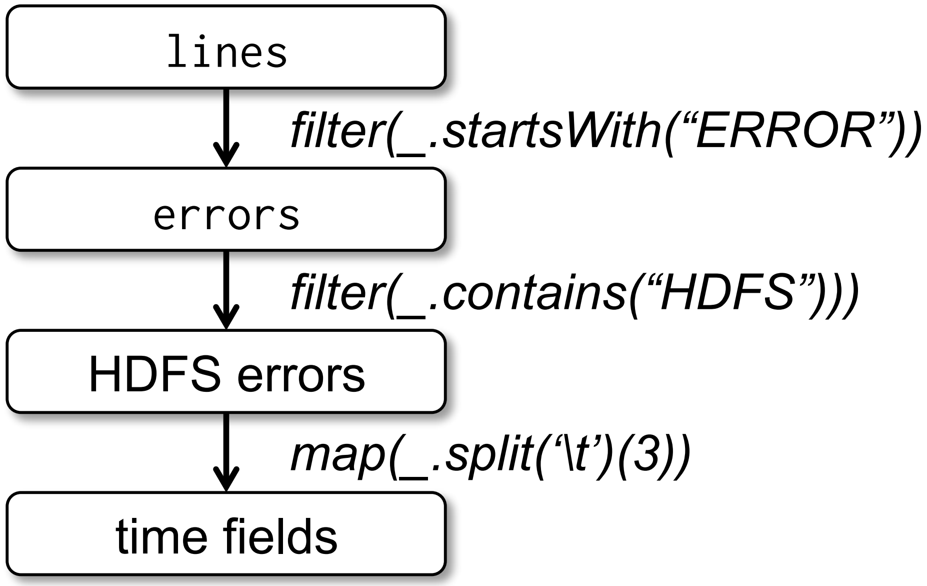


DAG (Directed Acyclic Graph).

Each stage is executed as a series of Task (one Task for each Partition).

RDD fault tolerance

- Logging lineage rather than the actual datakllk
- No replication (unless specified in persist action).
- Recompute only the lost partitions of an RDD.



lines

filter(_.startsWith("ERROR"))

errors

filter(_.contains("HDFS")))

HDFS errors

map(_.split('\t')(3))

time fields