SPARK

@CrazyJvm

What is Spark

- a fast and general-purpose cluster computing system.
- high efficiency
- high level api (Scala, Java, Python)

How to run

- Local
- Standalone
- Mesos
- YARN

and an interactive shell (Scala supported)

About Scala

- JVM based
- Statically typed
- Interoperate with Java (vice-versa)

try in interactive shell !

The most important concept : RDD

RDDs : resilient distributed datasets

internally, each RDD is characterized by five main properties:

* - A list of partitions
* - A function for computing each split
* - A list of dependencies on other RDDs
* - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
* - Optionally, a list of preferred locations to compute each split on (e.g. block locations for an HDFS file)

The most important concept : RDD

- immutable collections of objects spread across a cluster
- build through parallel transformations
- automatically rebuild on failure
- different storage level（memory management）

Over view

- RDDs
- Transformations (Lazy evaluation!!!)
- Action (def runJob[T, U: ClassManifest]

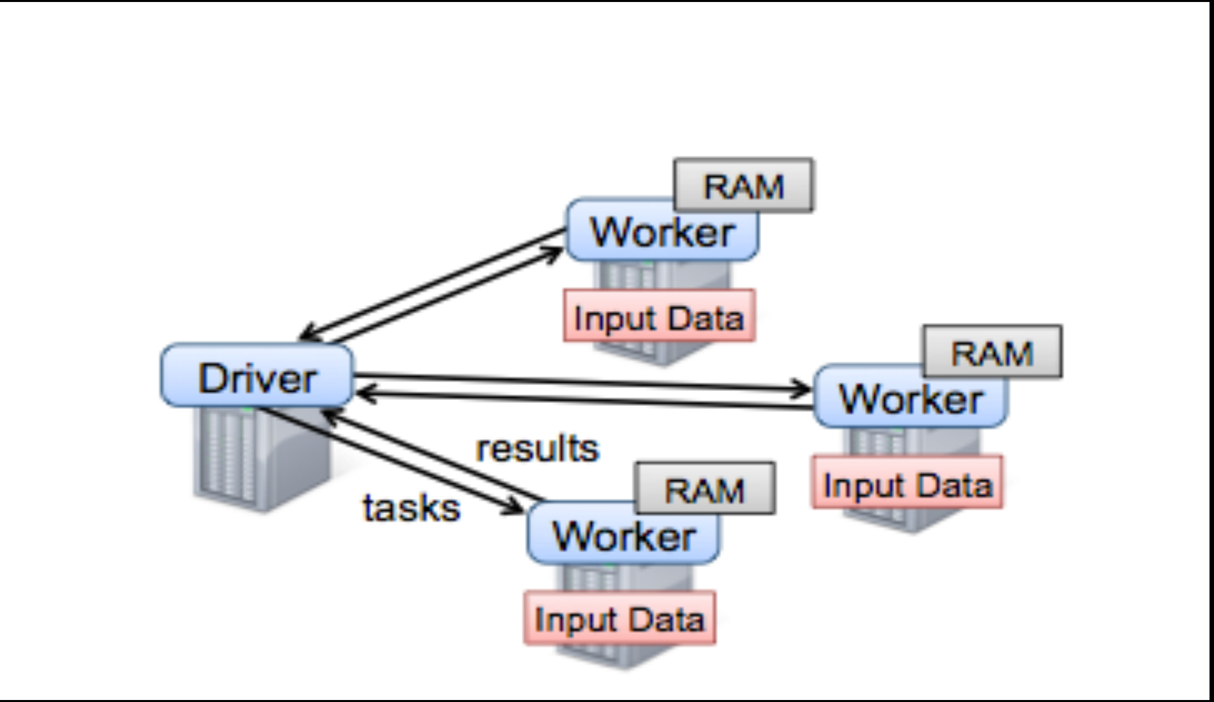
  (rdd: RDD[T],

  func: Iterator[T] => U): Array[U])

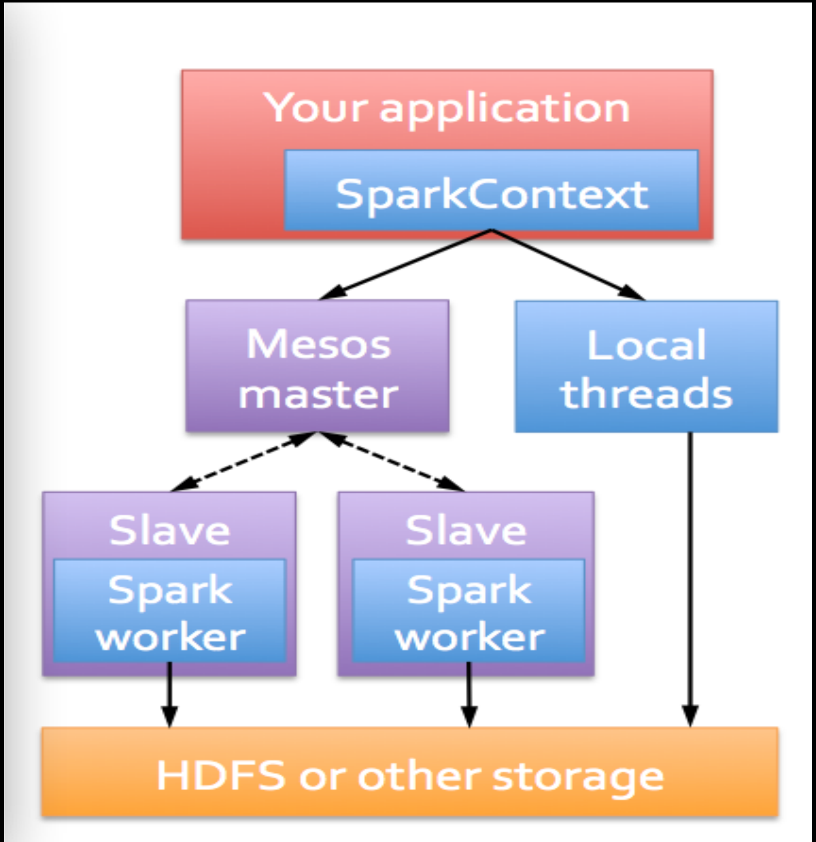
RDD:transformations & actions

Transformations	*map*(f : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(f : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(f : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(f : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(f : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(c : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(p : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(f : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(k : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

Spark runtime



Components



Your application
SparkContext
Mesos master
Local threads
Slave
Spark worker
Slave
Spark worker
HDFS or other storage

Just do it

- interactive shell
- local mode (get local data)
- standalone mode (get data from hdfs)
- programming in IDE(eclipse,idea)

Word Count

```
val text = sc.textFile("README.md")
val wc = text.flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_)
wc.collect
```

notice: reduceByKey is called by implicit conversion

```
implicit def rddToPairRDDFunctions[K: ClassManifest, V: ClassManifest]
        (rdd: RDD[(K, V)]) = new PairRDDFunctions(rdd)
```

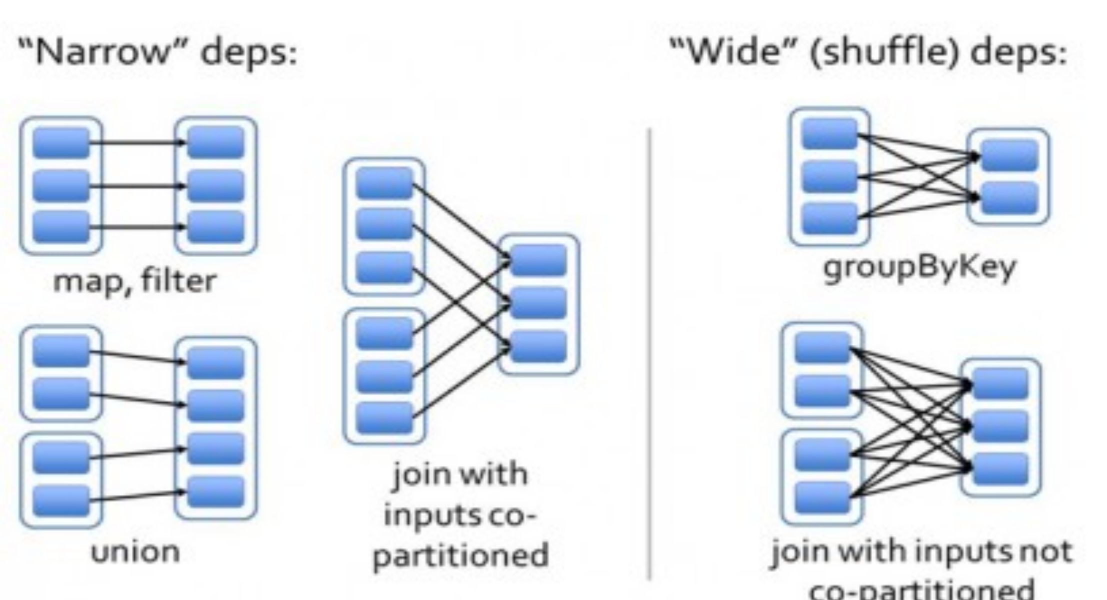
if we let wc.cache, what will happen?

RDD Lineage

- Narrow dependency:each partition of the parent RDD is used by at most one partition of the child RDD.
- Wide dependency: multiple child partitions may depend on a partition of parent RDD.

RDD Lineage

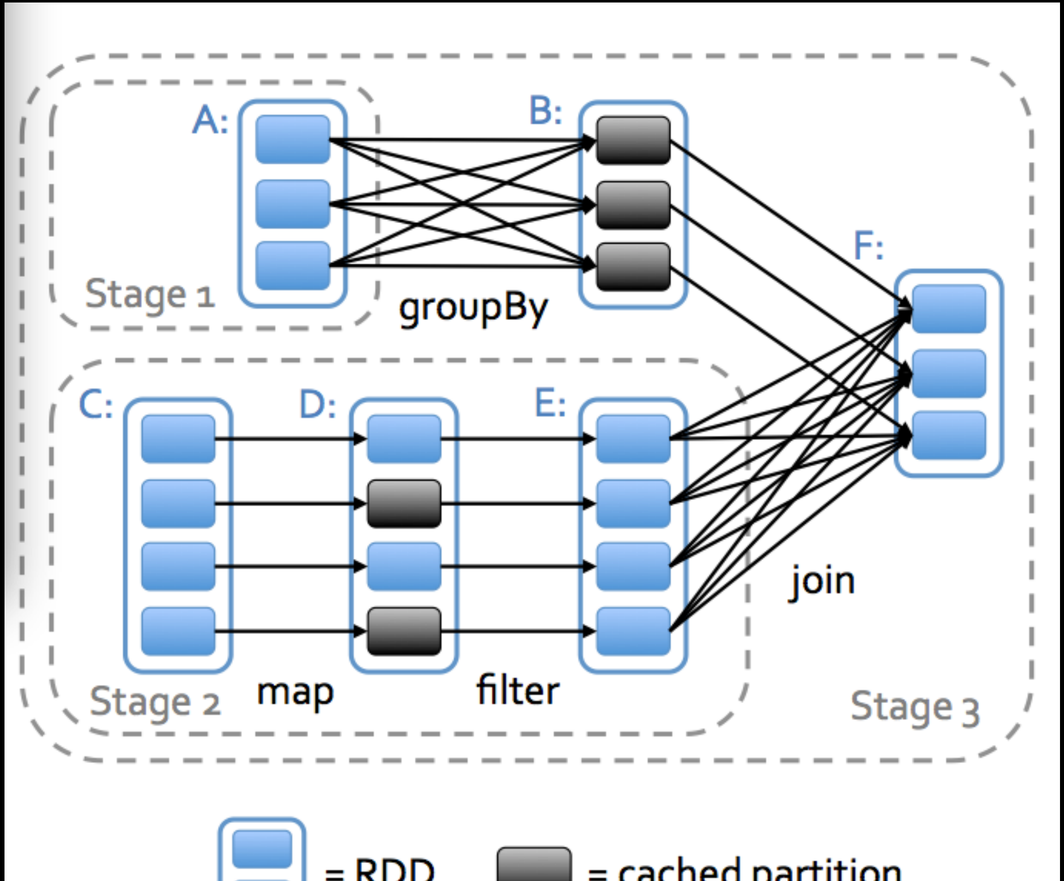
- optimization? -> pipeline
- the importance of co-partitioned



Task scheduler

- run general task graphs
- pipeline functions where possible
- Cache-aware data reuse and locality
- Partitioning-aware to avoid shuffles

Task scheduler



Schedule process

- RDD objects
- DAGScheduler
- TaskScheduler
- Worker

RDD fault tolerance

- recovery by lineage
- checkpoint

Q & A

thanks!