Apache Spark Fundamentals

Data Mining course 2016/2017

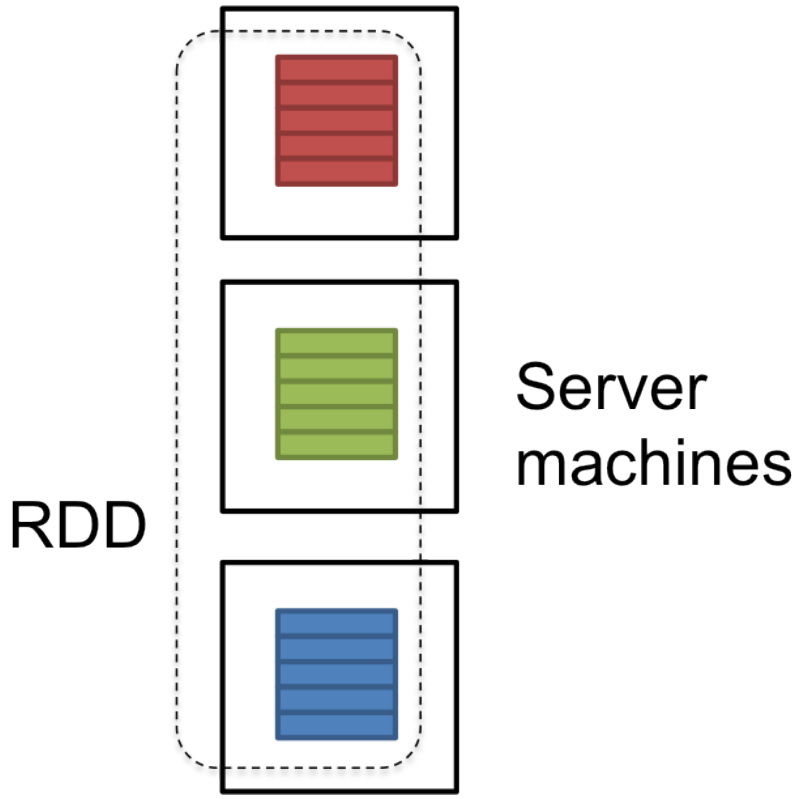
04/04/2017

An overview Spark

- Framework for the implementation of parallel/distributed algorithms
- MapReduce & much more
- Fault tolerant
- Efficient implementation: in-memory caching

RDD: The fundamental abstraction

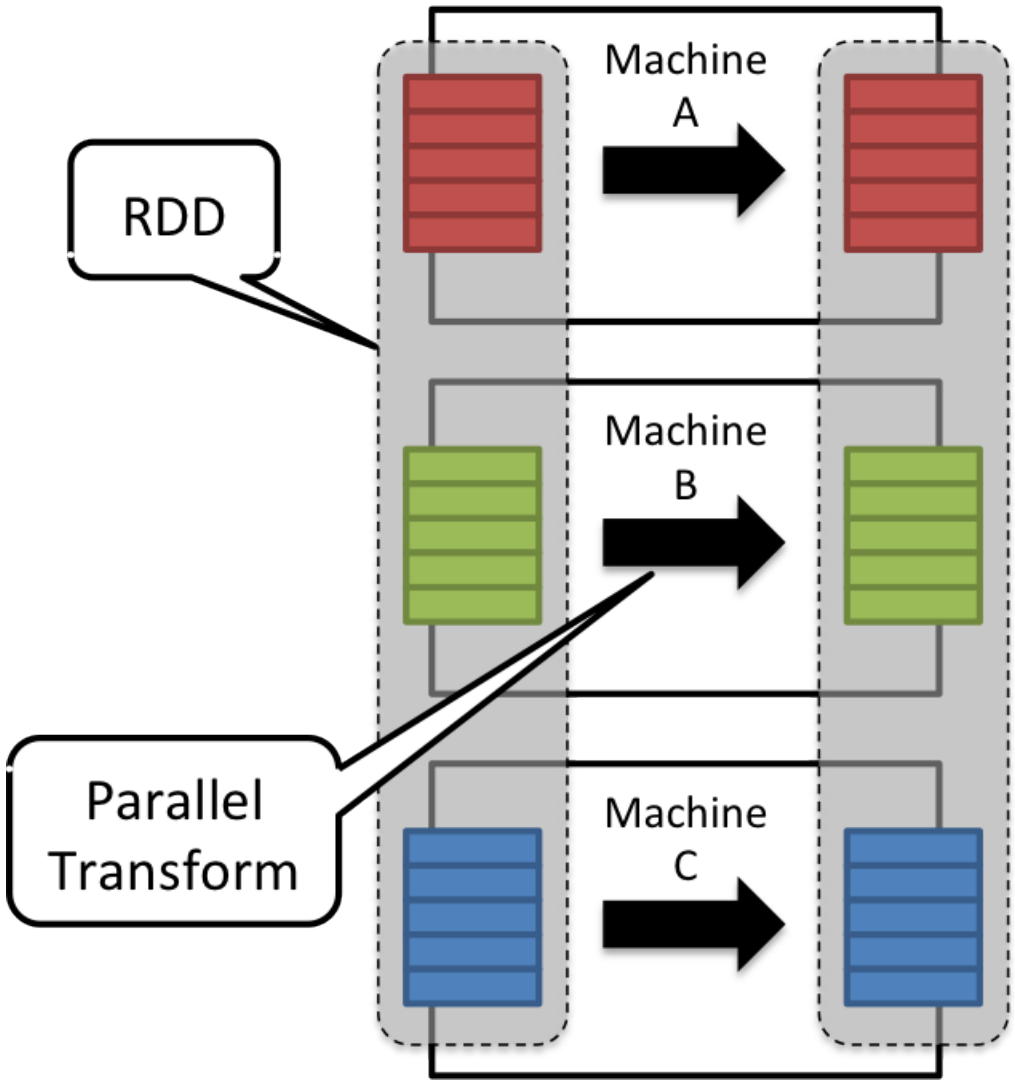
- Collection of elements of *uniform* type
- Possibly distributed across machines
- Java notation
  `JavaRDD<T> data = ...`



Partitioning

- Data is partitioned, possibly across machines
- The number of partitions is usually 2x/3x the number of cores
- Data placement is controlled by using *partitioners*
- A partitioner is a function $p(\cdot)$ that, given an element x of the RDD, returns an integer
- With P partitions, x will be placed in partition $p(x) \mod P$
- The default partitioning is by *hash code*
- Should be tuned when data is imbalanced

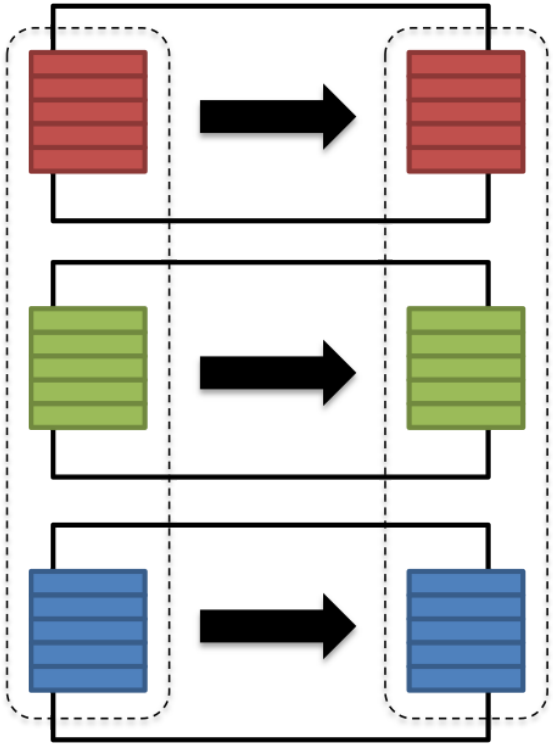
Transformations



Transformations

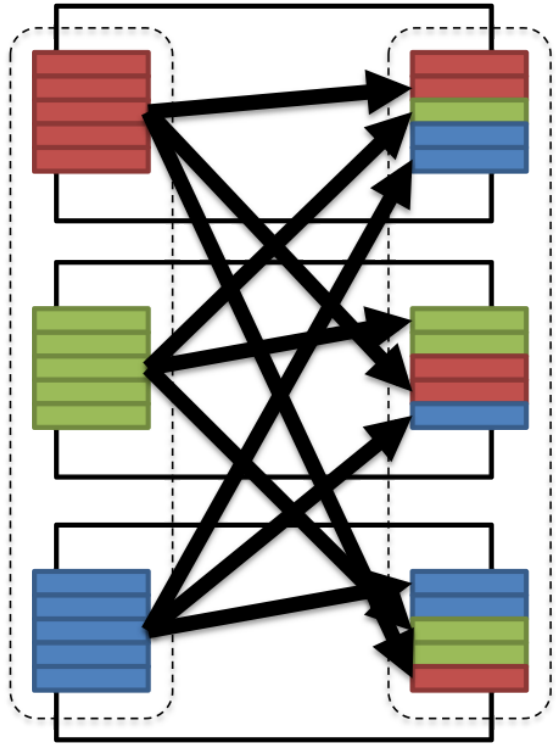
Narrow transformation

- Input and output stays in same partition
- No data movement is needed



Wide transformation

- Input from other partitions are required
- Data shuffling is needed before processing



Transformations

Narrow transformations

- `map`
- `flatmap`
- `filter`
- `union`
- `...`

Wide transformations

- `reduceByKey`
- `groupByKey`
- `sort`
- `join`
- `...`

`http://spark.apache.org/docs/latest/programming-guide.html#transformations`

Transformations, actions, lazyness, and caching

Transformation

- Yields another RDD
- Not computed immediately
- Defines a *lineage*

Action

- Returns a *local* result
- Forces the *evaluation* of all the ancestors in the lineage

Caching

- Store RDDs in memory
- Without caching, the *entire* lineage is evaluated starting from the input at *every* action.

And now, some practical examples!