Spark Essentials: *Transformations*

transformation	*description*
`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)
`sample(`*`withReplacement, fraction, seed`*`)`	sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*
`union(`*`otherDataset`*`)`	return a new dataset that contains the union of the elements in the source dataset and the argument
`distinct([`*`numTasks`*`]))`	return a new dataset that contains the distinct elements of the source dataset

Spark Essentials: *Transformations*

transformation	*description*
`groupByKey([`*`numTasks`***`])`**	when called on a dataset of `(K, V)` pairs, returns a dataset of `(K, Seq[V])` pairs
`reduceByKey(`*`func`***`, [`***`numTasks`***`])`**	when called on a dataset of `(K, V)` pairs, returns a dataset of `(K, V)` pairs where the values for each key are aggregated using the given reduce function
`sortByKey([`*`ascending`***`], [`***`numTasks`***`])`**	when called on a dataset of `(K, V)` pairs where `K` implements `Ordered`, returns a dataset of `(K, V)` pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
`join(`*`otherDataset`***`, [`***`numTasks`***`])`**	when called on datasets of type `(K, V)` and `(K, W)`, returns a dataset of `(K, (V, W))` pairs with all pairs of elements for each key
`cogroup(`*`otherDataset`***`, [`***`numTasks`***`])`**	when called on datasets of type `(K, V)` and `(K, W)`, returns a dataset of `(K, Seq[V], Seq[W])` tuples – also called `groupWith`
`cartesian(`*`otherDataset`***`)`**	when called on datasets of types `T` and `U`, returns a dataset of `(T, U)` pairs (all pairs of elements)

Spark Essentials: *Actions*

action	*description*
`reduce(`*`func`*`)`	aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), and should also 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 – 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
`first()`	return the first element of the dataset – similar to *take(1)*
`take(`*`n`*`)`	return an array with the first *`n`* elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
`takeSample(`*`withReplacement, fraction, seed`*`)`	return an array with a random sample of *`num`* elements of the dataset, with or without replacement, using the given random number generator seed

Spark Essentials: *Actions*

action	*description*
`saveAsTextFile(`*`path`*`)`	write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call `toString` on each element to convert it to a line of text in the file
`saveAsSequenceFile(`*`path`*`)`	write the elements of the dataset as a Hadoop `SequenceFile` in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's `Writable` interface or are implicitly convertible to `Writable` (Spark includes conversions for basic types like `Int`, `Double`, `String`, etc).
`countByKey()`	only available on RDDs of type `(K, V)`. Returns a \`Map\` of `(K, Int)` pairs with the count of each key
`foreach(`*`func`*`)`	run a function *func* on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems