Getting Started with Apache Spark

Download

Apache Spark Developer Cheat Sheet

Transformations (return new RDDs – Lazy)

Where	Function	DStream API	Description
RDD	map(function)	Yes	Return a new distributed dataset formed by passing each element of the source through a function.
RDD	filter(function)	Yes	Return a new dataset formed by selecting those elements of the source on which function returns true.
OrderedRDD Functions	filterByRange(lower, upper)	No	Returns an RDD containing only the elements in the the inclusive range lower to upper.
RDD	flatMap(function)	Yes	Similar to map, but each input item can be mapped to 0 or more output items (so function should return a Seq rather than a single item).
RDD	mapPartitions(function)	Yes	Similar to map, but runs separately on each partition of the RDD.
RDD	mapPartitionsWithIndex(function)	No	Similar to mapPartitions, but also provides function with an integer value representing the index of the partition.
RDD	sample(withReplacement, fraction, seed)	No	Sample a fraction of the data, with or without replacement, using a given random numb generator seed.
			Return a new dataset that

RDD	union(otherDataset)	Yes	contains the union of the elements in the datasets.
RDD	intersection(otherDataset)	No	Return a new RDD that contains the intersection of elements in the datasets.
RDD	distinct([numTasks])	No	Return a new dataset that contains the distinct elements of the source dataset.
PairRDD Functions	groupByKey([numTasks])	Yes	Returns a dataset of (K, Iterable<V>) pairs. Use reduceByKey or aggregateByKey to perform an aggregation (such as a sum or average).
PairRDD Functions	reduceByKey(function, [numTasks])	Yes	Returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function.
PairRDD Functions	aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])	No	Returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type.
OrderedRDD Functions	sortByKey([ascending], [numTasks])	No	Returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.
PairRDD Functions	join(otherDataset, [numTasks])	Yes	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. Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin.
PairRDD Functions	cogroup(otherDataset, [numTasks])	Yes	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples.
RDD	cartesian(otherDataset)	No	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).
RDD	pipe(command, [envVars])	No	Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script.
RDD	coalesce(numPartitions)	No	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
			Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it

RDD	repartition(numPartitions)	Yes	across them. This always shuffles all data over the network.
OrderedRDD Functions	repartitionAndSortWithinPartitions(partitioner)	No	Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. More efficient than calling repartition and then sorting.

Actions (return values – NOT Lazy)

Where	Function	DStream API	Description
RDD	reduce(function)	Yes	Aggregate the elements of the dataset using a function (which takes two arguments and returns one).
RDD	collect()	No	Return all the elements of the dataset as an array at the driver program. Best used on sufficiently small subsets of data.
RDD	count()	Yes	Return the number of elements in the dataset.
RDD	countByValue()	Yes	Return the count of each unique value in this RDD as a local map of (value, count) pairs.
RDD	first()	No	Return the first element of the dataset (similar to take(1)).
RDD	take(n)	No	Return an array with the first n elements of the dataset.
RDD	takeSample(withReplacement, num, [seed])	No	Return an array with a random sample of num elements of the dataset.
RDD	takeOrdered(n, [ordering])	No	Return the first n elements of the RDD using either their natural order or a custom comparator.
RDD	saveAsTextFile(path)	Yes	Write the elements of the dataset as a text. Spark will call toString on each element to convert it to a line of text in the file.
SequenceFileRDD Functions	saveAsSequenceFile(path) (Java and Scala)	No	Write the elements of the dataset as a Hadoop SequenceFile in a given path. For RDDs of key-value pairs that use Hadoop's Writable interface.
RDD	saveAsObjectFile(path) (Java and Scala)	Yes	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using SparkContext.objectFile().
PairRDD Functions	countByKey()	No	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

RDD	foreach(function)	Yes	Run a function on each element of the dataset. This is usually done for side effects such as updating an Accumulator.

Persistence Methods

Where	Function	DStream API	Description
RDD	cache()	Yes	Don't be afraid to call cache on RDDs to avoid unnecessary recomputation. NOTE: This is the same as persist(MEMORY_ONLY).
RDD	persist([Storage Level])	Yes	Persist this RDD with the default storage level.
RDD	unpersist()	No	Mark the RDD as non-persistent, and remove its blocks from memory and disk.
RDD	checkpoint()	Yes	Save to a file inside the checkpoint directory and all references to its parent RDDs will be removed.

Additional Transformation and Actions

Where	Function	Description
SparkContext	doubleRDDToDoubleRDDFunctions	Extra functions available on RDDs of Doubles
SparkContext	numericRDDToDoubleRDDFunctions	Extra functions available on RDDs of Doubles
SparkContext	rddToPairRDDFunctions	Extra functions available on RDDs of (key, value) pairs
SparkContext	hadoopFile()	Get an RDD for a Hadoop file with an arbitrary InputFormat
SparkContext	hadoopRDD()	Get an RDD for a Hadoop file with an arbitrary InputFormat
SparkContext	makeRDD()	Distribute a local Scala collection to form an RDD
SparkContext	parallelize()	Distribute a local Scala collection to form an RDD
SparkContext	textFile()	Read a text file from a file system URI
SparkContext	wholeTextFiles()	Read a directory of text files from a file system URI

Extended RDDs w/ Custom Transformations and Actions

RDD Name	Description
CoGroupedRDD	A RDD that cogroups its parents. For each key k in parent RDDs, the resulting RDD contains a tuple with the list of values for that key.
EdgeRDD	Storing the edges in columnar format on each partition for performance. It may additionally store the vertex attributes associated with each edge.
JdbcRDD	An RDD that executes an SQL query on a JDBC connection and reads results. For usage example, see test case JdbcRDDSuite.
ShuffledRDD	The resulting RDD from a shuffle.
VertexRDD	Ensures that there is only one entry for each vertex and by pre-indexing the entries for fast, efficient joins.

Streaming Transformations

Where	Function	Description
DStream	window(windowLength, slideInterval)	Return a new DStream which is computed based on windowed batches of the source DStream.
DStream	countByWindow(windowLength, slideInterval)	Return a sliding window count of elements in the stream.
DStream	reduceByWindow(function, windowLength, slideInterval)	Return a new single-element stream, created by aggregating elements in the stream over a sliding interval using function.
PairDStream Functions	reduceByKeyAndWindow(function, windowLength, slideInterval, [numTasks])	Returns a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function over batches in a sliding window.
PairDStream Functions	reduceByKeyAndWindow(function, invFunc, windowLength, slideInterval, [numTasks])	A more efficient version of the above reduceByKeyAndWindow(). Only applicable to those reduce functions which have a corresponding "inverse reduce" function. Checkpointing must be enabled for using this operation.
DStream	countByValueAndWindow(windowLength, slideInterval, [numTasks])	Returns a new DStream of (K, Long) pairs where the value of each key is its frequency within a sliding window.
DStream	transform(function)	The transform operation (along with its variations like transformWith) allows arbitrary RDD-to-RDD functions to be applied on a Dstream.
PairDStream Functions	updateStateByKey(function)	The updateStateByKey operation allows you to maintain arbitrary state while continuously updating it with new information.

RDD Persistence

Storage Level	Meaning
MEMORY_ONLY (default level)	Store RDD as deserialized Java objects. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly when needed.
MEMORY_AND_DISK	Store RDD as deserialized Java objects. If the RDD does not fit in memory, store the partitions that don't fit on disk, and load them when they're needed.
MEMORY_ONLY_SER	Store RDD as serialized Java objects. Generally more space-efficient than deserialized objects, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time 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.

Shared Data

Broadcast Variables Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

Language	Create, Evaluate
Scala	val broadcastVar = sc.broadcast(Array(1, 2, 3))
	broadcastVar.value
Java	Broadcast<int[]> broadcastVar = sc.broadcast(new int[] {1, 2, 3});
	broadcastVar.value();
Python	broadcastVar = sc.broadcast([1, 2, 3])
	broadcastVar.value

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

Language	Create, Add, Evaluate
Scala	val accum = sc.accumulator(0, My Accumulator)
	sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)
	accum.value

Java	Accumulator<Integer> accum = sc.accumulator(0);
	sc.parallelize(Arrays.asList(1, 2, 3, 4)).foreach(x -> accum.add(x))
	accum.value();
Python	accum = sc.accumulator(0)

MLlib Reference

Topic	**Description**
Data types	Vectors, points, matrices.
Basic Statistics	Summary, correlations, sampling, testing and random data.
Classification and regression	Includes SVMs, decision trees, naïve Bayes, etc...
Collaborative filtering	Commonly used for recommender systems.
Clustering	Clustering is an unsupervised learning approach.
Dimensionality reduction	Dimensionality reduction is the process of reducing the number of variables under consideration.
Feature extraction and transformation	Used in selecting a subset of relevant features (variables, predictors) for use in model construction.
Frequent pattern mining	Mining is usually among the first steps to analyze a large-scale dataset.
Optimization	Different optimization methods can have different convergence guarantees.
PMML model export	MLlib supports model export to Predictive Model Markup Language.

Other References

- Launching Jobs
- SQL and DataFrames Programming Guide
- GraphX Programming Guide
- SparkR Programming Guide

Previous Next