Week 3 NOTES

1. PartitionBy should always be followed by persist() otherwise we will alwaye end up re-partitionining

2. When working with pair RDDs favour mapValues over map, as map could change the Key, which could affect the partitionining massively

3. HashPartitioner / RangePartitioner

4. rdd.toDegubString is useful, shuffledRDD[...] may show that shuffling is planned, also check out "spark web UI"

5. reduceByKey to reduce shuffling (use this instead of groupByKey)

6. partition and persist datasets before joining can reduce shuffling

7. narrow dependency 1:1 with parent RDD, no shuffling

8. wide dependency 1:m with parent RDD, shuffling

9. cache after groupByKey, may aid the narrow/wide dependency / shuffling

10. rdd.dependencies may aid in showing shuffles

Week 4 NOTES

1. Dataframes are untyped

2. df.createOrReplaceTempView("people")

3. spark.sql("select \* from people")

4. Manually creating dataframe schema (slide 2 - Spark SQl)

5. Array[T] : ArrayType(element, containsNull)

6. Map[K,V] : MapType(keyType, valuetype, valueContainsNull)

7. case class : StructType(List[Scructfields])

8. import org.apache.spark.sql.types.\_

import spark.implicits.\_

9. df.show(), df.printSchema(),

10. Agg (Dataframes (1)) slides, 19:30 mark

11. groupBy followed by agg (typical)

12. $"columnName" spark.implicits.\_

($"age" > 12)

13. df.Select("...")

.Where("...")

.OrderBy("...")

14. column = agg($"age")

Dataframes(2)

- drop (drop alls)

- dropAll (drops all rows with ALL nulls)

- drop(seq) name the cols to drop

- fill

- replace

- all joins covered by generic join(...) method

DataSets

- head.schema.printTreeString()

- typesafe mix and match dataframe + rdd ops

- dataFrame.toDS

- import spark.implicits.\_

- TypedColumn = $"price".as[Double]

- agg has extra function that can be passed to it, such as "xxx.agg(avg($"price".as[Double]))"

- noReduceByKey but can be done via groupByKey + mapGroups

DONT USE mapGroups UNLESS YOU HAVE TO, IT WILL CAUSE SHUFFLING

val list = List((1,"me").....)

xxxDs = list.toDS

xxxDs.groupByKey(p=> p.\_1)

.mapGroups((k,vs)=> (k,vs.foldLeft("")((acc,p) => acc + p.\_2)))

- suggestion is to use Aggrgator instead, found in "org.apache.spark.sql.expressions.Aggregator"

-may need encoders from org.apache.spark.sql.Encoders





