Apache Spark

Session #2

Prasanth Prahladan
prpr2770@colorado.edu

RDD - Resilient Distributed Dataset

+ Immutable distributed collection of objects
+ Each RDD is split into multiple “partitions”.
+ “Partitions” are computed on different nodes of cluster

+ 2 operations: “transformations” + “actions”

RDD - Operations

- Transformations: [# RDD] -> [RDD]
  - construct new RDD from previous one.
  - map(), filter()

- Actions: [RDD] -> results(non-RDD data type)
  - compute results(numeric, etc) based on an RDD
  - returns result to Driver Program or,
  - saves result to external storage system.
  - count(), first()

RDD - Why different Operations?

+ Spark computes RDDs in a unique manner.
+ RDD can be defined at any time.
+ “lazy” computation - first time its used in an Action.
+ Spark observes the chain of transformations, before an action. Optimizes accordingly.

RDD - Persistence

+ Spark does not persist an RDD in memory or on disk.
+ By default, an RDD is recomputed each time you run an action on it.
+ RDD.persist() or RDD.cache()
+ Enable reuse of an RDD in multiple actions.

Working of a Spark program

1. Create input RDD from external data.
2. Apply Transformations:
    a. RDD -> new_RDD.
3. new_RDD.persist()
    a. if needed for reuse.
4. Launch Action:
    a. kicks off parallel computation
    b. (optimized and executed by Spark engine).

Creating RDDs

+ Loading external dataset
  + sc.textFile(), other examples.

+ Parallelizing a collection in driver-program
  + sc.parallelize()
  + useful for learning, prototyping, testing.
  + it requires entire memory in one machine.

RDD Transformations

+ “Lazy Evaluation”
  + Transformations are computed lazily
  + When an action is called, only then are the preceding transformations computed!
+ “Lineage Graph”
  + family-tree of RDDs defined/generated in program.
  + used to compute each RDD on demand, or
  + recovery of lost data if persistent RDD is damaged.

RDD Actions

- why? to do something with your data!
- what?
  - force evaluation of transformations req. for its RDD
  - returns a final value

- each time we call a new action, the entire RDD must be computed “from scratch”.

Understanding Lazy Evaluation

- why? reduce #passes over data, by grouping transformations
- At each call of a Transformation,
  - Spark internally records meta-data
  - no immediate execution, until action call.
- RDD
  - view 1: collection of data
  - view 2: consists of instructions on how to compute the data we build up through transformations.

Passing functions to Spark

+ Spark Operations take Functions as input.
+ Using Python: 3 options
  + lambda expressions
  + top-level functions
  + locally defined functions

+ Risk: Serializing the object containing the function called.

Common Operations

1. Basic RDDs
    a. Element-wise transformations
    b. Pseudo-set operations
    c. Actions
2. Converting b/w RDD Types
3. Persistence(Caching)

Common Operations

Key-Value Pair RDDs

+ ETL -> KV-pair RDD generation
+ Operations on KV-pairs : aggregation,etc
+ Layout of KV-pair RDD across cluster-nodes
  + Partitioning
  + controls and reduces communication costs
+ Why Partitioning? Performance!
  + local dataset -> choose the right Data-Structure.
  + distr. dataset -> choose the right Partitioning!

Creating KV-pair RDDs

+ RDDs of tuples!

+ Methods
  + Loading operations used for External data
  + Transformations of Regular RDDs

Transformations on Pair-RDDs

+ Input: Functions that operate on tuples

Transformations on Pair-RDDs

Tuning level of parallelism

- All transformations are distributed
- How does Spark decide to split up work?
  - RDD <= {# partitions}
  - the number of partitions determine degree of parallelism to use during operation.
  - aggregation/grouping operations - specify #partitions to use.

Repartitioning RDD

+ Change Partitioning of RDD outside context of aggregation/grouping operations.
+ repartition()
  + Shuffles data across network: create new partitions
+ coalesce()
  + optimized version
  + used to decrease # partitions of RDD
+ rdd.getNumPartitions()

Grouping Data

- grouping data by key
  - viewing all orders of a single customer together
  - Single RDD
    - rdd.groupByKey()
    - rdd.groupBy()
  - Multiple RDDs
    - cogroup()
      - used as building block for JOINS!
      - implement INTERSECT by KEY

Joins

+ analogy: SQL JOIN
  + combining fields from two tables using common val.

+ Types:
  + rightOuterJoin() and leftOuterJoin()
  + cross join
  + inner join

Sorting Data

rdd.sortByKey(ascending=True, keyfunc=@fhd)

Actions on Pair-RDDs

+ rdd.countByKey()
+ rdd.collectAsMap()
+ rdd.lookup(key)