UE Large Scale Processing @ ESIGELEC
2019/2020

13 – Spark SQL – RDD vs DataSet vs DataFrame

Abdel Dadouche
DJZ Consulting

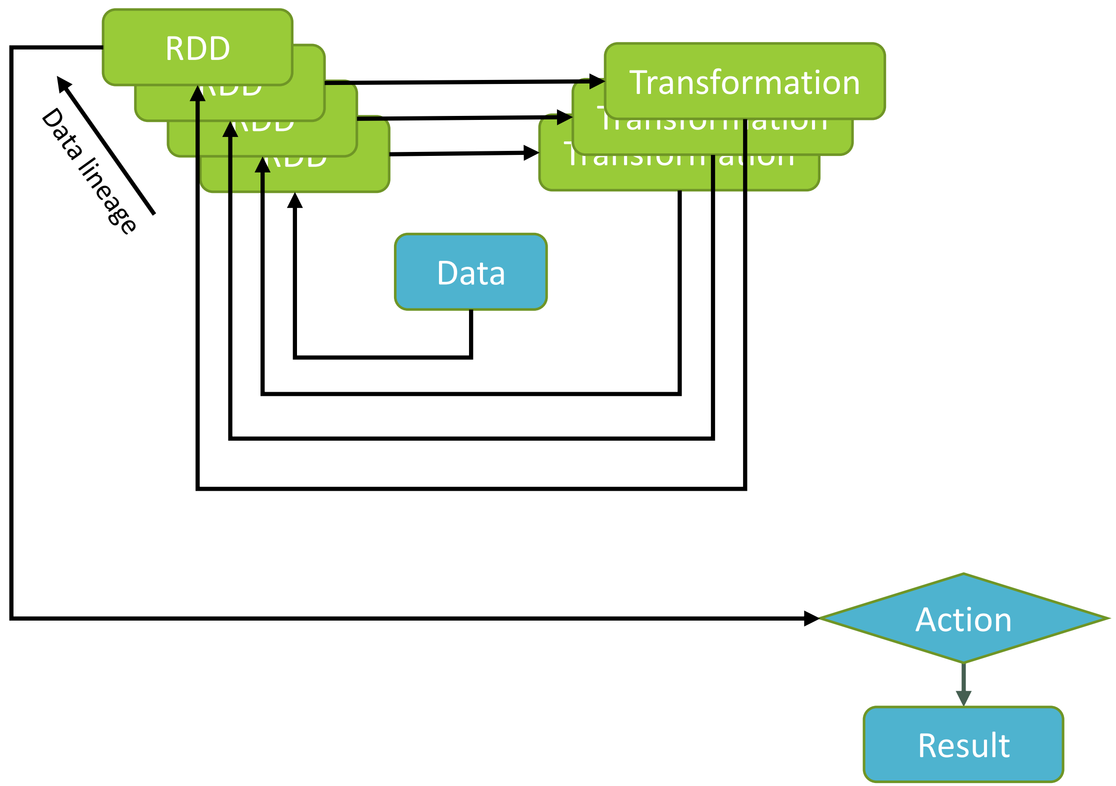
adadouche@hotmail.com
@adadouche

Some History

- Initially, only the RDD API was available starting 2012
  - Primary user-facing API
  - Really low level API
- Then, the DataFrame API appeared in 2013
  - an abstraction on top of the RDD
- And finally Dataset in 2015
  - provides both a strongly-typed API and an untyped API

Remember RDDs? Resilient Distributed Dataset?

- RDD:
  - immutable, iter-able, partion-able & lazy loaded data structure
  - Provide data lineage capabilities (can be reconstructed)
  - Represent each and every step of the execution
- Transformation
  - Create a new RDD from an existing one applying an operation (map, filter, Sample, Union…)
- Action:
  - Evaluate the chain of transformation on the RDD object and return a result



When to use RDDs? Common scenarios / use cases

- Need low-level transformations & actions or control on your dataset
- Use of unstructured dataset, such as media streams or streams of text
- Need to manipulate your data with functional programming constructs
- No need to process or access data attributes by name or column (use of schema)
- No specific optimizations and performances requirements for structured and semi-structured data

What is the DataFrame API in Spark?

- A DataFrame is a distributed collection of untyped objects, which can hold various types of tabular data
- Conceptually equal to a table in a relational db
- The DataFrame API allows you to perform relational/procedural operations using a domain-specific language (DSL) similar to R and Python Pandas data frames
- Leverage the SparkSQL Catalyst optimizer
- Provide a higher level of abstraction than RDD and a much simpler/concise syntax
- The DataFrame API is considered as ***"untyped"***

Ex: Calculate an average

- Using RDD

```
data = sc.textFile(…).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
    .reducByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
    .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
    .collect()
```

- Using DataFrames

```
sqlCtx.table("people") \
    .groupBy("name") \
    .agg("name", avg("age")) \
    .collect()
```

- Using SQL

```
SELECT name, avg(age) FROM people GROUP BY name
```

What is the Dataset API in Spark?

- A Dataset is an extension of a DataFrame
- The Dataset API allows users to assign a class to the records inside a DataFrame, and manipulate it as a collection of typed objects
- A Dataset is a distributed collection of "***strongly-typed***" objects
- However, as Python and R don't have compile-time type-safety, the concept of Dataset does not apply

- Since Spark 2.0, the Dataset API unified the DataFrame API to provide both the ***strongly-typed*** and ***untyped*** transformations (RDD)

  ➔For the ***untyped*** API, a DataFrame becomes an *alias* for a collection of generic objects *Dataset[Row]*, where a *Row* is a generic ***untyped*** object

Language	Main Abstraction Available
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python	DataFrame
R	DataFrame

When to use DataFrame/Datasets?

- Need rich semantics, high-level abstractions, and domain specific APIs (DF, DS)
- Need high-level expressions, filters, maps, aggregation, averages, sum, SQL queries, columnar access or use of lambda functions on semi-structured data (DF, DS)
- Need unification and simplification of APIs across Spark Libraries (DF, DS)
- Need higher degree of type-safety at compile time, typed objects, take advantage of Catalyst optimization, and benefit from more efficient code generation (DS)
- Developing in R (DF)
- Developing in Python (DF or RDD)

Benefits of the Dataset APIs

Static-typing and runtime type-safety

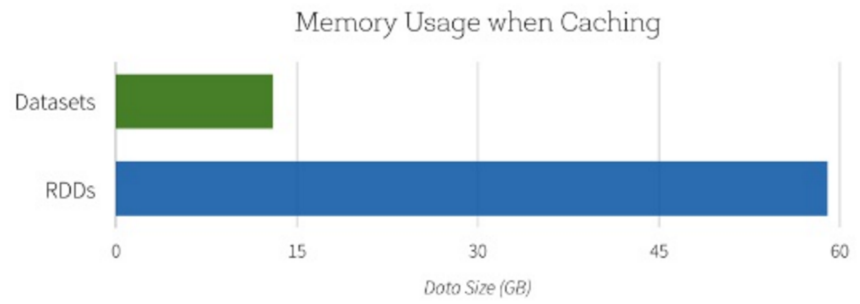
- All Dataset APIs are expressed as lambda functions and JVM typed objects
- Any mismatch of typed-parameters will be detected at compile time

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

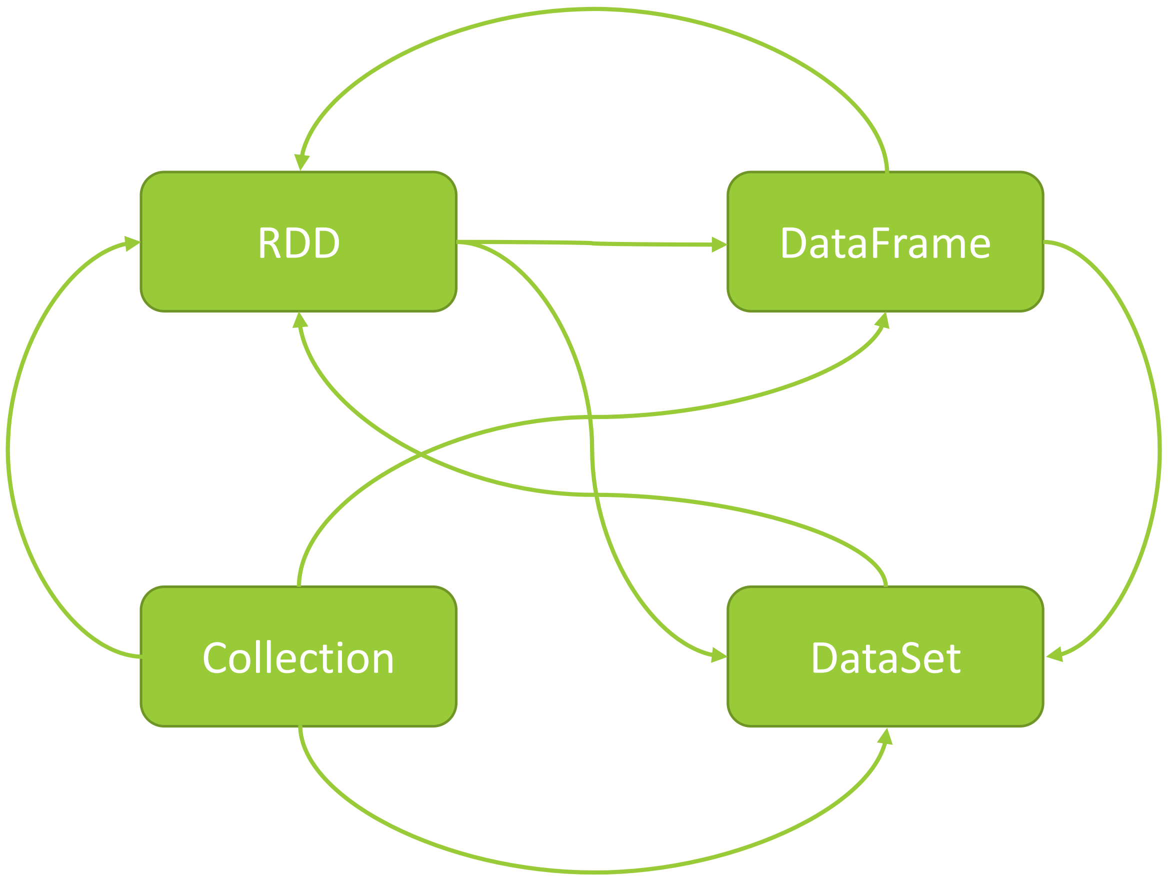
Performance and Optimization

- DataFrame and Dataset APIs are built on top of the Spark SQL engine and uses Catalyst

- Dataset typed JVM objects can be maped to Tungsten's internal memory representation using Encoders to efficiently serialize/deserialize JVM objects as well as generate compact bytecode that can execute at superior speeds.



Conversion of RDD / DataFrame / Dataset



Input	Output	Method / Property
Collection	DataSet	.toDS()
RDD	Dataset	.toDS()
Collection	DataFrame	.toDF()
RDD	DataFrame	.toDF()
DataFrame	Dataset	.as[SomeClass]
Collection	RDD	.rdd
DataFrame	RDD	.rdd
Dataset	RDD	.rdd

Collection : instantiated via SparkContext.parallelize()