

MA-INF 4223 - Lab Distributed Big Data Analytics

Spark Fundamentals II

Dr. Hajira Jabeen, Gezim Sejdiu

Winter Semester 2017/18



Lesson objectives

- ❖ After completing this lesson, you should be able to:
 - Understand and use various Spark Libraries
 - Spark SQL
 - Spark GraphX - graph processing



Pair RDDs

- A common form of data processing
- Main intuition behind the mapreduce
- Often beneficial to project down the complex datatypes to Key-value pairs

Distributed key-value pairs

- Additional specialised methods for working with data associated with Keys
- `groupByKey()`,`reduceByKey()`,`join`



Pair RDDs

Creation: Mostly from existing non-pair RDDs

E.g.

```
val pairRdd = rdd.map(page => (page.title, page.text))
```

- groupByKey
- reduceByKey
- mapValues
- keys
- Join
- leftOuterJoin/rightOuterJoin
- countByKey



Shuffling

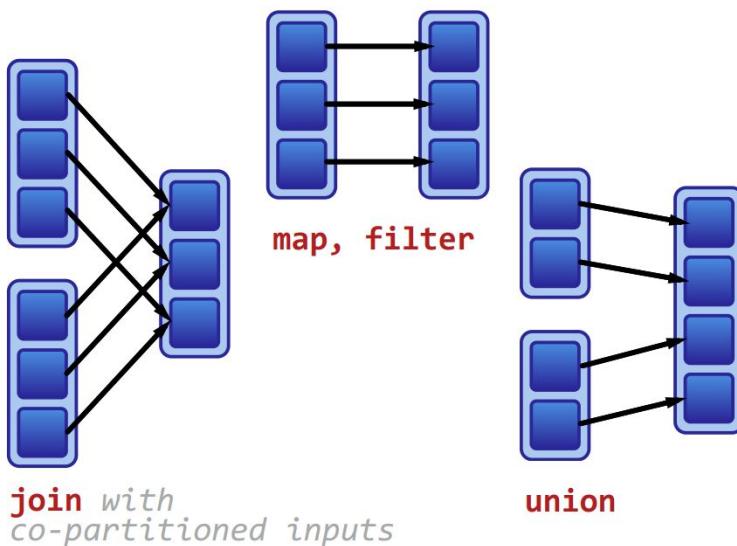
- `groupByKey()`
 - Shuffles all the keys across network to combine all the keys
- `reduceByKey(func: (V, V) => V): RDD[(K, V)]`
 - Conceptually, `reduceByKey` can be thought of as a combination of first doing `groupByKey` and then reducing on all the values grouped per key.
 - Reduces on the mapper side first
 - Reduce again after shuffling
 - Less data needs to be sent over the network
 - Non trivial gains in performance



Dependencies / Shuffling

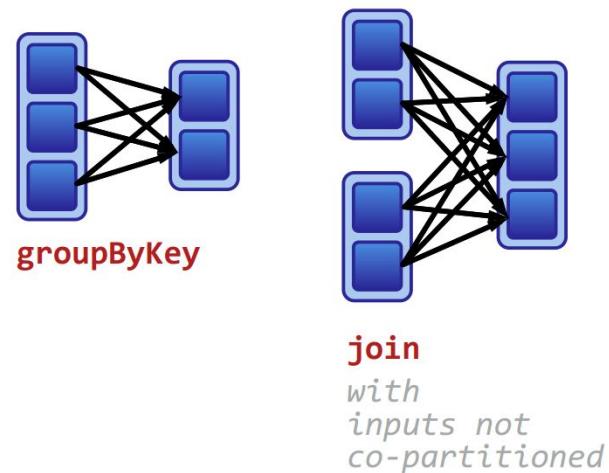
Narrow dependencies:

Each partition of the parent RDD is used by at most one partition of the child RDD.



Wide dependencies:

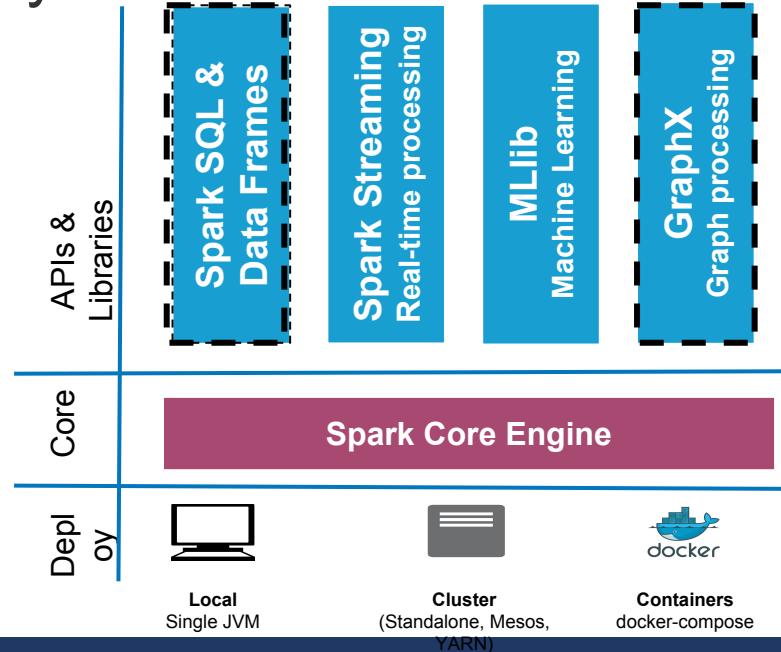
Each partition of the parent RDD may be depended on by multiple child partitions.





Spark Libraries

- ❖ A unified analytics stack





Overview

- ❖ [Spark SQL: Relational Data Processing in Spark](#)
- ❖ [GraphX: A Resilient Distributed Graph System on Spark](#)



Motivation

- ❖ Support relational processing both within Spark programs
- ❖ Provide high performance with established DBMS techniques
- ❖ Easily support new data sources, including semi-structured data and external databases amenable to query federation
- ❖ Enable extension with advanced analytics algorithms such as graph processing and machine learning



Motivation

- ❖ Users:
 - Want to perform ETL-relational
 - data Frames
 - Analytics - procedural tasks
 - UDFs



Spark SQL

- ❖ A module that integrates relational processing with Spark's Functional programming API
- ❖ SQL allows relational processing
- ❖ Perform complex analytics
 - Integration between relational and procedural processing through declarative Data Frame
 - Optimizer (catalyst)
 - Composable rules
 - Control code generation
 - Extension points
 - Schema inference for json
 - ML types
 - Query federation



Spark SQL

Three main APIs

- SQL literal Syntax
- DataFrames
- Datasets

Two specialised backend components

- Catalyst
- Tungsten



Data Frame

- ❖ DataFrames are collections of structured records that can be manipulated using Spark's procedural API,
- ❖ Supports relational APIs that allow richer optimizations.
- ❖ Created directly from Spark's built-in distributed collections of Java/Python objects,
- ❖ Enables relational processing in existing Spark Programs
- ❖ DataFrame operations in SparkSQL go through a relational optimizer, Catalyst



Catalyst

- ❖ Catalyst is the first production quality query optimizer built on such functional language.
- ❖ It contains an extensible query optimizer
- ❖ Catalyst uses features of the Scala programming language,
 - Pattern-matching,
 - Express composable rules
 - Turing complete language

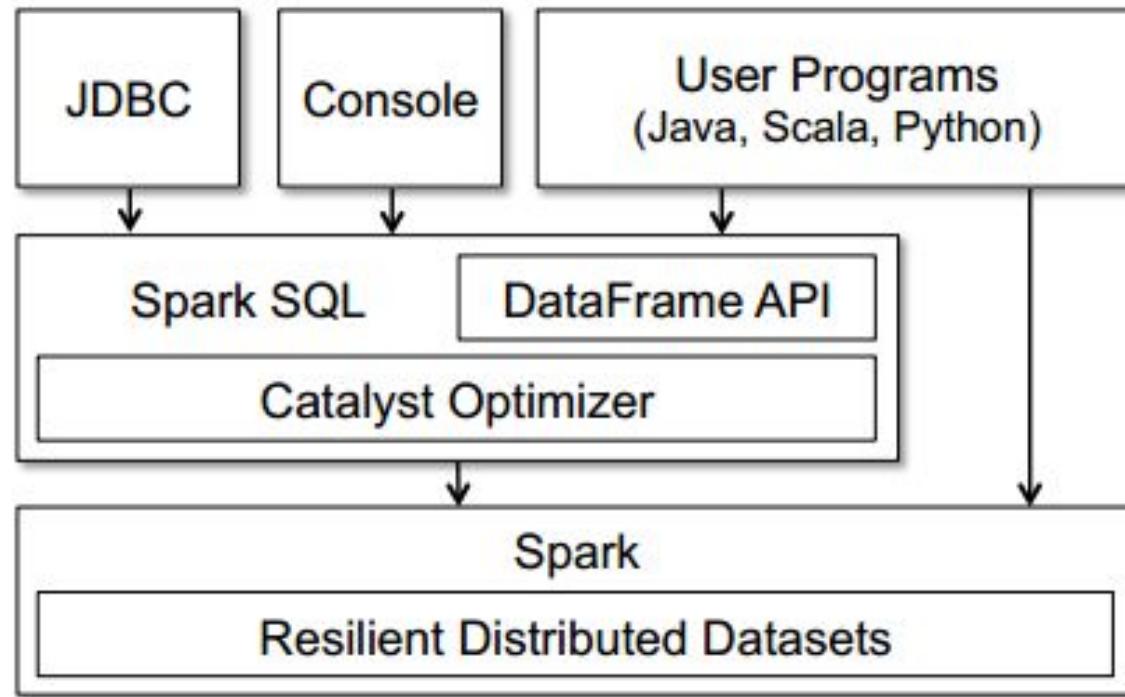


Catalyst

- ❖ Catalyst can also be
 - extended with new data sources,
 - semi-structured data
 - such as JSON
 - “smart” data stores to use push filters
 - e.g., HBase
 - user-defined functions;
 - User-defined types for domains e.g. machine learning.
- ❖ Spark SQL simultaneously makes Spark accessible to more users and improves optimizations



Spark SQL





DataFrame

- ❖ DataFrame is a distributed collection of rows with the “Known” schema like table in a relational database.
- ❖ Each DataFrame can also be viewed as an RDD of Row objects, allowing users to call procedural Spark APIs such as map.
- ❖ Spark DataFrames are lazy, in that each DataFrame object represents a logical plan to compute a dataset, but no execution occurs until the user calls a special “output operation” such as save



DataFrame

- ❖ Created from an RDD using .toDF()
- ❖ Reading from a file ()



Example

```
ctx = new HiveContext()
users = ctx.table("users")
young = users.where(users("age") < 21)
println(young.count())
```



Data Model

- ❖ DataFrames support all common relational operators, including
 - projection (select),
 - filter (where),
 - join, and
 - aggregations (groupBy).
- ❖ Users can break up their code into Scala, Java or Python functions that pass DataFrames between them to build a logical plan, and will still benefit from optimizations across the whole plan when they run an output operation.



Optimization

- ❖ The API analyze logical plans eagerly
 - identify whether the column names used in expressions exist in the underlying tables,
 - whether the data types are appropriate
- ❖ Spark SQL allows users to construct DataFrames directly against RDDs of objects native to the programming language.
- ❖ Spark SQL can automatically infer the schema of these objects using reflection



Optimization

- ❖ Uses columnar cache
 - reduce memory footprint by an order of magnitude because it applies columnar compression schemes such as dictionary encoding and run-length encoding.
- ❖ In Spark SQL, UDFs can be registered inline by passing Scala, Java or Python functions, which may use the full Spark API internally.



Catalyst - Extension

- ❖ Easy to add new optimization techniques and features to Spark SQL
- ❖ Enable external developers to extend the optimizer
 - E.g. adding data source specific rules that can push filtering or aggregation into external storage systems,
 - support for new data types.
- ❖ Catalyst supports both rule-based and cost-based optimizations



Catalyst

- ❖ Catalyst contains a general library for representing trees(Abstract Syntax Tree) and applying rules to manipulate them
- ❖ Catalyst offers several public extension points, including external data sources and user-defined types.



Trees

- ❖ The main data type in Catalyst is a tree composed of node objects.
- ❖ Each node has a node type and zero or more children.
- ❖ New node types are defined in Scala as subclasses of the TreeNode class.
- ❖ These objects are immutable and can be manipulated using functional transformations



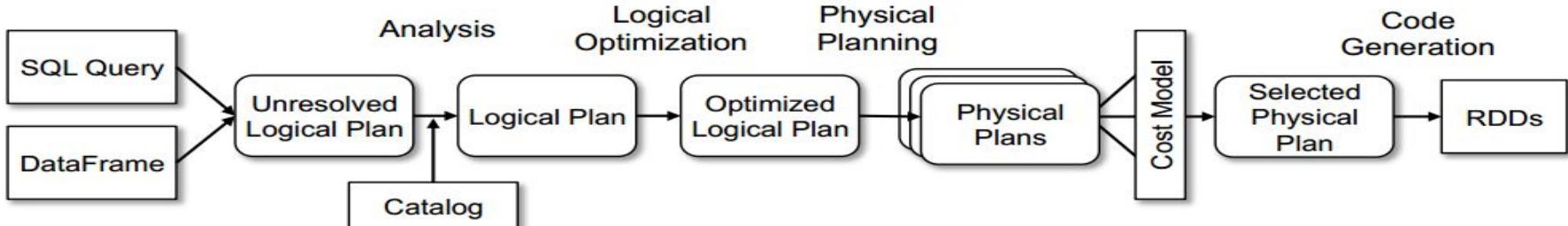
Rules

- ❖ Trees can be manipulated using rules, which are functions from a tree to another tree.
- ❖ While a rule can run arbitrary code on its input tree (given that this tree is just a Scala object),
- ❖ the most common approach is to use a set of pattern matching functions that find and replace subtrees with a specific structure.



Tree Transformation

- ❖ Catalyst's general tree transformation framework works in four phases
 - analyzing a logical plan to resolve references
 - logical plan optimization
 - physical planning
 - code generation to compile parts of the query to Java bytecode.





References

- [1]. [Spark SQL: Relational Data Processing in Spark](#) by Armbrust, Michael, Reynold Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi and Matei Zaharia *in SIGMOD Conference*, 2015.
- [3]. “Spark SQL, DataFrames and Datasets Guide” -
<http://spark.apache.org/docs/latest/sql-programming-guide.html>

THANK YOU !

<http://sda.cs.uni-bonn.de/teaching/dbda/>

- <http://sda.cs.uni-bonn.de/>
- <https://github.com/SANSA-Stack>
- <https://github.com/big-data-europe>
- <https://github.com/SmartDataAnalytics>



Dr. Hajira Jabeen
jabeen@cs.uni-bonn.de

Room A108 (Appointment per e-mail)



Gezim Sejdiu
sejdiu@cs.uni-bonn.de

Room A120 (Appointment per e-mail)