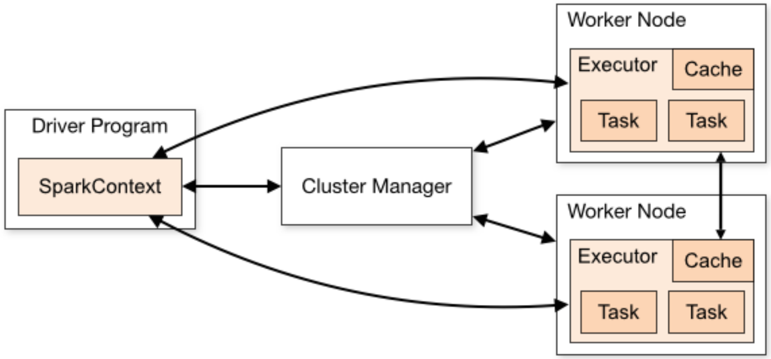
Modern Database Systems

Practicals: Spark

Doc. RNDr. Irena Holubova, Ph.D.

holubova@ksi.mff.cuni.cz

Spark Application



- Spark application = driver program
  - Runs the user's main function
  - Executes parallel operations on a cluster
    - Independent set of processes
    - Coordinated by SparkContext object in the driver program
- SparkContext can connect to several types of cluster managers
  - They allocate resources across applications
- When connected:
  1. Spark acquires executors on nodes in the cluster
     - Processes that run computations and store data for the application
  2. Sends the application code to the executors
     - Defined by JAR or Python files passed to SparkContext
  3. Sends tasks to the executors to run

Initializing Spark

1. Build a SparkConf object
   - Contains information about application
   - appName = application name to show on the cluster UI
   - master = Spark/Mesos/YARN cluster URL or string “local” to run in local mode
2. Create a JavaSparkContext object
   - Tells Spark how to access a cluster

```
SparkConf conf =
  new SparkConf().setAppName(appName).setMaster(master);
JavaSparkContext sc =
  new JavaSparkContext(conf);
```

Resilient Distributed Dataset (RDD)

- Immutable collection of elements partitioned across the nodes of the cluster
  - Can be operated on in parallel
  - Can be persisted in memory
    - MapReduce: has to be written od disk between Map and Reduce
  - Automatically recover from node failures
- Ways to create RDDs:
  1. Parallelizing an existing collection in a driver program
  2. Referencing a dataset in an external storage system
     - e.g., HDFS, HBase, …
     - In general: any offering a Hadoop InputFormat

https://spark.apache.org/docs/latest/rdd-programming-guide.html

Resilient Distributed Dataset (RDD)

Parallelized Collections

- Parallelized collections are created by calling SparkContext’s parallelize method
    - Elements of the collection are copied to form a distributed dataset
    - The distributed dataset (distData) can be operated on in parallel
        - See later

```
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);
JavaRDD<Integer> distData = sc.parallelize(data);
```

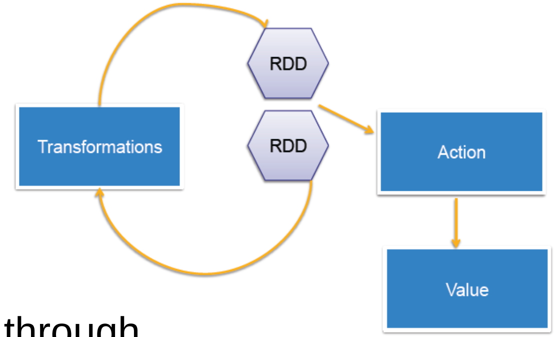
Resilient Distributed Dataset (RDD)

External Datasets

- Spark can create distributed datasets from any storage source supported by Hadoop
  - Local file system, HDFS, Cassandra, HBase, …
- Supports text files, SequenceFiles, and any other Hadoop InputFormat
- Example:
  - Text file RDDs can be created using SparkContext’s textFile method
    - Takes an URI for the file (local, HDFS, …)
    - Reads it as a collection of lines
    - Optional argument: number of partitions of the file
      - Default: one partition for each block of the file (128MB by default in HDFS)
  - Once created, distFile can be acted on by dataset operations

```
JavaRDD<String> distFile = sc.textFile("data.txt");
```

RDD Operations



1. Transformations = create (lazily) a new dataset from an existing one
   - e.g., map = passes each dataset element through a function and returns a new RDD representing the results
2. Actions = return a value to the driver program after running a computation on the dataset
   - e.g., reduce = aggregates all the elements of the RDD using some function and returns the final result to the driver program

- By default: each transformed RDD may be recomputed each time we run an action on it
  - We may also persist an RDD in memory using the persist (or cache) method
    - Much faster access the next time we query it
  - There is also support for persisting RDDs on disk or replicated across multiple nodes

Transformations

- **map**(func) Returns a new distributed dataset formed by passing each element of the source through a function func.
- **union**(otherDataset) Returns a new dataset that contains the union of the elements in the source dataset and the argument.
  - **intersection, distinct**
- **filter**(func) Returns a new dataset formed by selecting those elements of the source on which func returns true.
- **reduceByKey**(func, [numPartitions]) When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V. The number of reduce tasks is configurable through an optional second argument.
- **sortByKey**([ascending], [numPartitions]) When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the Boolean ascending argument.
- …

https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

Actions

- **reduce**(func) Aggregates the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
- **count**() Returns the number of elements in the dataset.
- **first**() Returns the first element of the dataset.
- **take**(n) Returns an array with the first n elements of the dataset.
- **takeOrdered**(n, [ordering]) Returns the first n elements of the RDD using either their natural order or a custom comparator.
- …

https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

Passing Functions

- **By lambda expression**

```
data.reduceByKey((a, b) -> a + b);
```

- **By interface function**
- Java: functions are represented by classes implementing interface Function[2,3,4]<IN[,IN[,IN[,IN]]], OUT> from package org.apache.spark.api.java.function
- Pass an instance of implemented class (either as an anonymous inner class or a named one)

```
data.reduceByKey(new Function2<Integer, Integer, Integer>() {
  @Override
  public Integer call(Integer a, Integer b) throws Exception {
    return a + b;
  }
});
```

Spark SQL

- Spark module for structured data processing
- Spark SQL data structures (DataFrame, Dataset) provide information about the structure of the data and the computation
- Supports execution of SQL queries
- Supports reading data from an existing database (Hive, MySQL, ...)

- The entry point is the SparkSession class

```
SparkSession spark =
SparkSession.builder().appName("AppName").getOrCreate();
```

DATAFRAME, DATASET

- **DataFrame**
  - Distributed collection of data, which is organized into named columns
  - Conceptually equivalent to a table in a relational database
  - Can be constructed from structured data files, external databases, existing RDDs, ...

```
Dataset<Row> dataFrame = spark.read().json("actors.json");
```

- **DataSet**
  - Distributed collection of data
  - Can be constructed from strongly-typed JVM objects and manipulated using transformations
  - Ability to use lambda functions

```
Dataset<Person> dataset =
spark.read().json("actors.json").as(actorEncoder);
```

(Optional) Assignment

- Chose your unique problem domain
  - E.g., the results of football matches of various teams
- Think about an original computation problem in your domain which might be solved using Spark
- Create respective sample data
  - They do not need to be large – this is not the aim of the assignment
- Submit either a script for the spark-shell or modify any on the `*.java` examples from practicals

References

- Spark Overview https://spark.apache.org/docs/latest/index.html
- Apache Spark Examples https://spark.apache.org/examples.html
- Apache Spark Quick Start https://spark.apache.org/docs/latest/quick-start.html
- A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html