Apache Spark Part 2

COMP9313: Big Data Management

RDD Persistence: Cache/Persist

- One of the most important capabilities in Spark is *persisting* (or *caching*) a dataset in memory across operations
- When you persist an RDD, each node stores any partitions of it. You can reuse it in other actions on that dataset
- Each persisted RDD can be stored using a different *storage level,* e.g.
  - MEMORY_ONLY:
    - Store RDD as deserialized Java objects in the JVM
    - If the RDD does not fit in memory, some partitions will not be cached and will be recomputed when they're needed
    - This is the default level
  - MEMORY_AND_DISK:
    - If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed
- cache() = persist(StorageLevel.MEMORY_ONLY)

Why Persisting RDD?

- If you repeat a transformation again, the file will be loaded again and computed again
- Persist will tell Spark to cache the data in memory, to reduce the data loading cost for further actions on the same data
- RDD.persist() will do nothing. It is a lazy operation. But now the RDD says "read this file and then cache the contents". The action will trigger computation and data caching

Spark Key-Value RDDs

- Similar to Map Reduce, Spark supports Key-Value pairs
- Each element of a *Pair RDD* is a pair tuple
- Some Key-Value transformation functions:

Key-Value Transformation	Description
`reduceByKey(`*`func`*`)`	return a new distributed 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
`sortByKey()`	return a new dataset (K,V) pairs sorted by keys in ascending order
`groupByKey()`	return a new dataset of (K, Iterable<V>) pairs

Spark Programming Model

How Spark Works?

- User application create RDDs, transform them, and run actions
- This results in a DAG (Directed Acyclic Graph) of operators
- DAG is compiled into stages
- Each stage is executed as a series of Tasks (one Task for each Partition)

What is a DAG?

- From Graph Theory, a Graph is a collection of nodes connected by branches.
- A DAG (Directed Acyclic Graph) is a directed graph in which there are no cycles or loops, i.e., if you start from a node along the directed branches, you would never visit the already visited node by any chance.

What is a DAG to Apache Spark?

- Spark Driver identifies the tasks implicitly that can be computed in parallel with partitioned data in the cluster.
- Spark Driver builds a logical flow of operations that can be represented in a graph which is directed and acyclic, also known as DAG (Directed Acyclic Graph)
- Spark builds its own plan of executions implicitly from the spark application provided

Narrow Transformation and Wide Transformation

- Narrow transformation
  - All the elements that are required to compute the records in single partition live in the single partition of parent RDD.
  - A limited subset of partition is used to calculate the result.
  - Narrow transformations are the result of map(), filter().
- Wide transformation
  - All the elements that are required to compute the records in the single partition may live in many partitions of parent RDD.
  - The partition may live in many partitions of parent RDD.
  - Wide transformations are the result of groupbyKey and reducebyKey

Word Count in Spark

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
.flatMap(s -> Arrays.asList(s.split(" ")).iterator())
.mapToPair(word -> new Tuple2<>(word, 1))
.reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

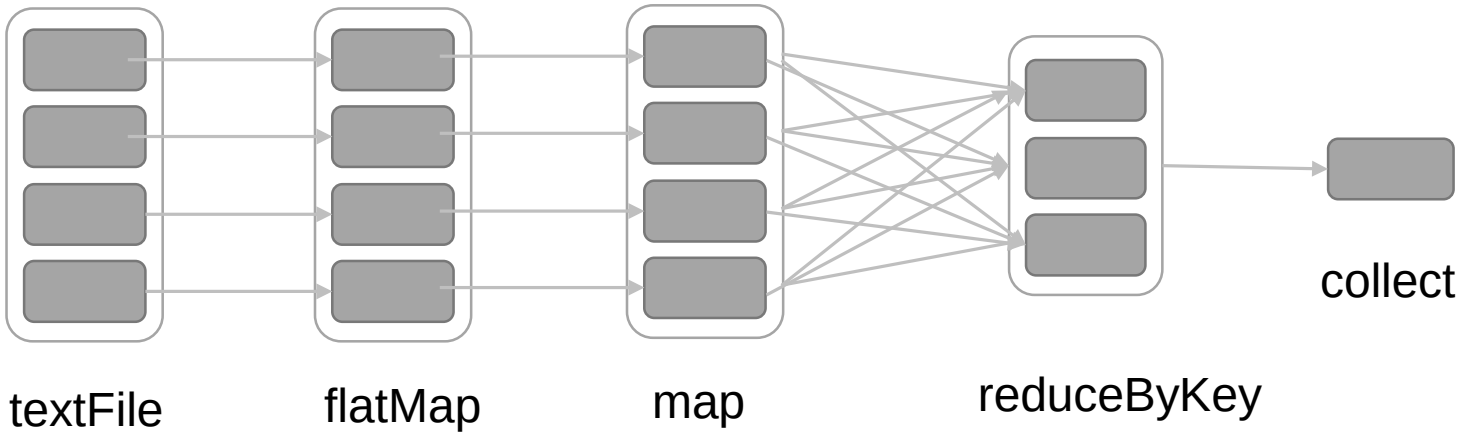
RDD[String]

RDD[List[String]]

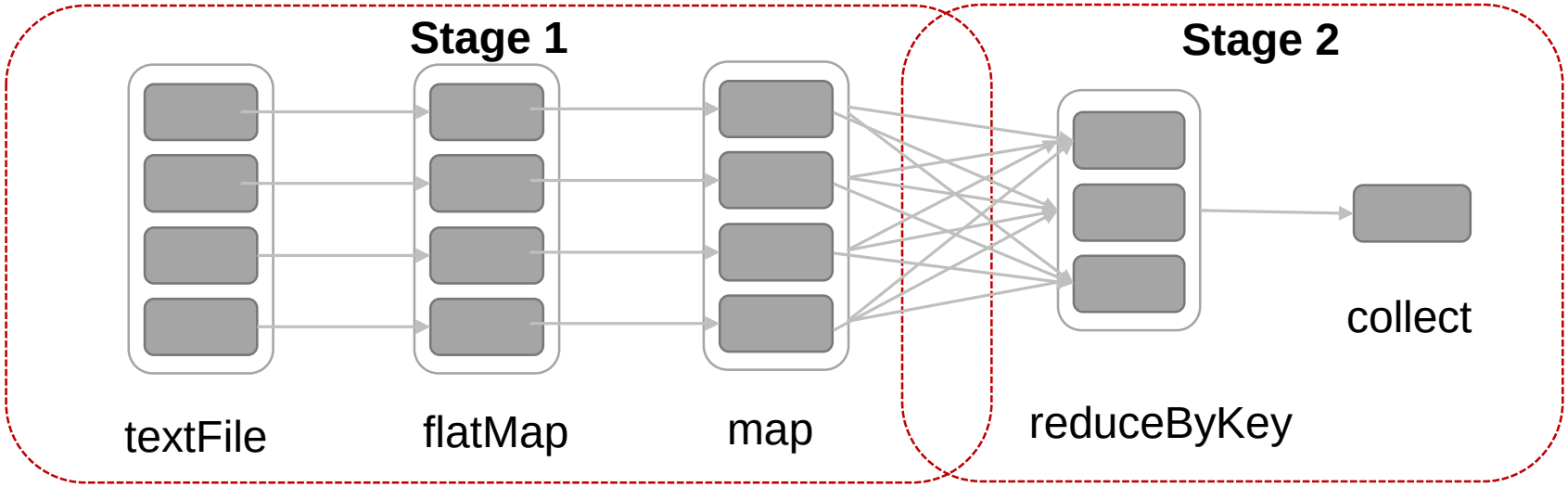
RDD[(String, Int)]

RDD[(String, Int)]

Array[(String, Int)]

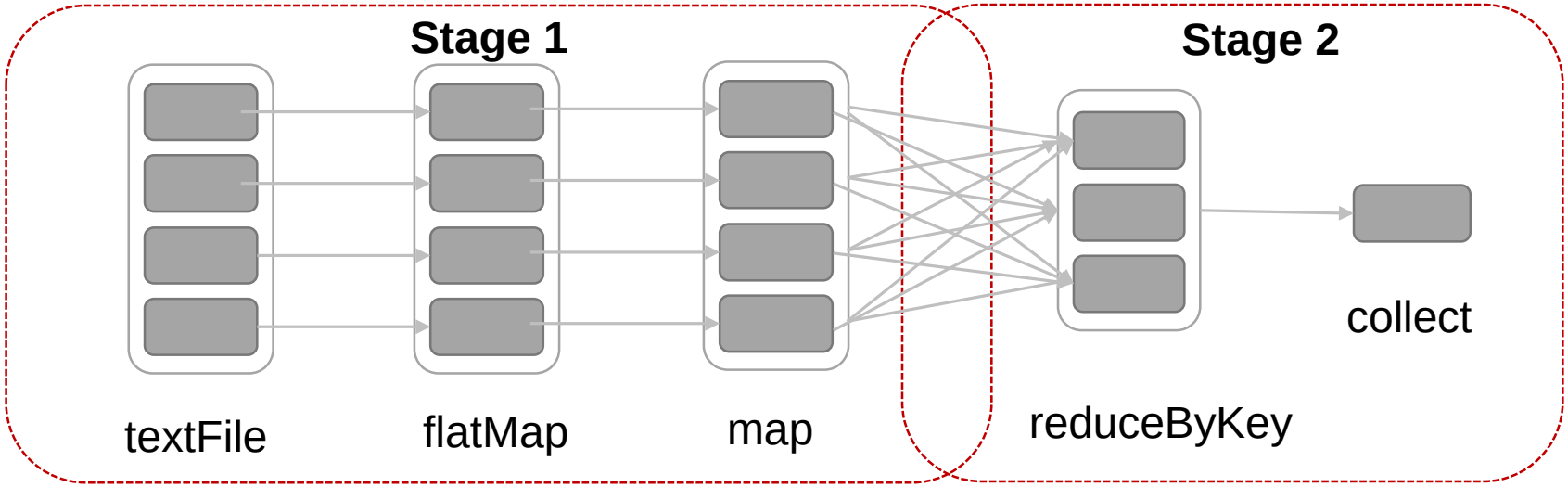


Execution Plan

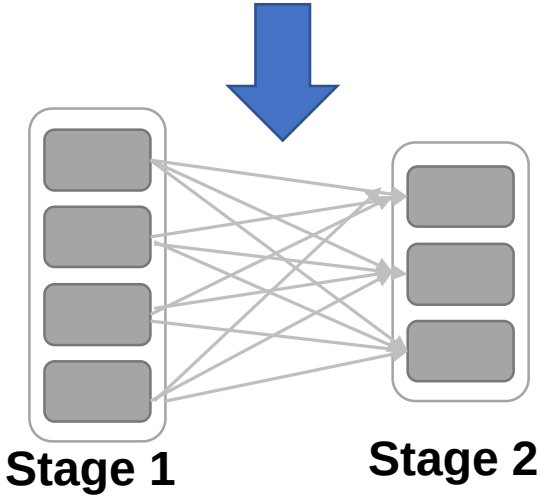


- The scheduler examines the RDD's lineage graph to build a DAG of stages
- Stages are sequences of RDDs, that don't have a Shuffle in between
- The boundaries are the shuffle stages

Execution Plan

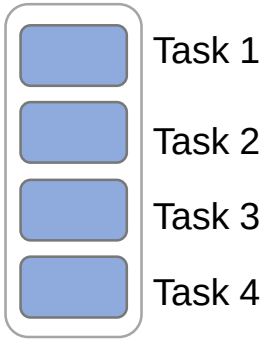


1. Read HDFS split
2. Apply both the maps
3. Start Partial reduce
4. Write shuffle data



1. Read shuffle data
2. Final reduce
3. Send result to driver program

Stage Execution



- Create a task for each Partition in the new RDD
- Serialize the Task
- Schedule and ship Tasks to Slaves

- All this happens internally

Word Count in Java (As a Whole View)

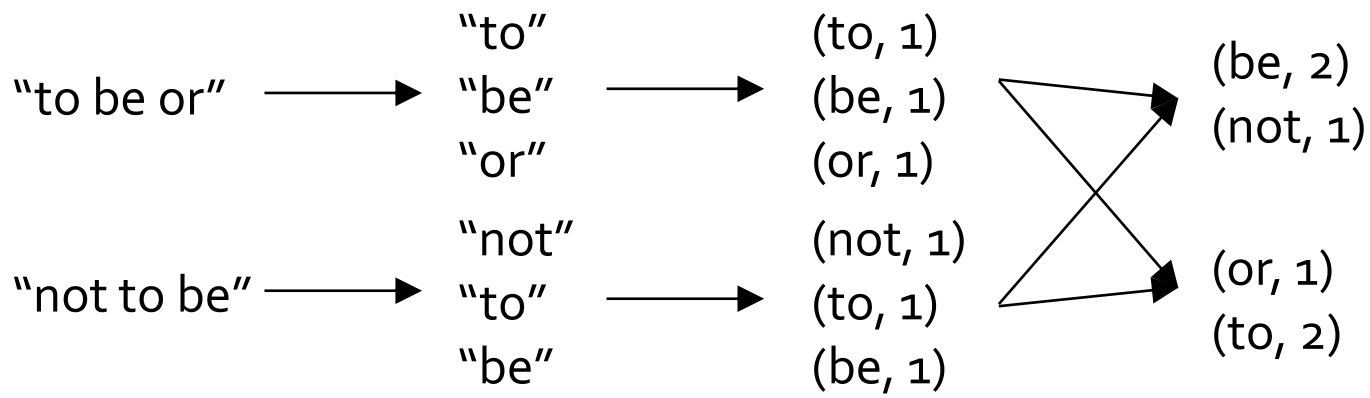
- Word Count using Scala in Spark

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
 .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
 .mapToPair(word -> new Tuple2<>(word, 1))
 .reduceByKey((a, b) -> a + b);
```

Transformation

```
counts.saveAsTextFile("hdfs://...");
```

Action



Self-Contained Applications

WordCount

- Linking with Apache Spark
  - The first step is to explicitly import the required spark classes into your Spark program

```
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.SparkConf;
```

- Initializing Spark
  - Create a Spark context object with the desired spark configuration that tells Apache Spark on how to access a cluster

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

  - SparkConf: Spark configuration class
  - setAppName: set the name for your application
  - setMaster: set the cluster master URL

setMaster

- Set the cluster master URL to connect to
- Parameters for setMaster:
  - local(default) - run locally with only one worker thread (no parallel)
  - local[k] - run locally with k worker threads
  - spark://HOST:PORT - connect to Spark standalone cluster URL
  - mesos://HOST:PORT - connect to Mesos cluster URL
  - yarn - connect to Yarn cluster URL
    - Specified in SPARK_HOME/conf/yarn-site.xml
- setMaster parameters configurations:
  - In source code
    - SparkConf().setAppName("wordCount").setMaster("local")
  - spark-submit
    - spark-submit --master local
  - In SPARK_HOME/conf/spark-default.conf
    - Set value for spark.master

WordCount

- Creating a Spark RDD
  - Create an input Spark RDD that reads the text file input.txt using the Spark Context created in the previous step

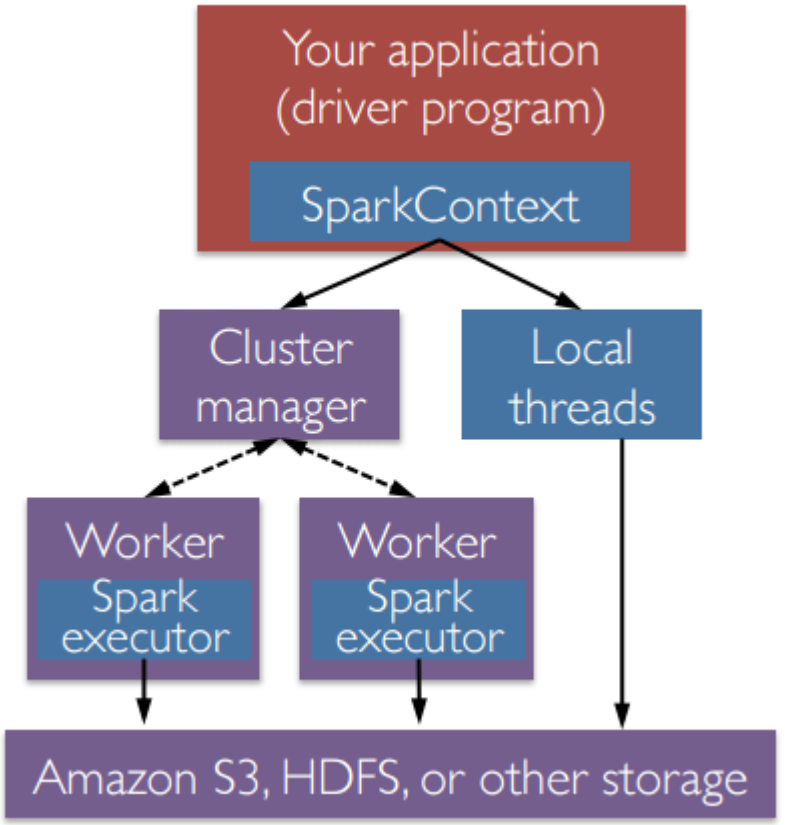
```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
```

- Spark RDD Transformations in Wordcount Example
  - flatMap() is used to tokenize the lines from input text file into words
  - mapToPair() method counts the frequency of each word
  - reduceByKey() method counts the repetitions of word in the text file
- Save the results to disk

```
counts.saveAsTextFile("hdfs://...");
```

Run the Application on a Cluster

- A Spark application is launched on a set of machines using an external service called a cluster manager
  - Local threads
  - Standalone
  - Mesos
  - Yarn
- Driver
- Executor



Launching a Program in a Cluster

- Spark provides a single script you can use to submit your program to it called **spark-submit**
  - The user submits an application using spark-submit
  - spark-submit launches the driver program and invokes the main() method specified by the user
  - The driver program contacts the cluster manager to ask for resources to launch executors
  - The cluster manager launches executors on behalf of the driver program
  - The driver process runs through the user application. Based on the RDD actions and transformations in the program, the driver sends work to executors in the form of tasks
  - Tasks are run on executor processes to compute and save results
  - If the driver's main() method exits or it calls SparkContext.stop(), it will terminate the executors and release resources from the cluster manager

Deploying Applications in Spark

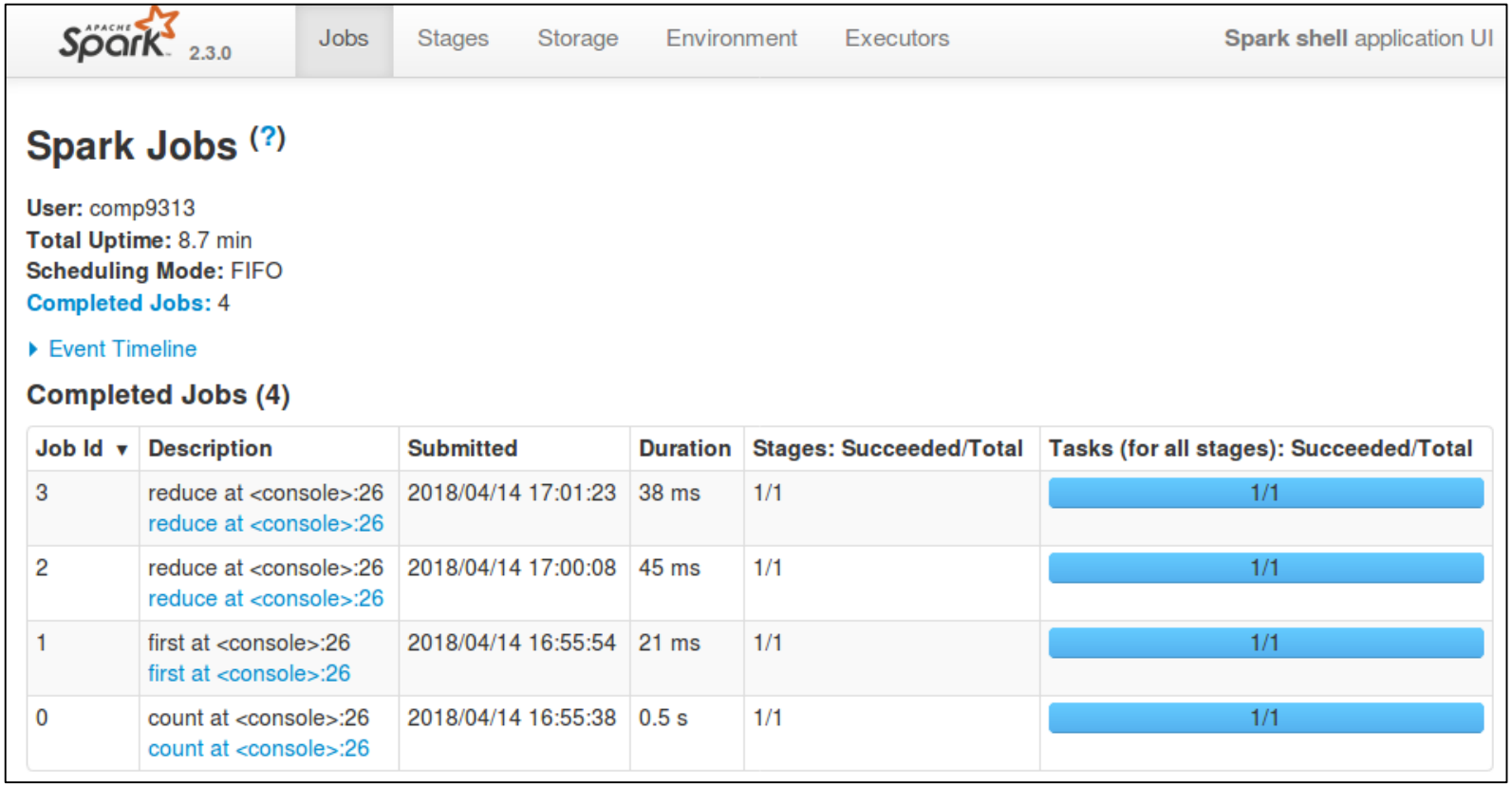
- spark-submit

Common flags	Explanation
--master	Indicates the cluster manager to connect to
--class	The “main” class of your application if you’re running a Java or Scala program
--name	A human-readable name for your application. This will be displayed in Spark’s web UI.
--executor-memory	The amount of memory to use for executors, in bytes. Suffixes can be used to specify larger quantities such as “512m” (512 megabytes) or “15g” (15 gigabytes)
--driver-memory	The amount of memory to use for the driver process, in bytes.

➢ spark-submit --master spark://hostname:7077 \
--class YOURCLASS \
--executor-memory 2g \
YOURJAR "options" "to your application" "go here"

Spark Web Console

- You can browse the web interface for the information of Spark Jobs, storage, etc. at: **http://localhost:4040**



Questions?