COMP9313: Big Data Management

Introduction to MapReduce and Spark

Motivation of MapReduce

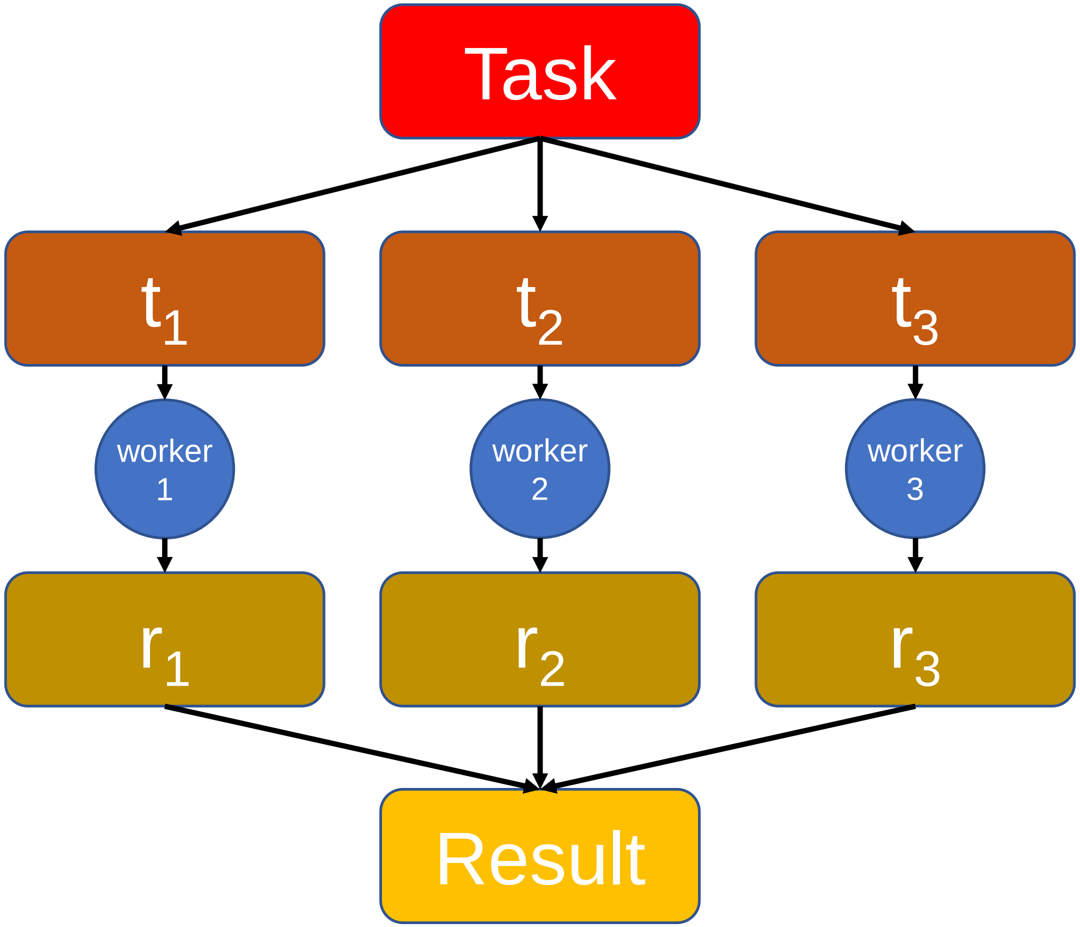
- Word count
  - output the number of occurrence for each word in the dataset.
- Naïve solution:

```
word_count(D):
    H = new dict
    For each w in D:
        H[w] += 1
    For each w in H:
        print (w, H[w])
```

- How to speed up?

Motivation of MapReduce

- Make use of multiple workers



There are some problems…

- Data reliability
- Equal split of data
- Delay of worker
- Failure of worker
- Aggregation the result

- We need to handle them all! In traditional way of parallel and distributed processing.

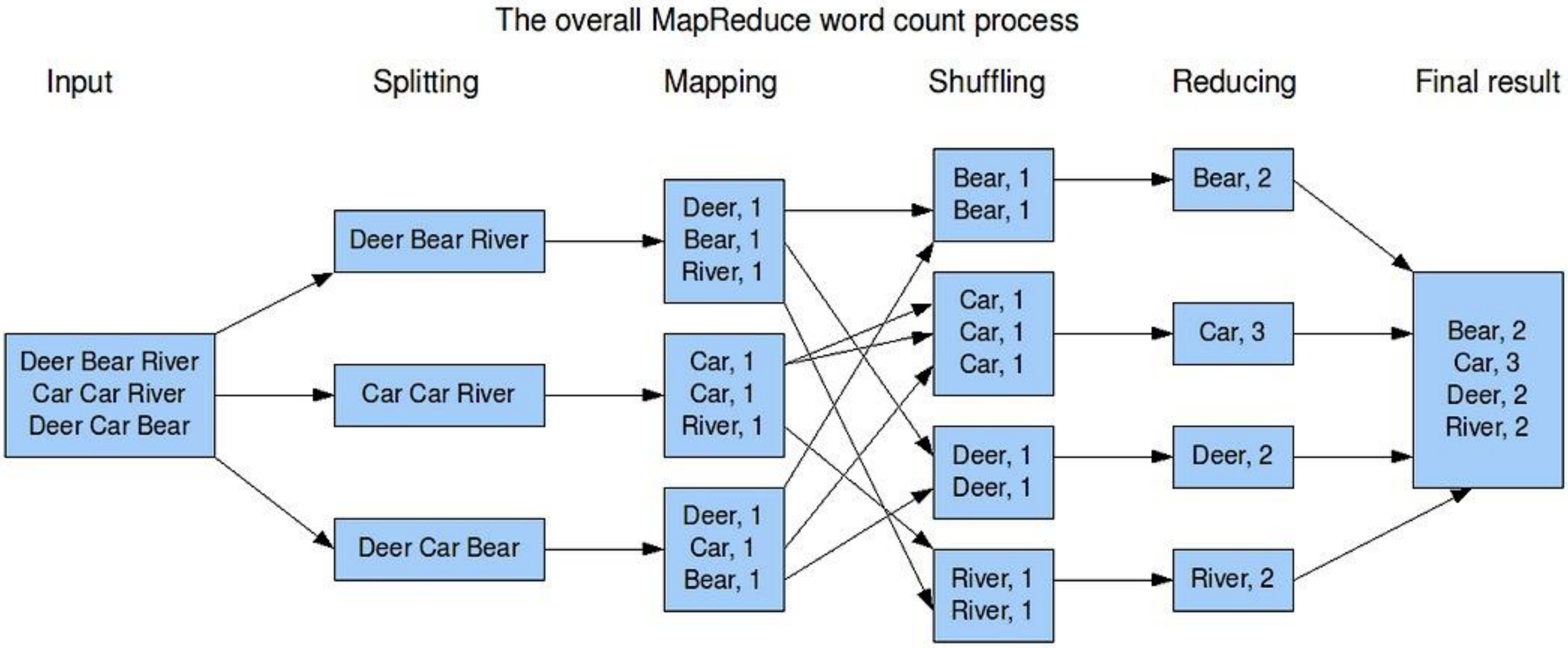
MapReduce

- MapReduce is a programming framework that
  - allows us to perform distributed and parallel processing on large data sets in a distributed environment
  - no need to bother about the issues like reliability, fault tolerance etc
  - offers the flexibility to write code logic without caring about the design issues of the system

Map Reduce

- MapReduce consists of Map and Reduce
- Map
  - Reads a block of data
  - Produces key-value pairs as intermediate outputs
- Reduce
  - Receive key-value pairs from multiple map jobs
  - aggregates the intermediate data tuples to the final output

A Simple MapReduce Example



Pseudo Code of Word Count

```
Map(D):
    for each w in D:
        emit(w, 1)


Reduce(t, counts): # e.g., bear, [1, 1]
    sum = 0
    for c in counts:
        sum = sum + c
    emit (t, sum)
```

Advantages of MapReduce

- Parallel processing
  - Jobs are divided to multiple nodes
  - Nodes work simultaneously
  - Processing time reduced

- Data locality
  - Moving processing to the data
    - Opposite from traditional way

We will discuss more on MapReduce, but not now…

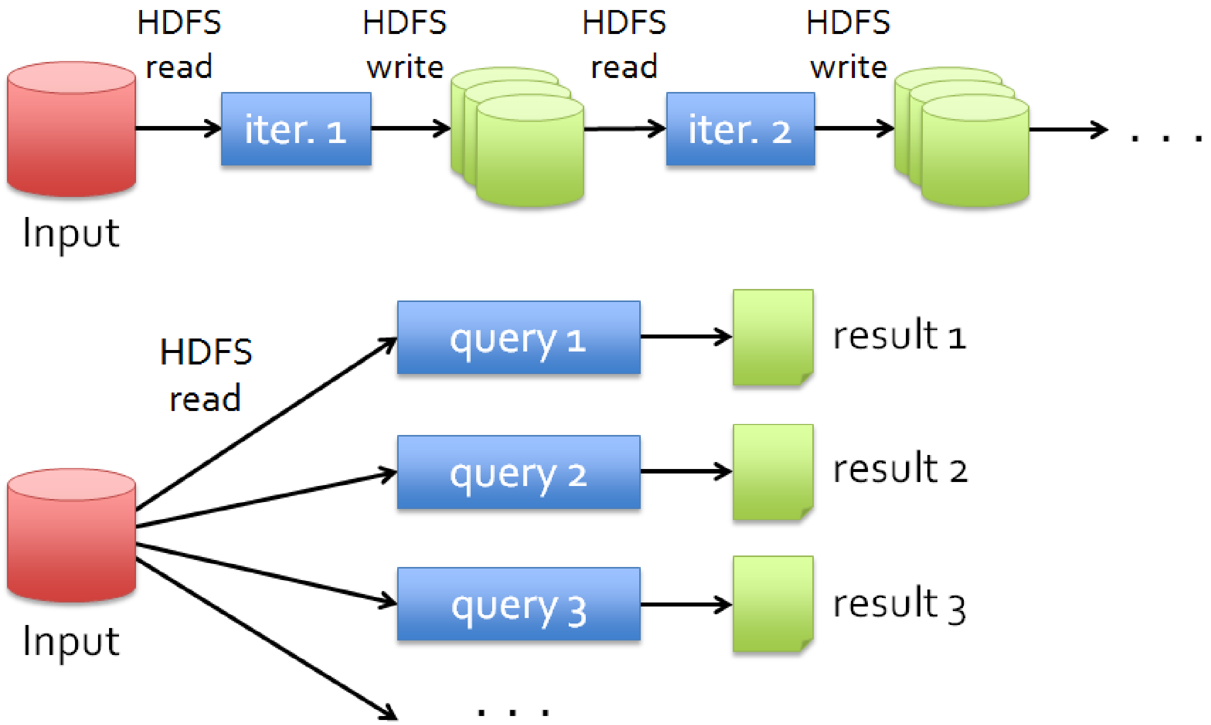
Motivation of Spark

- MapReduce greatly simplified big data analysis on large, unreliable clusters. It is great at one-pass computation.
- But as soon as it got popular, users wanted more:
  - more **complex**, multi-pass analytics (e.g. ML, graph)
  - more **interactive** ad-hoc queries
  - more **real-time** stream processing

Limitations of MapReduce

- As a general programming model:
  - more suitable for one-pass computation on a large dataset
  - hard to compose and nest multiple operations
  - no means of expressing iterative operations
- As implemented in Hadoop
  - all datasets are read from disk, then stored back on to disk
  - all data is (usually) triple-replicated for reliability

Data Sharing in Hadoop MapReduce



- **Slow** due to replication, serialization, and disk IO
- Complex apps, streaming, and interactive queries all need one thing that MapReduce lacks:
  - Efficient primitives for **data sharing**

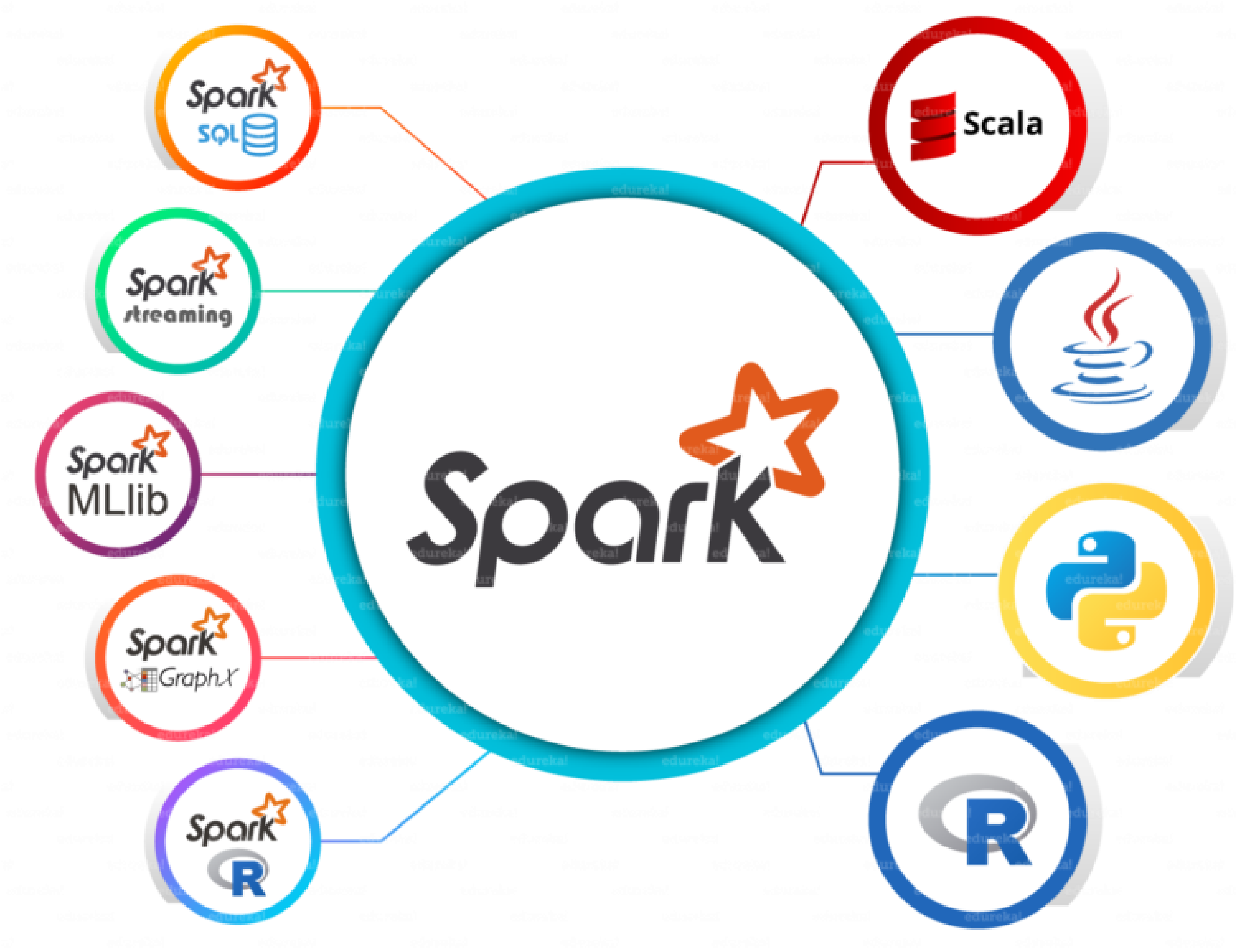
What is Spark?

- Apache Spark is an open-source cluster computing framework for real-time processing.
- Spark provides an interface for programming entire clusters with
  - implicit data parallelism
  - fault-tolerance
- Built on top of Hadoop MapReduce
  - extends the MapReduce model to efficiently use more types of computations

Spark Features

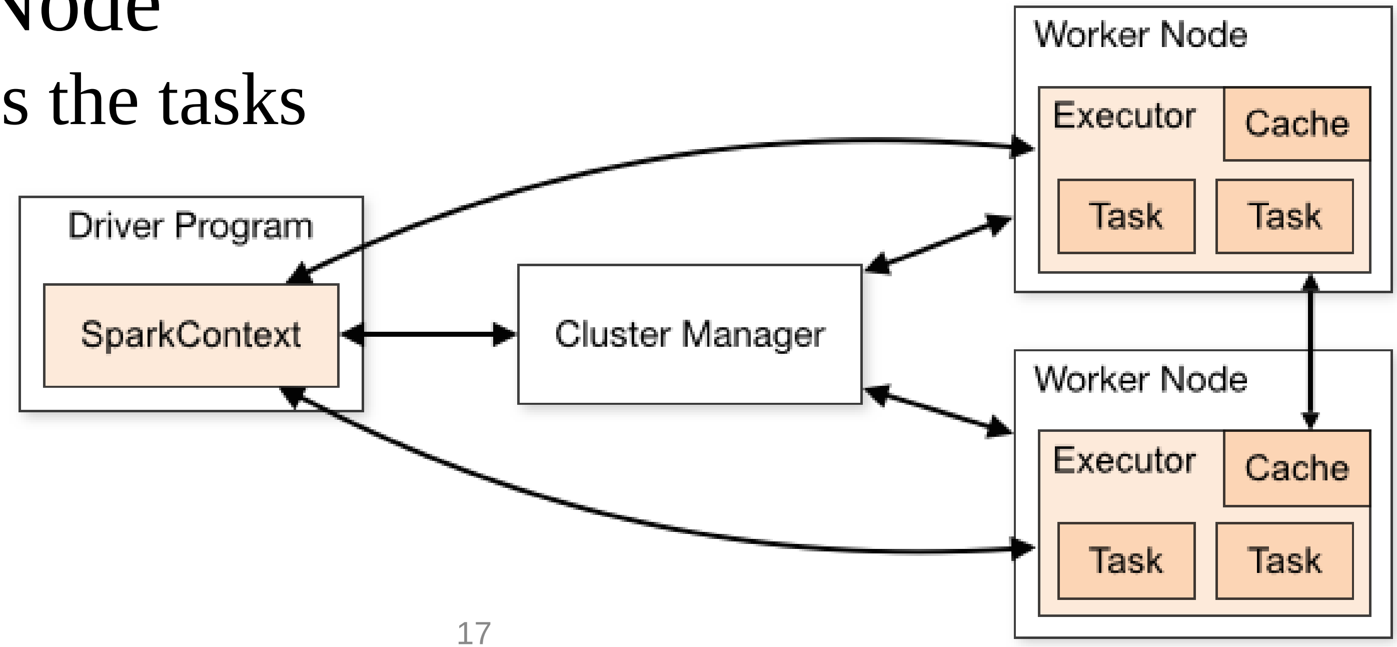
- Polyglot
- Speed
- Multiple formats
- Lazy evaluation
- Real time computation
- Hadoop integration
- Machine learning

Spark Eco-System



Spark Architecture

- Master Node
  - takes care of the job execution within the cluster
- Cluster Manager
  - allocates resources across applications
- Worker Node
  - executes the tasks



Resilient Distributed Dataset (RDD)

- RDD is where the data stays
- RDD is the fundamental data structure of Apache Spark
    - is a collection of elements
        - Dataset
    - can be operated on in parallel
        - Distributed
    - fault tolerant
        - Resilient

Features of Spark RDD

- In memory computation
- Partitioning
- Fault tolerance
- Immutability
- Persistence
- Coarse-grained operations
- Location-stickiness

Create RDDs

- Parallelizing an existing collection in your driver program
  - Normally, Spark tries to set the number of partitions automatically based on your cluster
- Referencing a dataset in an external storage system
  - HDFS, HBase, or any data source offering a Hadoop InputFormat
  - By default, Spark creates one partition for each block of the file

RDD Operations

- Transformations
  - functions that take an RDD as the input and produce one or many RDDs as the output
  - Narrow Transformation
  - Wide Transformation
- Actions
  - RDD operations that produce non-RDD values.
  - returns final result of RDD computations

Narrow and Wide Transformations

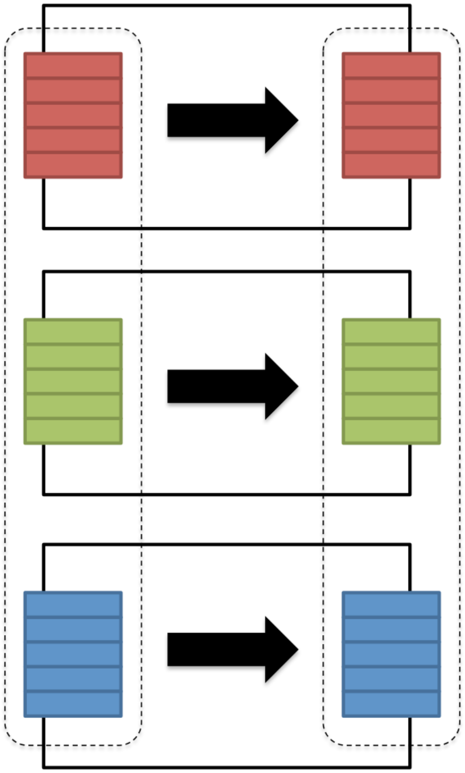
Narrow transformation involves no data shuffling

- map
- flatMap
- filter
- sample

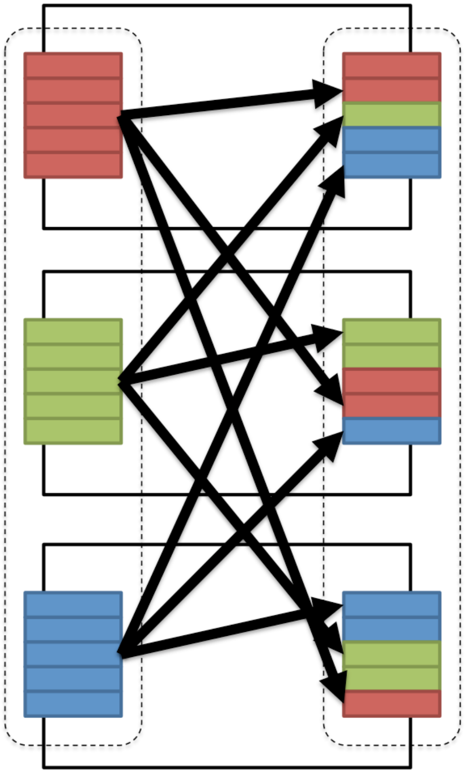
Wide transformation involves data shuffling

- sortByKey
- reduceByKey
- groupByKey
- join

Narrow transformation



Wide transformation



Action

- Actions are the operations which are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver
  - collect
  - take
  - reduce
  - forEach
  - sample
  - count
  - save

Lineage

- RDD lineage is the graph of all the ancestor RDDs of an RDD
  - Also called RDD operator graph or RDD dependency graph
- Nodes: RDDs
- Edges: dependencies between RDDs

Fault tolerance of RDD

- All the RDDs generated from fault tolerant data are fault tolerant
- If a worker fails, and any partition of an RDD is lost
  - the partition can be re-computed from the original fault-tolerant dataset using the lineage
  - task will be assigned to another worker

DAG in Spark

- DAG is a direct graph with no cycle
  - Node: RDDs, results
  - Edge: Operations to be applied on RDD
- On the calling of Action, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the task
- DAG operations can do better global optimization than other systems like MapReduce

DAG, Stages and Tasks

- DAG Scheduler splits the graph into multiple stages
- Stages are created based on transformations
  - The narrow transformations will be grouped together into a single stage
  - Wide transformation define the boundary of 2 stages
- DAG scheduler will then submit the stages into the task scheduler
  - Number of tasks depends on the number of partitions
  - The stages that are not interdependent may be submitted to the cluster for execution in parallel

Lineage vs. DAG in Spark

- They are both DAG (data structure)
- Different end nodes
- Different roles in Spark