Big Data

Spark Transformations

Spark Transformations

❖ Transformations

Transformation	Returned result of the Transformation
map(func)	Pass each element of the source data through function func
filter(func)	Select elements which are true based on function func
flatMap(func)	Maps each input item to 0 or other output items (Returns a sequence Seq)
mapPartitions(func)	Map running function func separately on each RDD partition/block

Spark Transformations

❖ Transformations

Transformation	Returned result of the Transformation
mapPartitionsWithIndex(func)	Run mapPartitions and provide the partition's index value
Sample(withReplacement, fraction, seed)	Sample data based on fraction using the generator seed with/without replacement
union(otherDataset)	Union of the source dataset and otherDataset elements
intersection(otherDataset)	Intersection of the source dataset and otherDataset elements

Spark Transformations

❖ Transformations

Transformation	Returned result of the Transformation
distinct([numTasks]))	Distinct elements of the source dataset
groupByKey([numTasks])	(K, V) pairs transformed into (K, Iterable<V>) pairs
reduceByKey(func, [numTasks])	(K, V) pairs transformed into (K, V) pairs with each key is aggregated with Reduce function func

Spark Transformations

❖ Transformations

Transformation	Returned result of the Transformation
join(otherDataset, [numTasks])	(K, V) and (K, W) pairs transformed into (K, (V, W)) pairs (Other join options: leftOuterJoin, rightOuterJoin, fullOuterJoin)
cogroup(otherDataset, [numTasks])	(K, V) and (K, W) pairs transformed into (K, (Iterable<V>, Iterable<W>)) tuples (a.k.a. groupWith)

Spark Transformations

❖ Transformations

Transformation	Returned result of the Transformation
cartesian(otherDataset)	Datasets of types T and U are transform ed into a dataset of (T, U) element pairs
pipe(command, [envVars])	Each RDD partition is pipelined through a shell command (Pipeline is chain of sequential processes)
coalesce(numPartitions)	Number of RDD partitions are reduced to numPartitions (Used to filter down a large dataset)

Spark Transformations

❖ Transformations

Transformation	Returned result of the Transformation
repartition(numPartitions)	Number of RDD partitions are changed to numPartitions and randomly reshuffle the RDD data to create a data balancing effect among the partitions
repartitionAndSortWithinPartitions (partitioner)	Number of RDD partitions are changed to numPartitions and sort records based on their keys within each partition

Big Data

Spark Actions

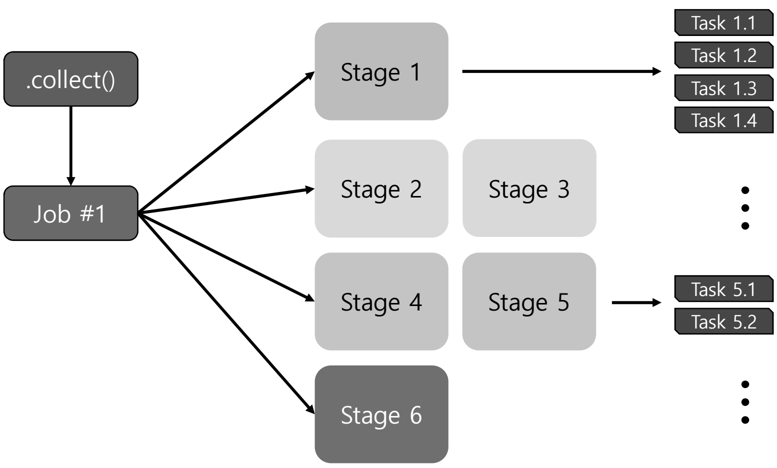
Spark Actions

❖ When an Action is executed

- Job will start to run multiple Stages (both serial and parallel)
- Each Stage will execute multiple Tasks in parallel on each partition of the RDD

Spark Actions

❖ Action is executed



Spark Actions

❖ Actions

Action	Returned result of the Action
reduce(func)	Aggregate dataset using a function func (takes 2 arguments and returns 1). For parallel processing, a commutative and associative func is recommended
collect()	Creates an array of the dataset. Useful after operations with small data subset outputs (e.g., filter)

Spark Actions

❖ Actions

Action	Returned result of the Action
count()	Counts the number of elements in the dataset
take(n)	Creates an array of the first n elements of the dataset
first()	Selects the first element of the dataset (same as take(1))
takeSample (withReplacement, num, [seed])	Creates an array with (num number of) random samples of the dataset (with/without replacement, with random number generator seed option)

Spark Actions

❖ Actions

Action	Returned result of the Action
takeOrdered(n, [ordering])	Selects the first n elements of the RDD (in natural order or using a custom comparator)
saveAsTextFile(path)	Creates a text file (or set within the file) of the dataset elements (in the local filesystem or Hadoop-supported file system)
saveAsSequenceFile(path)	Write the elements of the dataset as a Hadoop SequenceFile in a given path (in the local filesystem or Hadoop supported file system)

Spark Actions

❖ Actions

Action	Returned result of the Action
saveAsObjectFile(path)	Apply Java serialization and write to the dataset in a given path (in the local files ystem or Hadoop-supported file system)
countByKey()	Creates a hashmap of (K, Int) pairs with the count of each key.
foreach(func)	For each element of the dataset function func is applied

Big Data

Spark DAG

Spark DAG

❖ DAG (Directed Acyclic Graph)

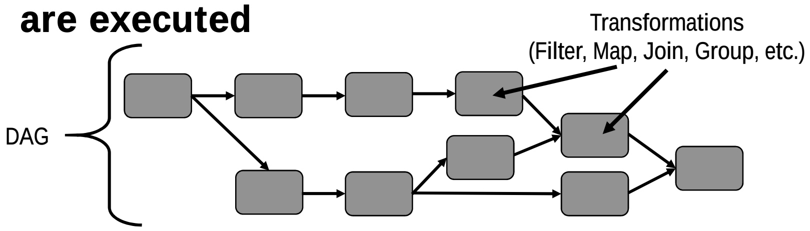
A. Sequence of Serial & Parallel computations to be conducted on a RDD

B. Computation sequence of Transformations (resulting in a chain of Parent RDD to Child RDD relations) are represented using a lineage graph

Spark DAG

❖ **Lazy Transformations**

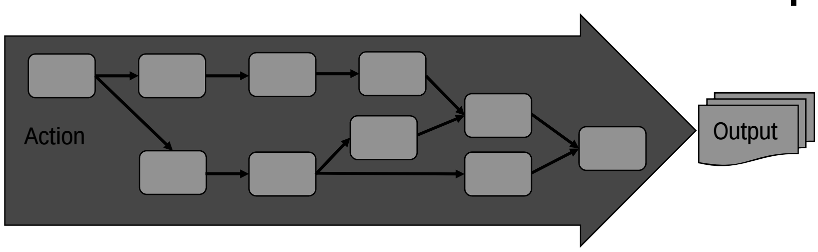
1. **Transformations/operators in Spark are executed Lazy**
2. **While the DAG (Directed Acyclic Graph) is being setup, none of the Transformations are executed**



Spark DAG

❖ **Lazy Transformations**

3. **When an Action is executed then**
   A. **Scheduler finds an optimal way to execute the DAG**
   B. **All transformations of the DAG Lineage Graph are executed**
4. **Results of the Action are delivered to the Output**



Spark Scheduling

❖ **Hadoop processes example**

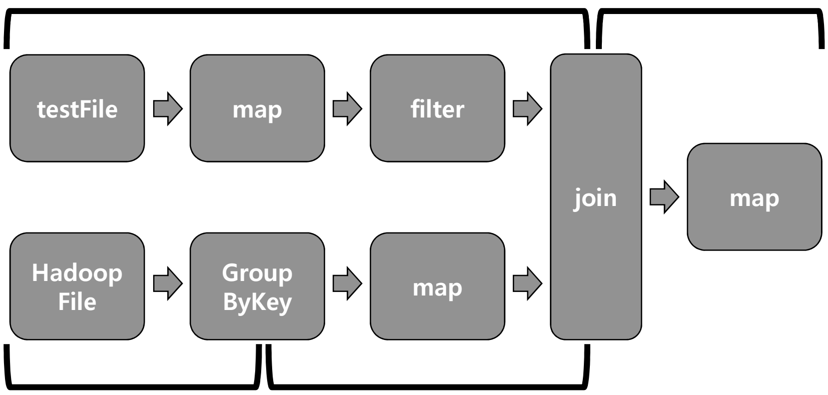
- **Read from HDFS ➔ Map ➔**
  ➔ Combine (Join, Shuffle) ➔
  ➔ Partition ➔ Reduce ➔
  ➔ Write back to HDFS

❖ **Spark DAG & Stage example**

- **GroupByKey and Join require a Shuffle, so they form a Stage boundary, where Map and Filter are Pipelined**

Spark Scheduling

❖ **Spark DAG Stages & Pipeline processes**



Spark DAG

❖ **Advantages of Lazy Transformations**

- **All optimized data processing methods are decided after the scheduler can check the entire DAG (Directed Acyclic Graph) sequence of work that needs to be done**
- **High efficiency**

Spark DAG

❖ **Advantages of Lazy Transformations**

- **Unnecessary operations are avoided**
- **Processing capability can be more effectively shared**
- **Reduced memory consumption**

Spark DAG

❖ RDD DAG Computation Example

1. Action is called on the RDD
2. Spark creates a DAG and submits it to the DAG scheduler
3. DAG scheduler divides operators into Stages of Tasks
   - Each stage is comprised of tasks based on partitions of the RDD input data

Spark DAG

❖ RDD DAG Computation Example

4. DAG scheduler pipelines operators together
   - Multiple operators can be scheduled in a single stage
   - Final result of a DAG scheduler is a set of stages to process
5. Stages are passed on to the Task Scheduler

Spark DAG

❖ **RDD DAG Computation Example**

6. **Task scheduler launches Tasks through the Cluster Manager (Mesos, YARN, Spark Standalone)**
   - **Task scheduler doesn't know about dependencies of the stages**
7. **Worker executes the Tasks on the Slave**

Spark DAG

❖ **RDD Resilience**

- **Read-only collection of objects partitioned across a set of machines**
- **RDD dataset can be cached across multiple nodes and reuse the dataset in multiple parallel operations**
- **Lineage based Fault Tolerance**
  - **If a partition is lost, RDD Lineage is used to rebuild just the lost/erroneous partition**

Spark DAG

❖ Transformation Types

- **Narrow Transformation**
  - Data is only shuffled within the partition
  - Narrow Transformation Examples
    - Map, Filter, Union, etc.
- **Wide Transformation**
  - Data is shuffled across different partitions
  - Wide Transformation Examples
    - groupByKey, reduceByKey, etc.

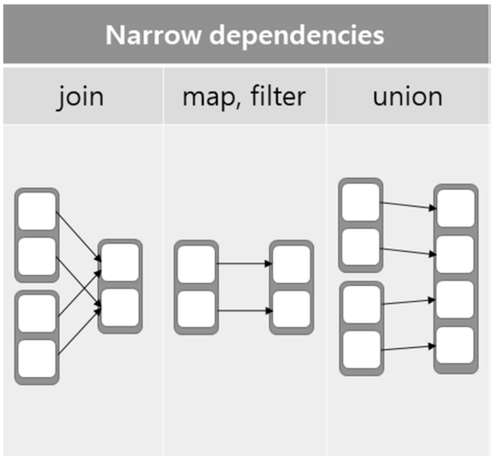
Spark DAG

❖ Lineage Dependency

- **Narrow Dependency**
  - Each Child partition has a 1-to-1 relation with a single partition in the Parent RDD
  - Parent RDD partitions are each used by one (or none) Child partition
- **Wide Dependency**
  - Child partitions have a 1-to-Many relation with multiple partitions of the Parent RDD
  - Parent RDD partitions are used in multiple Child partitions

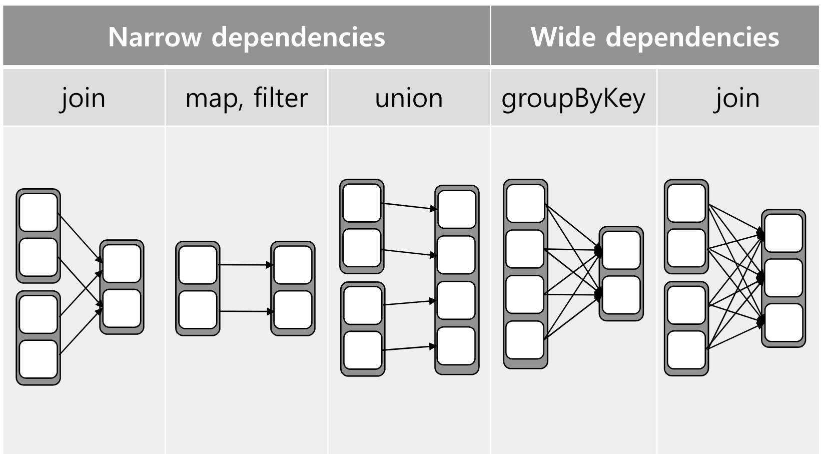
Spark DAG

❖ Lineage Dependency



Spark DAG

❖ Lineage Dependency



Big Data

References

References

- Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia, *Learning Spark: Lightning-Fast Big Data Analysis*. 1st Edition. O'Reilly, 2015.
- Sameer Farooqui, Databricks, **Advanced Apache Spark Training**, Devops Advanced Class, Spark Summit East 2015, http://slideshare.net/databricks, www.linkedin.com/in/blueplastic, March 2015.
- Apache Spark documents (all documents and tutorials were used)
  - http://spark.apache.org/docs/latest/rdd-programming-guide.html
  - http://spark.apache.org/docs/latest/rdd-programming-guide.html#working-with-key-value-pairs
  - https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#rdd-persistence
- Wikipedia, www.wikipedia.org
- Stackoverflow, https://stackoverflow.com/questions
- Bernard Marr, "Spark Or Hadoop -- Which Is The Best Big Data Framework?," Forbes, Tech, June 22, 2015.
- Quick introduction to Apache Spark, https://www.youtube.com/watch?v=TgiBvKcGL24
- Wide vs Narrow Dependencies, https://github.com/rohgar/scala-spark-4/wiki/Wide-vs-Narrow-Dependencies

References

- Partitions and Partitioning, https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-partitions.html
- Neo4j, "From Relational to Neo4j," https://neo4j.com/developer/graph-db-vs-rdbms/ (last accessed Jan. 1, 2018).

Image Sources

- By Robivy64 at English Wikipedia [Public domain], via Wikimedia Commons
- Teravolt at English Wikipedia [CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons
- By Konradr (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons