NYU Center for Data Science

Week 05.3: Spark

DS-GA 1004: Big Data

slido

How's lab 2 going?

ⓘ Start presenting to display the poll results on this slide.

Tips for Lab 2

(Esp. #5)

- Think from the end (last reducer) first
  - Top-down, not bottom-up
- Identify the minimum information your reducer (or mapper) will need
  - ⇒ maximize parallelism!
- Analyze your steps
  - How many mappers/reducers will run?
  - How much data does each see? Generate?
  - How does this change with
    - # of docs?
    - # of words?
  - How does it compare to the naive algorithm?

Previously…

1. Distributed storage
2. The Hadoop distributed file system (HDFS)

   $ hadoop fs -command …

3. HDFS and Map-reduce

This week

1. Spark
2. RDDs
3. Spark-SQL

Resilient distributed datasets (RDDs)

- RDD:
  - **Data source**
  - Lineage graph of **transformations** to apply to **data**
  - + interfaces for data **partitioning** and **iteration**

- Think of this as **deferred computation**
  - Nothing is **computed** until you ask for it
  - Nothing is **saved** until you say so
  - This makes optimization possible

Some notation:

RDD[T] denotes an RDD with data of type T, e.g.

- RDD[String]
- RDD[Tuple(String, Float)]

slido

Which RDBMS structure is most similar to a Spark RDD?

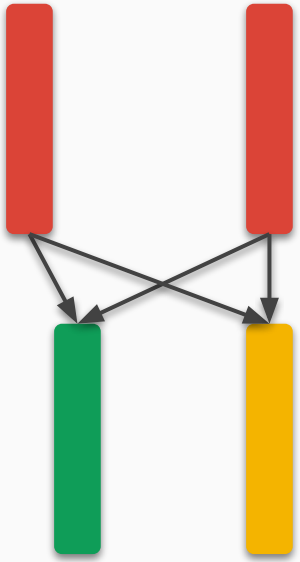
ⓘ Start presenting to display the poll results on this slide.

RDDs are more than columns

- RDDs can be derived from other RDDs through **transformations**
- RDDs also expose **partition** information, which influences how data is stored in HDFS
- Since Spark 2.0, one or more RDDs can be collected to form DataFrames
  - You could have an RDD with compound type, but DataFrames are more convenient

Lineage graphs

- It’s called a “lineage graph”, but it need not be linear!
- **Any RDD** can depend on **multiple parent RDDs**
- Once a **parent RDD** has been computed,
  it can be cached and reused by **multiple descendents**!



Transformations

Transformations turn one or more RDDs into a new RDD

Transformations are cheap to construct because they don't actually do the computation

Building an RDD is like **writing** (not ***running***) a map-reduce script or a SQL query

- Examples:
  - **map**(**function** T → U) ⇒ RDD[T] → RDD[U]
  - **filter**(**function** T → **Boolean**) ⇒ RDD[T] → RDD[T]
  - **union**() ⇒ (RDD[T], RDD[T]) → RDD[T]

```
lines = spark.textFile("hdfs://…")

errors = lines.filter(_.startsWith("ERROR"))

errors.filter(_.contains("MySQL"))
      .map(_.split('\t')(3))
      .collect()
```

Actions

Actions are what execute computation defined by an RDD

Results of actions are not RDDs

- Examples:
  - **count**() ⇒ RDD[T] → Integer
  - **collect**() ⇒ RDD[T] → Sequence[T]
  - **reduce**(**function** (T, T) → T) ⇒ RDD[T] → T
  - **save**(**path**) ⇒ Save RDD to file system or HDFS

```
lines = spark.textFile("hdfs://…")

errors = lines.filter(_.startsWith("ERROR"))

errors.filter(_.contains("MySQL"))
      .map(_.split('\t')(3))
      .collect()
```

slido

Map-Reduce requires both map and reduce to be deterministic functions. Is the same true of Spark transformations?

ⓘ Start presenting to display the poll results on this slide.

slido

[2] Map-Reduce requires both map and reduce to be deterministic functions. Is the same true of Spark transformations?

ⓘ Start presenting to display the poll results on this slide.

Determinism in Spark

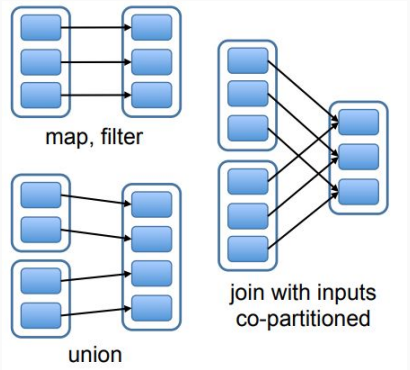
- Transformations **need to be deterministic** functions
- Otherwise, **caching, reuse, and reconstruction** of an RDD from its lineage graph cannot be guaranteed to be correct
- What problems could this present? When would randomization be helpful?

Narrow and wide dependencies

Narrow dependencies

Partition of parent RDD goes to at most 1 partition of child RDDs

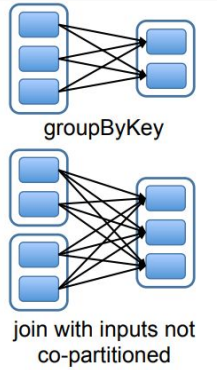
- Low communication
- Localized
- Easy to pipeline
- Easy failure recovery



Wide dependencies

Partition of parent RDD goes to multiple child RDD partitions

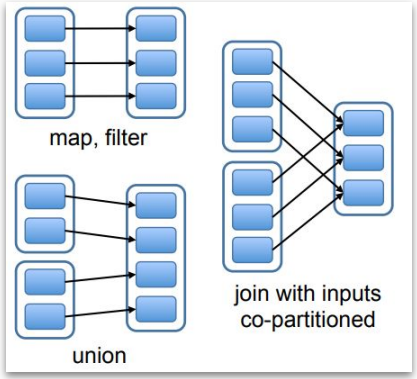
- High communication
- High latency
- Difficult to pipeline
- Difficult to recover



Figures adapted from [Zaharia et al., 2012]

Repartitioning

- Sometimes you know in advance which columns of a DataFrame will be filtered
  - E.g., dates or timestamps
- You can give hints to Spark that RDD partitions should align accordingly
  - df.repartition(# PARTITIONS, col("NAME OF COLUMN"))
  - This can reduce the width of RDD dependencies
- **This is much like indexing in RDBMS**



DataFrames and RDDs

- When using RDBMS (and DataFrames) we often think of data in **rows**
  - 1 row = 1 record
  - ~= 1 line in a CSV
- DataFrames are implemented as **collections of RDDs**
  - 1 column = 1 RDD
- As a user, this doesn't change much, but it does change how we think about **storage** ⇒ next week's topic!

Spark-SQL

- Spark 2.x allows you to express queries in SQL
  - Or using an object-method chaining API -- the two are equivalent!
- Queries are executed against DataFrames
  - DataFrames are secretly RDDs, not RDBMS tables!
- Queries can be optimized by analyzing the RDD lineage graph

```
df.createOrReplaceTempView('my_table')

res = spark.sql('  SELECT zip_code, sum(height) as H
                   FROM my_table
                   GROUP BY zip_code')
res.show()
```

```
df.groupBy('zip_code')
  .sum('height')
  .withColumnRenamed('sum(height)', 'H')
  .show()
```

slido

Think back to DeWitt & Stonebreaker. Which of their criticisms of MapReduce do you think also apply to Spark?

ⓘ Start presenting to display the poll results on this slide.

MapReduce criticisms and Spark

- Too low-level?
  - Complex analyses can be written easily
  - Spark-SQL is high level
- Poor implementation?
  - Query optimization doesn't replace indexing, but it can help
- Missing RDBMS features?
  - Partitioning is kind of like indexing, but not exactly
  - We do have schemas though!
  - Transactions: less relevant due to read-only data
- RDBMS compatibility
  - Spark isn't an RDBMS ¯_(ツ)_/¯
  - But integration with other frameworks that speak in DataFrames is getting better

Wrap-up on Spark

- RDD framework is more flexible than Map-Reduce
- Caching can make interactive jobs faster
- SparkSQL / DataFrames API makes development easy
- This will be Lab 3

Next week

Back to storage!

- Column-Oriented Storage
- Dremel & Parquet
  - Parquet is the default storage format for Spark

slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.