Apache Spark

Advanced I/O Concepts

Splittable File Types and Compression

- Certain file formats are fundamentally “splittable.”
- This can improve speed because it makes it possible for Spark to avoid reading an entire file, and
  - access only the parts of the file necessary to satisfy your query
- Additionally if you’re using something like HDFS, splitting a file can provide further optimization if that file spans multiple blocks
- How you store your data is of immense consequence when it comes to making your Spark jobs run smoothly
- We recommend Parquet with gzip compression

Reading Data in Parallel

- Multiple executors cannot read from the same file at the same time, but
- They can read different files at the same time
- This means that when you read from a folder with multiple files in it, each one of those files will become a partition in your DataFrame and be read in by available executors in parallel

Writing Data in Parallel

- The number of files or data written is dependent on the number of partitions the DataFrame has
- By default, one file is written per partition of the data
- This means that although we specify a “file,” it’s actually a number of files within a folder, with the name of the specified file
- For example, the following code
  - csvFile.repartition(5).write.format("csv").save("/tmp/multiple.csv")
- will end up with five files inside of that folder. As you can see from the list call:
  - /tmp/multiple.csv/part-00000-767df509-ec97-4740-8e15-4e173d365a8b.csv
  - /tmp/multiple.csv/part-00001-767df509-ec97-4740-8e15-4e173d365a8b.csv
  - /tmp/multiple.csv/part-00002-767df509-ec97-4740-8e15-4e173d365a8b.csv
  - /tmp/multiple.csv/part-00003-767df509-ec97-4740-8e15-4e173d365a8b.csv
  - /tmp/multiple.csv/part-00004-767df509-ec97-4740-8e15-4e173d365a8b.csv

Managing File Size

- An important factor not so much for writing data but reading it later on.
- When you're writing lots of small files, there's a significant metadata overhead that you incur managing all of those files
- Spark especially does not do well with small files, although many file systems (like HDFS) don't handle lots of small files well, either. You might hear this referred to as the "**small file problem.**"
- The opposite is also true: you don't want files that are too large
  - either, because it becomes inefficient to have to read entire blocks of data when you need only a few rows.

Managing File Size

- Spark 2.2+ introduced a new method for controlling file sizes
- You can use the **maxRecordsPerFile** option
- This allows you to better control file sizes by controlling the number of records that are written to each file
- For example, if you set an option for a writer as
  - df.write.option("maxRecordsPerFile", 5000)
- Spark will ensure that files will contain at most 5,000 records

Thanks