Data Storage Tips for Optimal Spark Performance

Vida Ha

Spark Summit West 2015

Today's Talk

About Me

- Vida Ha - Solutions Engineer at Databricks

Poor Data File Storage Choices Result in:

- Exceptions that are difficult to diagnose and fix.
- Slow Spark Jobs.
- Inaccurate Data Analysis.

Goal: Understand the Best Practices for Storing and working with Data in Files with Spark

Agenda

- **Basic Data Storage Decisions**
  - File Sizes
  - Compression Formats
- **Data Format Best Practices**
  - Plain Text, Structured Text, Data Interchange, Columnar
- **General Tips**

Basic Data Storage Decisions

Choosing a File Size

- **Too Small**
  - Lots of time opening file handles.
- **Too Big**
  - Files need to be splittable.
- **Optimize for your File System**
- **Good Rule of Thumb: 64 MB - 1 GB**

Choosing a Compression Format

- **The Obvious**
  - Minimize the Compressed Size of Files.
  - Decoding characteristics.
- **Pay attention to Splittable vs. NonSplittable.**
- **Common Compression Formats for Big Data:**
  - gzip, Snappy, bzip2, LZO, and LZ4.
  - Columnar formats for Structured Data - **Parquet.**

Data Format Best Practices

Plain Text

- **sc.textFile() splits the file into lines.**
  - So keep your lines a reasonable size.
  - Or use a different method to read data in.
- **Keep file sizes < 1 GB if compressed with a non-splittable format.**

Basic Structured Text Tiles

- **Use Spark transformations to ETL the data.**
- **Optionally use Spark SQL for analyzing.**
- **Handle the inevitable malformed data with Spark:**
  - Use a filter transformation to drop bad lines.
  - Or use a map function to fix bad lines.
- **Includes CSV, JSON, XML.**

CSV

- **Use the relatively new Spark-CSV package.**
- **Spark SQL Malformed Data Tip:**
  - Use a where clause and sanity check fields.

JSON

- **Ideally have one JSON object per line.**
  - Otherwise, DIY parsing the JSON.
- **Prefer specifying a schema over inferSchema.**
- **Watch out for arbitrary number of keys.**
  - Inferring Schema will result in an executor OOM error.
- **Spark SQL Malformed Data Tip:**
  - Bad inputs have a column called _corrupt_record and other columns will be null.

XML

- **Not an ideal Big Data Format.**
  - Typically not one XML object per line.
  - So you'll need to DIY parse.
- **Not a lot of built in library support for XML.**
- **Watch out for very large compressed files.**
- **May need to employ the General Tips to parse.**

Data Interchange Formats with a Schema

- **Good Practice to enforce an API with backward compatibility.**
  - Avro, Parquet, and Thrift are common ones.
- **Usually, good compression.**
- **Data format itself is not corrupt.**
  - But underlying records still be.

Avro

- **Use DataFile Writer to write multiple objects.**
- **Use spark-avro package to read in.**
- **Don't transfer "AvroKey" across driver and workers.**
  - AvroKey is not serializable.
  - Pull out fields of interest or convert to JSON.

Protocol Buffers

- **Need to figure out how to encode multiple in a file.**
  - Encode in Sequence Files or other similar file format.
  - Prepend the number of bytes before reading the next message.
  - Base64 encode one per line.
- **Currently, no built in Spark package to support.**
  - Opportunity to contribute to open source community.
  - For now, convert to JSON and read into Spark SQL.

Columnar Formats: Parquet & ORD

- **Great for use with Spark SQL.**
  - Parquet is actually best practice for Spark SQL.
- **Makes it easy to pull only certain records at a time.**
- **Worth time to encode if multiple passes on data.**
- **Do not support appending one record at a time.**
  - Not good for collecting log records.

General Tips

Reuse Hadoop Libraries

- **HadoopRDD & NewHadoopRDD**
  - Reuse Hadoop File format libraries.
- **Hive SerDe's are supported in Spark SQL**
  - Load your SerDe jar onto your Spark Cluster.
  - Issue a SHOW CREATE TABLE command.
  - Enter the create table command (with EXTERNAL)

Controlling the Output File Size

- **Spark generally writes one file per partition.**
- **Use coalesce to write out less files.**
- **Use repartition to write out more files.**
  - This will cause a shuffle of the data.
- **If repartition is too expensive:**
  - Call mapPartition to get an iterator and write out as many (or few) files as you would like.

sc.wholeTextFiles()

- **The whole file should fit into memory.**
- **Good for file formats that aren't splittable by line.**
  - Such as XML files.
- **Will need to performance tune.**

Use “filename” as the RDD element

- **filenames = sc.parallelize([“s3:/file1”, “s3:/file2”, …])**
- **Allows you to function on a single file.**
  - **filenames.map(…call_function_on_filename)**
- **Use to decode non-standard compression formats.**
- **Use to split up files that aren’t separable by line.**

Thank you.

Any questions?