Caching

CLEANING DATA WITH APACHE SPARK IN PYTHON

Mike Metzger
Data Engineering Consultant

What is caching?

Caching in Spark:

- Stores DataFrames in memory or on disk

- Improves speed on later transformations / actions

- Reduces resource usage

Disadvantages of caching

- Very large data sets may not fit in memory

- Local disk based caching may not be a performance improvement

- Cached objects may not be available

Caching tips

When developing Spark tasks:

- Cache only if you need it

- Try caching DataFrames at various points and determine if your performance improves

- Cache in memory and fast SSD / NVMe storage

- Cache to slow local disk if needed

- Use intermediate files!

- Stop caching objects when finished

Implementing caching

Call `.cache()` on the DataFrame before Action

```
voter_df = spark.read.csv('voter_data.txt.gz')
voter_df.cache().count()
```

```
voter_df = voter_df.withColumn('ID', monotonically_increasing_id())
voter_df = voter_df.cache()
voter_df.show()
```

More cache operations

Check `.is_cached` to determine cache status

```
print(voter_df.is_cached)
```

```
True
```

Call `.unpersist()` when finished with DataFrame

```
voter_df.unpersist()
```

Let's Practice!

CLEANING DATA WITH APACHE SPARK IN PYTHON

Improve import performance

Mike Metzger
Data Engineering Consultant

Spark clusters

Spark Clusters are made of two types of processes

- Driver process

- Worker processes

Import performance

Important parameters:

- Number of objects (Files, Network locations, etc)
  - More objects better than larger ones

  - Can import via wildcard

```
airport_df = spark.read.csv('airports-*.txt.gz')
```

- General size of objects
  - Spark performs better if objects are of similar size

Schemas

A well-defined schema will drastically improve import performance

- Avoids reading the data multiple times

- Provides validation on import

How to split objects

- Use OS utilities / scripts (split, cut, awk)

```
split -l 10000 -d largefile chunk-
```

- Use custom scripts

- Write out to Parquet

```python
df_csv = spark.read.csv('singlelargefile.csv')
df_csv.write.parquet('data.parquet')
df = spark.read.parquet('data.parquet')
```

Let's practice!

CLEANING DATA WITH APACHE SPARK IN PYTHON

Cluster sizing tips

CLEANING DATA WITH APACHE SPARK IN PYTHON

Mike Metzger
Data Engineering Consultant

Configuration options

- Spark contains many configuration settings

- These can be modified to match needs

- Reading configuration settings:

```
spark.conf.get(<configuration name>)
```

- Writing configuration settings

```
spark.conf.set(<configuration name>)
```

Cluster Types

Spark deployment options:

- Single node

- Standalone

- Managed
  - YARN

  - Mesos

  - Kubernetes

Driver

- Task assignment

- Result consolidation

- Shared data access

Tips:

- Driver node should have double the memory of the worker

- Fast local storage helpful

Worker

- Runs actual tasks

- Ideally has all code, data, and resources for a given task

Recommendations:

- More worker nodes is often better than larger workers

- Test to find the balance

- Fast local storage extremely useful

Let's practice!

CLEANING DATA WITH APACHE SPARK IN PYTHON

Performance improvements

CLEANING DATA WITH APACHE SPARK IN PYTHON

Mike Metzger
Data Engineering Consultant

Explaining the Spark execution plan

```
voter_df = df.select(df['VOTER NAME']).distinct()
voter_df.explain()
```

```
== Physical Plan ==
*(2) HashAggregate(keys=[VOTER NAME#15], functions=[])
+- Exchange hashpartitioning(VOTER NAME#15, 200)
   +- *(1) HashAggregate(keys=[VOTER NAME#15], functions=[])
      +- *(1) FileScan csv [VOTER NAME#15] Batched: false, Format: CSV, Location:
      InMemoryFileIndex[file:/DallasCouncilVotes.csv.gz],
      PartitionFilters: [], PushedFilters: [],
      ReadSchema: struct<VOTER NAME:string>
```

What is shuffling?

Shuffling refers to moving data around to various workers to complete a task

- Hides complexity from the user

- Can be slow to complete

- Lowers overall throughput

- Is often necessary, but try to minimize

How to limit shuffling?

- Limit use of `.repartition(num_partitions)`
  - Use `.coalesce(num_partitions)` instead

- Use care when calling `.join()`

- Use `.broadcast()`

- May not need to limit it

Broadcasting

Broadcasting:

- Provides a copy of an object to each worker

- Prevents undue / excess communication between nodes

- Can drastically speed up `.join()` operations

Use the `.broadcast(<DataFrame>)` method

```python
from pyspark.sql.functions import broadcast
combined_df = df_1.join(broadcast(df_2))
```

Let's practice!

CLEANING DATA WITH APACHE SPARK IN PYTHON