Shuffle Partitioning

Objective

Shuffling as something good that can increase parallelism

Deciding optimal shuffling based on cluster size

Partitioning

Determines the degree of parallelism in a job

- each task processes one partition

Determines the degree of I/O parallelism

Small partitions

- data I/O overhead
- large task launch overhead
- easy to recompute if executor dies

Large partitions

- more CPU usage for actual data processing
- few tasks/parallelism
- long time to process
- large amount of memory needed
- hard to recompute if executor dies

Shuffle Partitioning

How do we choose the optimal shuffle partitions?

Short answer: no silver bullet

Recommendations

- if intermediate data is large, increase shuffle partitions
- if you have idle cores, increase shuffle partitions
- if intermediate partitions are very small (<1MB), decrease shuffle partitions
- minimize shuffles altogether

Optimal partition size = 10 – 100MB of uncompressed data

```
spark.sql.shuffle.partitions = 1000
spark.default.parallelism = 100
```

Exercises

Thought experiments with a cluster

- a dataset X GB in size is being shuffled for a complex job
- shuffle is required, but job taking too long
- optimize the shuffle: pick the correct number of partitions to use

```
spark.sql.shuffle.partitions = ?
```

Things to keep in mind

- optimal partition size 10-100MB
- largest desired partition size 200MB
- CPU cores must not be idle

Exercise 1

Scenario

- you see a shuffle write of 210GB in the Spark UI
- job takes a long time
- you have 4 executors with 4 cores each

Recommendation

- your parallelism is 16, so at least 16 partitions
- largest partition recommended size 200MB, so at least 1050 partitions

```
spark.sql.shuffle.partitions = 1050
```

Result

- optimal time/task as partition is of optimal size
- 100% cluster utilization

Recommendation: allocate more CPU cores

Exercise 2

Scenario

- you see a shuffle write of 210GB in the Spark UI
- job takes a long time
- you have 200 executors with 8 cores each

Recommendation

- your parallelism is 1600, so at least 1600 partitions
- largest partition recommended size 200MB, so at least 1050 partitions

```
spark.sql.shuffle.partitions = 1600
```

Result

- partition size ~134MB, roughly perfect size
- 100% cluster utilization and high parallelism

Exercise 3

Scenario

- you see a shuffle write of ~1GB in the Spark UI
- job takes a long time
- you have 2 executors with 4 cores each

Recommendation

- your parallelism is 8, so at least 8 partitions
- smallest partition recommended size 10MB, so at most 100 partitions

```
spark.sql.shuffle.partitions = 10
```

Result

- partition size ~100MB, roughly perfect size
- 100% cluster utilization
- 100x overhead reduction from task creation compared to 10000 partitions => 3-10x perf boost

Spark rocks