LOAD BIG DATA EFFICIENTLY

PART 7: THE SMALL FILE PROBLEM

Filter and write

col1	col2
1	A
2	B
3	A

col1	col2
4	B
5	B
6	B

col1	col2
7	A
8	A
9	B

col1	col2
10	B
11	A
12	A

Driver

- Assigns data to partitions
- Delegates the work to executors, meaning partitions to tasks
- Create execution plans
- Saves meta data about the files

Worker 1

Core1 **Core2**

Worker 2

Core1 **Core2**

Reference: https://youtu.be/kCydZHkqXc0

Problem Cause of Small files

- File meta data stored in driver memory
- Efforts querying data including open file, closing file and checking the storage files and directories
- Scheduling overhead for delegating the partitions to tasks
- Reduced parallalism due to more created partitions
- Increased CPU costs due to serialisation and deserialisation

Reference: Small File, Large Impact — Addressing the Small File Issue in Spark | by Siraj | Medium

Solving the Problem

- Increase the file size. Either by checking the source options or having an intermediate process saving files in bigger files. I like Delta and you the bin packing (optimize) option
- Use file formats like Parquet or Avro with less meta data to be stored
- Reduce meta data by defining the schema
- Reduce number files per partition and thus increase number of partitions e.g. by using maxPartitionBytes or openCostInBytes

Experiment

- 10 Mio rows saved as 8 or 100.000 files
- Meaning 1,25 Mio vs 100 rows per file

- For JSON, CSV, PARQUET and AVRO we will compare:
  - Loading time small files vs big files
  - Schema vs no schema to reduce meta data

Reference: https://youtu.be/zRZ_XrNQl6Y

Write times

Format	Size	File size small	File Size Big	Small	Large
JSON	1218 MB	12,47 KB	152 MB	11 min	3 s
CSV	593 MB	6,07 KB	74 MB	8,3 min	3 s
PARQUET	82 MB	0,82 KB	10 MB	9,3 min	2 s
AVRO	69 MB	0,71 KB	9 MB	27 min	1 s

Results Absolute

Format	Experiment	Meta Time (S)	Loading Time (S)	Total (S)	Meta Time (L)	Loading Time (L)	Total (L)
JSON	w schema	21 s + 6,8 min	18 s	**7,7 min**	2 s	3 s	**6 s**
JSON	w/o auto schema	-	18 s	**18 s**	-	3 s	**3 s**
JSON	w target schema	-	21 s	**21 s**	-	6 s	**6 s**
CSV	w/o schema	19 s + 86 ms + 6,9 min	30 s	**7,7 min**	25 ms + 6s	6 s	**12 s**
CSV	w auto schema	-	32 s	**32 s**	-	6 s	**6 s**
CSV	w target schema	-	31 s	**31 s**	-	6 s	**6 s**
PARQUET	w/o schema	0,3 s	5,8 min	**5,8 min**	30 ms	0,6 s	**0,6 s**
PARQUET	w target schema	-	33 s	**33 s**	-	0,4 s	**0,4 s**
AVRO	w/o schema	-	5,1 min	**5,1min**	-	0,9 s	**0,9 s**
AVRO	w target schema	-	16 s	**16 s**	-	1 s	**1 s**

Results % to auto schema

Format	w/o vs with schema small	w/o vs with schema large	Small vs large w/o schema	Small vs large with schema
JSON	- 96 %	- 50 %	- 99 %	- 83 %
CSV	- 93 %	- 50 %	- 97 %	- 81 %
PARQUET	- 91 %	- 33 %	- 99,99 %	- 99 %
AVRO	- 95 %	- 0 %	- 99,99 %	- 94 %

Adjust Partition Size

Max Partition Size:

- Influences the size of a partition
- based on the config "spark.sql.files.maxPartitionBytes"
- defaults to 128 MB

Open Cost Per Bytes

- Represents the cost of creating a new partition
- based on the config "spark.sql.files.openCostInBytes"
- defaults to 4 MB
- Technically it adds the cost, e.g. 4 MB, to each file which is called padding
- Official description: The estimated cost to open a file, measured by the number of bytes that could be scanned in the same time. This is used when putting multiple files into a partition. It is better to over-estimate, then the partitions with small files will be faster than partitions with bigger files (which is scheduled first). This configuration is effective only when using file-based sources such as Parquet, JSON and ORC.

Reference: https://youtu.be/vkOxEHEKYhA

Results of open Cost In Bytes

OpenCost MB	JSON	AVRO
1 MB	5,8 min	5,5 min
2 MB	21 s	40 s
4 MB	21 s	41 s
6 MB	21 s	42 s
8 MB	22 s	2,1 min
10 MB	1,8 min	3,7 min
w/o schema 4 MB	7,7 min	5,1 min

Reference: https://youtu.be/C-ykod0Yyf4

Summary

- We saw that bigger files are faster than smaller
- Defining the schema can reduce especially for smaller files the overhead significantly
- The openCostsInBytes parameter can influence the loading time by adding overhead costs to the file sizes and reduce number of files per partition/task