Spark optimization project report part 2

Summary

Here I tried using some other techniques to optimize the Spark code: broadcasting, caching, repartition and window function

As the Spark configuration parameter "spark.sql.shuffle.partitions" has a significant impact on the performance, I did experiments for two scenarios: set it as the default value, which is 200 or set it as 4

As a result, I found the caching solution has a similar performance as the original one but the three other solutions have performance gain in one of the two scenarios: "spark.sql.shuffle.partitions" = 4 or "spark.sql.shuffle.partitions" =200.

	"spark.sql.shuffle.partitions"= 4	"Spark.sql.shuffle.partitions" =default value (200)
Broadcasting	Similar	Broadcasting is much faster and **takes** ⅛ ~¼ time of the original
Caching	Similar	Similar
Repartition	Repartition **saves** ¼ ~ ⅓ time than the original	Similar
Window Function	Similar	Window Function is much faster and **takes** about ¼ time of the original

I. Experiments

Here are the detailed results of the experiments. The scripts are in subfolder 'scripts for report2'.

1. Broadcasting

- Code enhanced

```
resultDF = questionsDF.join(broadcast(answers_month), 'question_id').select('question_id', 'creation_date', 'title', 'month', 'cnt')
```

- Performance comparison

When “spark.sql.shuffle.partitions”= 4: similar performance

Round	Original	Broadcasting
1	2.41	2.36
2	2.42	2.37

When “spark.sql.shuffle.partitions”= default (200): broadcasting is much faster performance.

Round	Original	Broadcasting
1	22	4.05
2	19.5	4.27

2. Caching

The caching solution and the original one have similar performance. I think the reason is in this case, dataframe is only used once .

- Code enhanced

```
answersDF.cache()

answers_month = answersDF.withColumn('month', month('creation_date')).groupBy('question_id', 'month').agg(count('*').alias('cnt'))
answers_month.cache()

resultDF = questionsDF.join(answers_month, 'question_id').select('question_id', 'creation_date', 'title', 'month',
…
```

```
answers_month.unpersist()
answersDF.unpersist()
```

- Performance comparison

  When “spark.sql.shuffle.partitions”= 4: similar performance

Round	Original	Caching
1	2.76	2.57
2	2.65	2.80

When “spark.sql.shuffle.partitions”= default(200): e similar performance

Round	Original	Caching
1	22.3	21.6
2	22	19.94
3	19.76	20

3. Repartition

- Code enhanced

```
answers_month = answersDF.withColumn('month',
month('creation_date')).groupBy('question_id', 'month').agg(count('*').alias('cnt'))

answers_month.orderBy('question_id').repartition(4, 'question_id')

resultDF = questionsDF.join(answers_month, 'question_id').select('question_id',
'creation_date', 'title', 'month',
```

- Performance comparison

When “spark.sql.shuffle.partitions”= 4: the repartition solution has gain stable and significant improvement

Round	Original	Repartition
1	3.15	2.24
2	2.9	2.0
3	2.65	1.98

When “spark.sql.shuffle.partitions”= default(200): similar performance.

Round	Original	Caching
1	21.38	20.11
2	20.00	20.33

4. Window function

- Code change

```
from pyspark.sql.window import Window

win = Window.partitionBy('question_id', 'month')

winAnswersDF = answersDF \
    .withColumn('month', month('creation_date')) \
    .withColumnRenamed('creation_date', 'answer_creation_date')


winResultDF = questionsDF \
    .join(winAnswersDF, 'question_id') \
    .withColumn('cnt', count('*').over(win)) \
    .select('question_id', 'creation_date', 'title', 'month', 'cnt') \
    .distinct() \
    .orderBy('question_id', 'month')
```

- Performance comparison

When “spark.sql.shuffle.partitions”= 4: similar performance and the original solution is slightly better

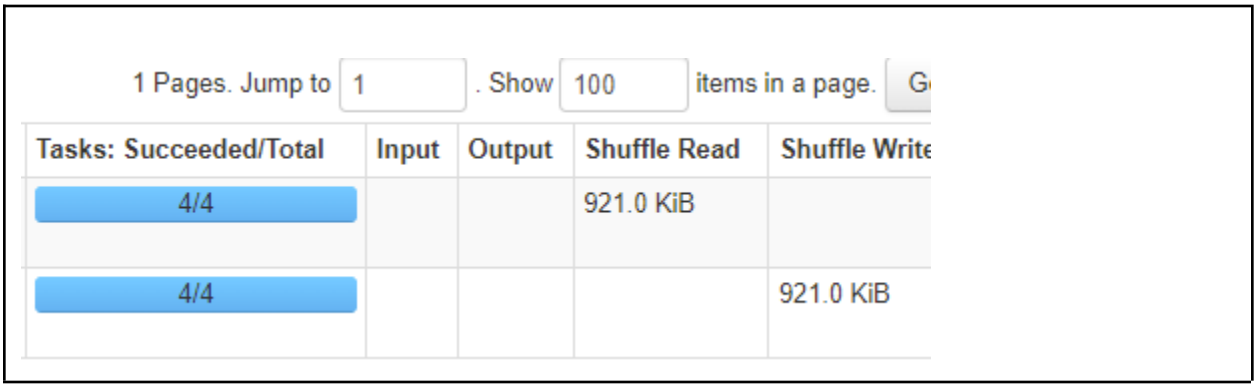
Round	Original	Window Function
1	2.61	2.76
2	2.57	2.74
3	2.61	2.80

When “spark.sql.shuffle.partitions”= default (200): the window function solution runs much faster

Round	Original	Window Function
1	19.11	4.45
2	19:3	4.29

II. Shuffle read/shuffle write

Among all solutions, ‘Broadcasting’ has least shuffle read and shuffle write, both are 921k; the other three approaches have similar volume of shuffle read/write – window function has slightly less.



Repartition

1 Pages. Jump to 1 . Show 100 items in a page. Go

	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
	85 ms	4/4			5.7 MiB	
	0.2 s	4/4				4.9 MiB
	54 ms	4/4			921.0 KiB	758.1 KiB
	0.2 s	4/4				921.0 KiB

Caching

1 Pages. Jump to 1 . Show 100 items in a page. Go

Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4/4			5.7 MiB	
4/4			921.0 KiB	758.1 KiB
4/4				921.0 KiB
4/4				4.9 MiB

Window function

1 Pages. Jump to 1 . Show 100 items in a page. G

	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
	4/4			5.9 MiB	
	4/4				4.9 MiB
	4/4				943.4 KiB