Optimizing Apache Spark

The Five Most Common Performance Problems

Serialization

The 5 Most Common Performance Problems (The 5 Ss)
Serialization

- Serialization is the last of our “most common problems”
- Spark 1.x provided significant performance gains over other solutions
- Spark 2.x, namely Spark SQL & the DataFrames API brought even more performance gains by abstracting out the execution plans
- We no longer write “map” operations with custom code but instead express our transformations with the SQL and DataFrames APIs
- That means that with Spark 2.x, when we regress back to code, we are going to see more performance hits

The 5 Most Common Performance Problems (The 5 Ss)

Serialization - Why it's bad

- Spark SQL and DataFrame instructions are compact, and optimized for distributing those instructions from the Driver to each Executor
- When we use code, that code has to be serialized, sent to the Executors and then deserialized before it can be expected
- Python takes an even hard hit because the Python code has to be pickled **AND** Spark must instantiate an instance of the Python interpreter in each and every Executor
- Compared that to the Python version of the DataFrames API which uses Python to express the operations executed in the JVM

The 5 Most Common Performance Problems (The 5 Ss)
Serialization - Catalyst Optimizer

- Besides the cost of serialization, there is another problem…
- These features create an analysis barrier for the Catalyst Optimizer
- The Catalyst Optimizer cannot connect code before and after UDF
- The UDF is a black box which means it limits optimizations to the code before and after, excluding the UDF and how all the code works together

The 5 Most Common Performance Problems (The 5 Ss)
Serialization - How Bad?

Remember our benchmarks?

- See Experiment #5980, **Step D-S** and **Step D-P**
- These use a do-nothing Lambda and a strait read
- The Scala version takes < 15 minutes
- The Python version takes ~2.5 hours!
- All because we executed this python code: `lambda x: None`

The 5 Most Common Performance Problems (The 5 Ss)

Serialization - Scala's Overhead

Let's see how serialization effects Scala in Experiment #4538 for Scala

- See **Step D** which uses higher-order functions
  - Uses functions from `org.apache.spark.sql.functions`
  - Note the execution time
- See **Step E** which uses two UDFs
  - See `parseId(..)` and `parseType(..)`
  - Note the execution time
- See **Step F** which uses Typed Transformations
  - See the `map(..)` operation
  - Note the execution time

The 5 Most Common Performance Problems (The 5 Ss)
Serialization - Scala's Overhead, Review

Step	Type	Duration
C	Baseline	~3 min
D	Higher-order Functions	~25 min
E	UDFs	~35 min
F	Typed Transformations	25+ min

The 5 Most Common Performance Problems (The 5 Ss)

Serialization - Python's Overhead

Let's see how serialization effects Python in Experiment #4538 for Python

- See **Step D** which uses higher-order functions
  - Uses functions from `pyspark.sql.functions`
  - Note the execution time
- See **Step E** which uses two UDFs
  - See `parseId(..)` and `parseType(..)`
  - Note the execution time
- See **Step F** which uses Pandas (or Vectorized) UDFs
  - See `@pandas_udf parseId(..)` and `@pandas_udf parseType(..)`
  - Note the execution time

The 5 Most Common Performance Problems (The 5 Ss)
Serialization - Python's Overhead, Review

Step	Type	Duration
C	Baseline	~3 min
D	Higher-order Functions	~25 min
E	UDFs	~105 min
F	Panda/Vectorized UDFs	~70 min

How do the two stack up against each other?

The 5 Most Common Performance Problems (The 5 Ss)

Serialization - Python vs Scala

Step	Type	Scala Duration	Python Duration
C	Baseline	~3 min	~3 min
D	Higher-order Functions	~25 min	~25 min
E	UDFs	~35 min	~105 min
F - Scala	Typed Transformations	~25 min	n/a
F - Python	Panda/Vectorized UDFs	n/a	> 70 min

Same

Really Bad

Bad

The 5 Most Common Performance Problems (The 5 Ss)

Serialization - Why?

Why do we still see such a proliferation of these [poorly performing] features?

- Integration with 3rd-party libraries
  - Common in the data sciences
  - In some cases there is no other choice
- Attempting to integrate with the company's existing frameworks
  - e.g. custom business objects
  - or proprietary libraries
- Migrations from legacy systems like Hadoop
  - Copy and pasting code instead of rewriting them as higher-order functions

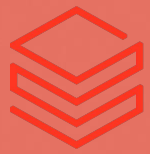
What can we do to mitigate the serialization issues?

The 5 Most Common Performance Problems (The 5 Ss)
Serialization - Mitigation

- Short answer, don't use UDFs, Vectorized UDFs or Typed Transformations
- The need for these features is REALLY rare
- The SQL higher-order functions are very robust
- But if you have to...
  - Use Vectorized UDFs over "standard" Python UDFs
  - Use Typed Transformations over "standard" Scala UDFs

Should we rewrite our Spark code to use Scala instead of Python?

No!



databricks