7_shared_variables

February 21, 2022

[1]:
```
# Import SparkContext and SparkConf
from pyspark import SparkContext, SparkConf

# Initialize spark
conf = SparkConf().setAppName("sharedVars")
sc = SparkContext(conf=conf)
```

```
22/02/21 14:32:34 WARN Utils: Your hostname, ThinkCentre resolves to a loopback
address: 127.0.1.1; using 10.180.5.223 instead (on interface eno1)
22/02/21 14:32:34 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
22/02/21 14:32:34 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4041.
Attempting port 4042.
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4042.
Attempting port 4043.
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4043.
Attempting port 4044.
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4044.
Attempting port 4045.
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4045.
Attempting port 4046.
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4046.
Attempting port 4047.
22/02/21 14:32:36 WARN Utils: Service 'SparkUI' could not bind on port 4047.
Attempting port 4048.
```

1 Broadcast

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

cached means:

- they definitely should NOT be anything super large, like a large table or massive vector.
- Secondly, broadcast variables are immutable, meaning that they cannot be changed later on.

This may seem inconvenient but it truly suits their use case. If you need something that can change, I'd certainly point you to accumulators.

So now we know that broadcast variables are: - Immutable - Distributed to the cluster - Fit in memory

Note:

Spark actions are executed through a set of stages, separated by distributed "shuffle" operations. Spark automatically broadcasts the common data needed by tasks within each stage. The data broadcasted this way is cached in serialized form and deserialized before running each task. This means that explicitly creating broadcast variables is only useful when tasks across multiple stages need the same data or when caching the data in deserialized form is important.

[2]:
```
# Broadcast a list
b = sc.broadcast([1, 2, 3])

# Now [1,2,3] is present with all the executors

b.value
```

[2]:
```
[1, 2, 3]
```

[3]:
```
# How to remove a boradcast variable from RAM?
#b.unpersist()
#b.value
```

2 Accumulator

Accumulators are variables that are only "added" to through an associative and commutative operation and can therefore be efficiently supported in parallel. ** They can be used to implement counters (as in MapReduce) or sums.** Spark natively supports accumulators of numeric types.

- An accumulator is created from an initial value v by calling SparkContext.accumulator(v).
- Tasks running on a cluster can then add to it using the add method or the += operator.
- However, they cannot read its value. Only the driver program can read the accumulator's value, using its value method.

[4]:
```
# 2. Accumulator
accum = sc.accumulator(0) # initilaize accum with zero
accum
```

[4]:
```
Accumulator<id=0, value=0>
```

[5]:
```
sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
accum.value
```

[5]: 10

For accumulator updates performed inside actions only,

Spark guarantees that each task's update to the accumulator will only be applied once, i.e. restarted tasks will not update the value.

accumulator updates are not guaranteed to be executed when made within a lazy transformation like map().