TOP 3 TROUBLESHOOTING TIPS TO KEEP YOU SPARKING

RUSSELL CARDULLO
ENGINEERING TECHNICAL LEAD

September 13, 2013

Data Science, Scala, Spark

We've been experimenting with Spark over the past few weeks. Our two main motivations for doing so were Spark's excellent support for iterative algorithms and the new Spark streaming features for real-time data processing.

While we found the Spark APIs easy to work with, we did run into a few beginner mistakes that we wanted to share.

THE SPARK EXECUTION MODEL

Most of the issues we encountered had to deal with understanding the execution environment of our code. Spark jobs consist of a driver program that executes parallel operations on a distributed dataset. In order to troubleshoot issues it helps to have an understanding of where different parts of your code run.

The cannonical Spark example from their homepage is the venerable word count:

```
1  file = spark.textFile("hdfs://...")
2
3  file.flatMap(line => line.split(" "))
```

```
    .map(word => (word, 1))
    .reduceByKey(_ + _)
```

While this code is launched from a Spark driver program, Spark will execute portions of it on remote workers. Basically anything within a closure (i.e. the functions you pass to `map`, `reduce`, etc.) will run in the scope of a worker while everything outside that runs in the scope of the driver. With that in mind the one rule to help troubleshoot issues: **objects created outside the scope of the closure are not necessarily in the same state within the closure**.

To better illustrate that here are some specific issues we ran into:

1 - WHY DID MY SPARK JOB FAIL WITH NOTSERIALIZABLEEXCEPTION?

This was a common issue we ran into in our first few jobs. Any objects created outside of the scope of the closure will be serialized to the workers. For instance given the following code:

```
object MyFirstSparkJob {
  def main(args: Array[String]) {
    val ssc = new StreamingContext(args(0), "BeaconCount", Seconds(1))
    val parser = new JSONParser // <-- INSTANTIATED HERE

    val lines = ssc.textFileStream("beacons.txt")
    lines.map(line => parser.parse(line)) // <-- IN THE CLOSURE
    lines.foreach(line => println(line))

    ssc.start()
  }
}
```

The object `parser` is referenced within the closure of a parallel operation and will be serialized to the workers. However the `JSONParser` class is not serializable which causes the error.

To fix this error we would either need to change `JSONParser` to extend `Serializable`, or move the creation of the object inside of the map closure.

2 - WHY IS MY SPARK JOB SO SLOW AND ONLY USING A SINGLE THREAD?

References to singleton objects with the closure of a parallel operation will bottleneck the process as these references will happen within the driver program. Consider the following code:

```
object JSONParser {
  def parse(raw: String): String = ...
}

object MyFirstSparkJob {
  def main(args: Array[String]) {
    val ssc = new StreamingContext(args(0), "BeaconCount", Seconds(1))

    val lines = ssc.textFileStream("beacons.txt")
    lines.map(line => JSONParser.parse(line))
    lines.foreach(line => println(line))

    ssc.start()
  }
}
```

This is similar to the code from our first example, but here the `parser` instance is now a singleton created in the scope of our driver program. This object is not passed in the context of the workers so Spark will execute code referencing that object only in the driver program, effectively creating a bottleneck for your job.

To fix this you could turn this object into a serializable class that can be passed to the worker processes. Or you could use broadcast variables to make that object available in the context of each worker.

3 - WHY DID MY SPARK JOB FAIL WITH JAVA.LANG.ILLEGALARGUMENTEXCEPTION:

SHUFFLE ID NNNN REGISTERED TWICE?

This issue was particularly nasty. In order to trace this we checked the Spark logs for all instances of 'job NNNN'. What we found was that earlier that job had thrown an exception that was not caught within the Spark driver program.

While we aren't sure whether this is an issue in the Spark framework, we've been able to eliminiate these issues by making sure that we catch exceptions we can recover from within the parallel operations. Luckily in our case we were able to handle the exception within the closure.

ADDITIONAL READING

As of now we're on our third week of using Spark and we're continuing to add more learnings under our belt. The following resources were extremely valuable in helping us ramp up:

- AMP Camp Big Data Mini Course - great place to get started learning about Spark with an awesome step by step tutorial. Seriously if you haven't been

with lots of examples to learn from.
- Spark Users Mailing List - the mailing list just moved here a few months ago but there's a lot of recent activity.
- Spark Users Google Groups List - the old mailing list for Spark, still has lots of good info.

« The Right Time for Structure: Retrofitting Backbone.js to a jQuery Application

Notice to startups: You are doing data science wrong »

COMMENTS

John Meehan • 2 years ago

Can you give an example of how to modify the parser object in section 2 (single threaded) so that it's a serializable class, e.g. with jackson ObjectMapper or similar?

• Reply • Share ›

Russell Cardullo ➚ John Meehan • 2 years ago

One way you could do it is to make that a class that extends Serializable, something like:

class JSONParser extends Serializable

Then in your job you'll need to create a new instance of JSONParser.

However I'm not sure any of this is necessary in Spark 1.0 and newer (this post was written when Spark 0.7.3 was the latest). I've noticed that similar patterns don't suffer the same performance penalty.

王猛 • 2 years ago

good!

• Reply • Share ›

LATEST POST ON NATIVE

From LinkedIn to Sharethrough: The Perks And Adjustments Settling Into Startup Life

PRODUCTS +

SFP

SAM

CUSTOMERS +

Publishers

Advertisers

RESOURCES +

Insights

Publication

Conference

Support

ABOUT US +

Careers

Company

Press

Engineering

CONTACT +

Contact Us

LinkedIn

Facebook