Walter Blair
603 Assignment 2

Singleton

In Paul Anderson's lab, we're trying to make progress in the application of Apache Spark for computational genomics. I started wading into my research topic this past spring and have had a huge learning curve trying to get my head around distributed in-memory computing and the oddities of Scala. I'm happy to report that this class has already helped me better understand the design-aspects of and pragmatic decisions behind Spark.

When we started talking about singletons, my first thought was the SparkContext (later wrapped by SparkSession) of Apache Spark. To back up and provide a bit of context of my own, my adviser and I actually just submitted this past Sunday a conference paper on work stemming from the singleton-nature of a SparkContext. My thesis research is centered around the improvement of genomics processing pipelines, where a bunch of individual applications are linked together to take some gene sequence input and spit out a meaningful processed/analyzed output. Apache Spark is great for doing this quickly, because it provides an in-memory distributed framework for each of these applications to do very fast computations. The trouble with linking applications together, though, is that each stand-alone application runs in a stand-alone SparkContext that does all the behind-the-scenes work of coordinating memory, cluster nodes, etc. When you start an application you start a SparkContext in which it runs, and when the application is over you close the SparkContext. To join them together in order to preserve/share data structures is a problem that we were trying to help solve, but the singleton design pattern helps me understand why the problem exists in the first place.

The SparkContext is usually treated like a singleton, but I find it really interesting that it's not actually a singleton. Fig. 1 shows the start of the source code for SparkContext. SparkContext extends Spark Logging which ties into the logging use-case for the singleton design pattern discussed in class. The comments state that by default only one SparkContext can exist per JVM, but that this limitation can be lifted. Why would you want to lift this limitation? I don't really know! In my mind, I'd want a single machine running a single Spark job rather than multiple independent Spark jobs. As mentioned in the documentation, there was a request opened for Spark to support multiple contexts per JVM and a discussion of several use cases that would thereby be [only slightly] improved. One exchange in this thread clenched my decision to use this example for this assignment:

> *Jason: "I guess I get a bit confused on the use of SparkContext anyway, if the intent isn't to have multiple, then why not force it to be a singleton and possibly a different name that would imply only a single instance should exist." (https://goo.gl/J7qGx2)*
>
> *Reply: 'Jason Hubbard I bet that in retrospect it would have been better to make the `SparkContext` uninstantiable and access it only via a factory method. Too late for that now. '*

This is a fascinating intersection of design choices and pragmatic realities in the software development process. I can't quite tell whether the design choice was originally to allow the instantiation of multiple SparkContexts or the developers just had to cut some corners, throw a few cares to the wind, and meet whatever deadline was looming over them. Elsewhere in the thread someone mentions that there are scattered throughout the source several assumptions that multiple contexts can be created, which is

probably the reason why developers have not bothered to refactor to enforce the singleton design that seems now to be preferable.

```
63  /**
64   * Main entry point for Spark functionality. A SparkContext represents the connection to a Spark
65   * cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.
66   *
67   * Only one SparkContext may be active per JVM.  You must `stop()` the active SparkContext before
68   * creating a new one.  This limitation may eventually be removed; see SPARK-2243 for more details.
69   *
70   * @param config a Spark Config object describing the application configuration. Any settings in
71   *   this config overrides the default configs as well as system properties.
72   */
73  class SparkContext(config: SparkConf) extends Logging {
74
75    // The call site where this SparkContext was constructed.
76    private val creationSite: CallSite = Utils.getCallSite()
77
78    // If true, log warnings instead of throwing exceptions when multiple SparkContexts are active
79    private val allowMultipleContexts: Boolean =
80      config.getBoolean("spark.driver.allowMultipleContexts", false)
81
82    // In order to prevent multiple SparkContexts from being active at the same time, mark this
83    // context as having started construction.
84    // NOTE: this must be placed at the beginning of the SparkContext constructor.
85    SparkContext.markPartiallyConstructed(this, allowMultipleContexts)
86
```

Figure 1. Source code for SparkContext, including code that sets default to only one per JVM.

Every bit of Spark code I've seen acts like SparkContext is a singleton. The code below is a snippet from a stand-alone application I wrote that creates a SparkSession which wraps SparkContext in the latest Spark versions, and Fig. 2 shows the SparkContext in an interactive Scala/Spark-shell session.

```
class TrimS {
  // case class must be defined outside of scope
  case class Read(name1: String, seq: String, name2: String, phred: String)
  def main(args: Array[String]) {

    val spark = SparkSession
      .builder()
      .appName("Scala Trimmer")
      .master("local")    // for testing in IntelliJ
      .getOrCreate()
```

In summary, Apache Spark's SparkContext looks like a singleton and smells like a singleton but isn't enforced as such. While one user expressed interest in using multiple contexts per JVM, every piece of Spark software I've seen instantiates SparkContext as a singleton object. It seems clear to me that the designers hadn't brushed up on their object-oriented design patterns before cranking out Spark. I think this may partly be due to the funkiness of Scala which among other oddities doesn't name static objects or methods. The simplest way to enforce a singleton is simply to define an Object rather than a Class, and the Spark developer in the thread mentioned that a well-structured factory could enforce a singleton design pattern.

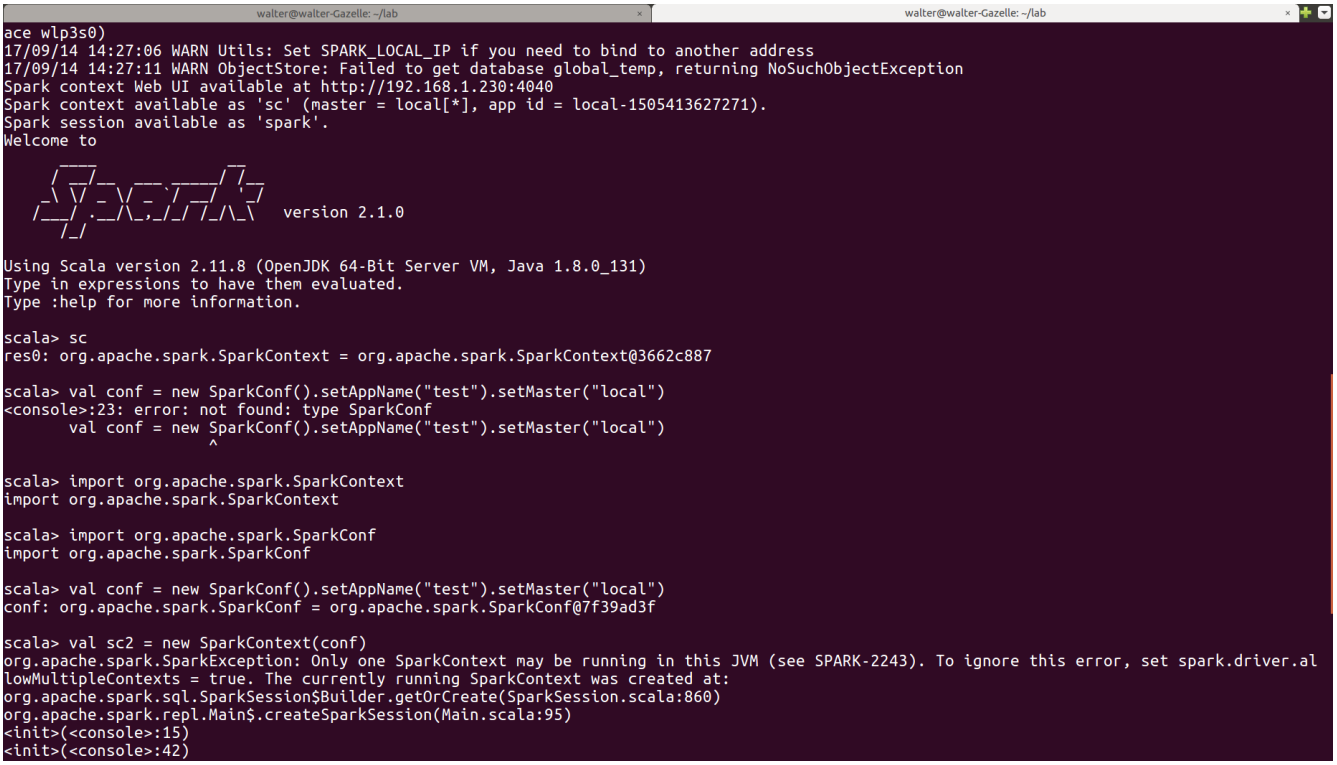

Figure 2. An interactive Spark session where I try to create another SparkContext and receive an error that by default only one SparkContext can exist per JVM. Console in this screenshot is pasted below for larger font.

Spark context available as 'sc' (master = local[*], app id = local-1505413627271).
Spark session available as 'spark'.
Welcome to

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/
```

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_131)
Type in expressions to have them evaluated.
Type :help for more information.

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@3662c887

scala> val conf = new SparkConf().setAppName("test").setMaster("local")
<console>:23: error: not found: type SparkConf
       val conf = new SparkConf().setAppName("test").setMaster("local")
                      ^

scala> import org.apache.spark.SparkContext
```

```
import org.apache.spark.SparkContext

scala> import org.apache.spark.SparkConf
import org.apache.spark.SparkConf

scala> val conf = new SparkConf().setAppName("test").setMaster("local")
conf: org.apache.spark.SparkConf = org.apache.spark.SparkConf@7f39ad3f

scala> val sc2 = new SparkContext(conf)
org.apache.spark.SparkException: Only one SparkContext may be running in this JVM (see SPARK-2243). To ignore this error, set spark.driver.allowMultipleContexts = true. The currently running SparkContext was created at:
org.apache.spark.sql.SparkSession$Builder.getOrCreate(SparkSession.scala:860)
...
...
...
```