Accessing ROOT from the JVM (Java/Scala)

Jim Pivarski

2016-02-08

Motivation

Most of the big data-pipeline frameworks used in industry run on the Java Virtual Machine (JVM); most physics data is in ROOT.

In particular, Apache Spark is written in Scala.

- Scala is a JVM language (essentially interchangeable with Java, but more friendly for data analysis; has a REPL).
- Spark supports analyses in Scala, Java, Python through sockets (Py4J), and R through pipes (stdin/stdout).
- No support for C/C++ or other native code.
- Sockets and pipes both introduce serialization and transmission overhead.

Similar motivation as for PyROOT: like Python, the JVM is a platform that is increasingly being used for data analysis.

We need an efficient and robust bridge.

Technologies

FreeHEP-ROOTIO

Pure-Java reimplementation of ROOT I/O on `java.freehep.org`.

- Hard to find (docs point to a JAR compiled in 2001).
- But it lives! `svn://svn.freehep.org/svn/freehep/trunk` has recent commits: 2014 (`src/main`) and 2015 (`pom.xml`).
- Reads and writes ROOT files with Java reflection to dynamically create runtime objects.
- FreeHEP-ROOTIO compiles with unit tests removed (they require access to an internal GLAST server).
- It's not on Maven Central ("`freehep-io`" is unrelated).
- Starting to test it in the Scala REPL: this is promising.

Technologies

Java Native Interface (JNI)

For compiling C/C++ code that can be used in Java programs.

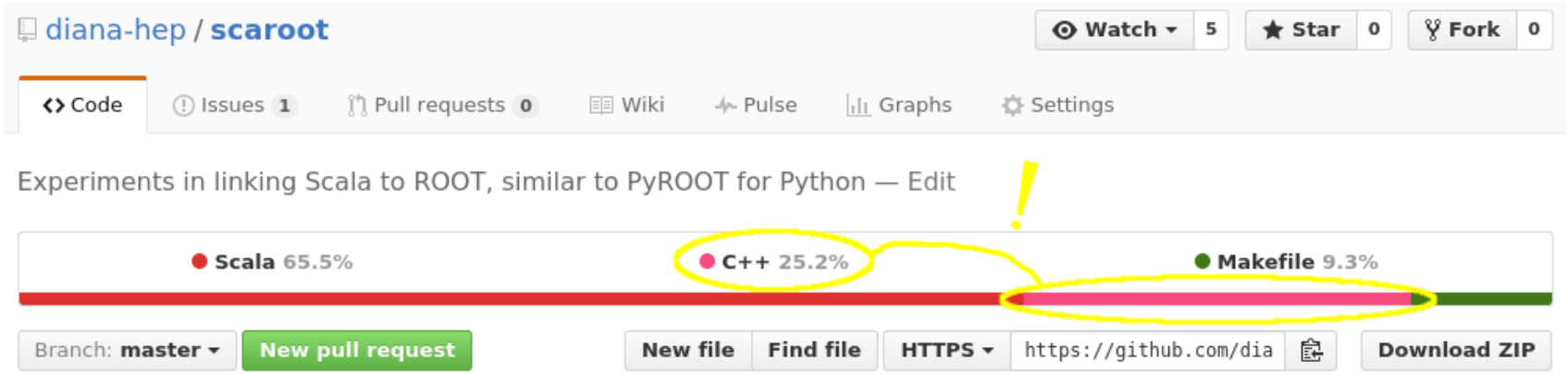
- Java community is strongly biased against it.
  (Unlike the equivalent in Python, which is frequently used.)
  - C/C++ memory has fixed locations; Java has a generational garbage collector. (Python has fixed memory, like C/C++.)
  - Java classes have no destructors other than `finalize()`, which is not guaranteed to be called (like Python `__del__`).
  - `try-finally` is recommended to avoid memory leaks.
- Attempted, not promising: mysterious segmentation faults.

Java Native Access (JNA)

Links Java code to pre-built shared libraries (`.so` files).

- Same issues as above except the interface is cleaner.
- Implicit data transformation overhead: "100 μs per call?"
- Promising: no myserious segmentation faults.

ScaROOT



- Tested JNI (unsuccessfully) and JNA (successfully).
  - Can open ROOT file and print `->ls()` from Scala.
- Set up a clean build environment with Maven and Make:
  - `mvn install` command runs `make` to build C++ first, then Scala (mixed with any Java, if needed).
  - C-style symbol names (`extern "C"`) in `scaroot.so`.
  - `scaroot.so` enclosed within `scaroot.jar`.
  - User submits only `scaroot.jar` to the Spark cluster, but `LD_LIBRARY_PATH` must be pointing to ROOT on the cluster.
  - Perhaps I can encapsulate a whole version of ROOT in the `scaroot.jar`, so the whole thing gets sent with the workflow. Needs testing.
- Namespace: `org.dianahep`, GroupID: `org.diana-hep`.

src/main/cpp/scaroot.cpp

```
#include <stdint.h>
#include "TFile.h"

extern "C" {
  int64_t new_TFile(char *fileName);
  void delete_TFile(int64_t pointer);
  void TFile_ls(int64_t pointer);
}

int64_t new_TFile(char *fileName) {
  TFile *tfile = new TFile(fileName);
  return (int64_t)tfile;
}

void delete_TFile(int64_t pointer) {
  TFile *tfile = (TFile*)pointer;
  delete tfile;
}

void TFile_ls(int64_t pointer) {
  TFile *tfile = (TFile*)pointer;
  tfile->ls();
}
```

src/main/scala/org/dianahep/scaroot/Main.scala

```
package org.dianahep

import com.sun.jna._

package scaroot {
  object ROOTLibrary extends Library {
    Native.register("/resources/native/scaroot.so")  // in JAR
    @native def new_TFile(fileName: String): Long
    @native def delete_TFile(pointer: Long): Unit
    @native def TFile_ls(pointer: Long): Unit
  }

  object Main {
    def main(args: Array[String]) {
      val pointer = ROOTLibrary.new_TFile("Event.root")
      println(s"pointer value $pointer")
      ROOTLibrary.TFile_ls(pointer)
      println(s"see a listing?")
      ROOTLibrary.delete_TFile(pointer)
      println(s"still here?")
    }
  }
}
```

src/main/cpp/Makefile

```
all: scaroot.cpp
    g++ -fPIC -shared -Wl,--no-as-needed \
        $(shell root-config --cflags --ldflags --libs) \
        -o ../../../src/main/resources/native/scaroot.so \
        scaroot.cpp
```

pom.xml fragment

```
...
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals><goal>exec</goal></goals>
      <configuration>
        <workingDirectory>src/main/cpp</workingDirectory>
        <executable>make</executable>
      </configuration>
...
```

Directory structure