ROOT4J / SPARK-ROOT: ROOT I/O for JVM and Applications for Apache Spark

V. Khristenko[1] J. Pivarski [2]

[1] Department of Physics
The University of Iowa

[2] Princeton University - DIANA

ROOT I/O Workshop, 2017

Outline

Motivation

- Enable access to Physics Data from SPARK.
- ROOT Data Format is, almost, self-descriptive -> JVM-based I/O is therefore a realistic goal!
- Open up ROOT for the use with Big Data Platforms (Spark is just a single example)

What SPARK-ROOT is

Only I/O

The primary objective of this work is to provide a JVM-based access to ROOT's binary format

- SPARK-ROOT is a ROOT's I/O Library for JVM.
- SPARK-ROOT is purely Java/Scala based.
- SPARK-ROOT implements a new Spark Data Source, similar to Parquet, Avro.

TTree as Spark Dataframe

SPARK-ROOT allows to access binary ROOT format within JVM directly and represent ROOT TTree as Spark's Dataset/Dataframe/RDD.

Supported Datatypes

- Basic Types: Integer, Boolean, Float, Double, Long, Char, Char*
- Fixed-size Arrays and variable sized arrays
- Multidimensional Arrays
- Pointers to basic Types - a la dynamic arrays
- Structs (in multi-leaf style)
- STL Collections (for now, map/vector) of basic types
- Nested STL Collections of basic types
- STL String
- Composite Classes of Basic Types and of Composite Classes
- STL Collections of Composite Classes
- STL Collections of Composite with STL Collections of Composite as class members - multi-level hierarchy
- TClonesArray, when member class is available before Read-Time!

Supported Functionality

- JIT compilation using TStreamerInfo to get to TTree
- Automatic Spark Schema Inferral for supported types in the TTree.
- Proper Branch Flattening
- Hadoop DFS Support
- Early Stage Filtering

Limitations

Run/Read-Time Limitations of Spark

Spark builds a schema before the actual reading is done. It imposes constraints that all the data types must be known a priori to reading! Not the case for ROOT!

```
class Base {...};
class Derived1 : public Base {...};
class Derived2 : public Base {...};

std::vector<Base*> - at read/run-time can be ...
1) std::vector<Derived1>
2) std::vector<Derived2>
3) std::vector<Base>

Same idea applies to TClonesArray.
```

Maven Central

Maven Central!

We are on Maven Central - include spark-root as package dependency -> no need to compile yourself!

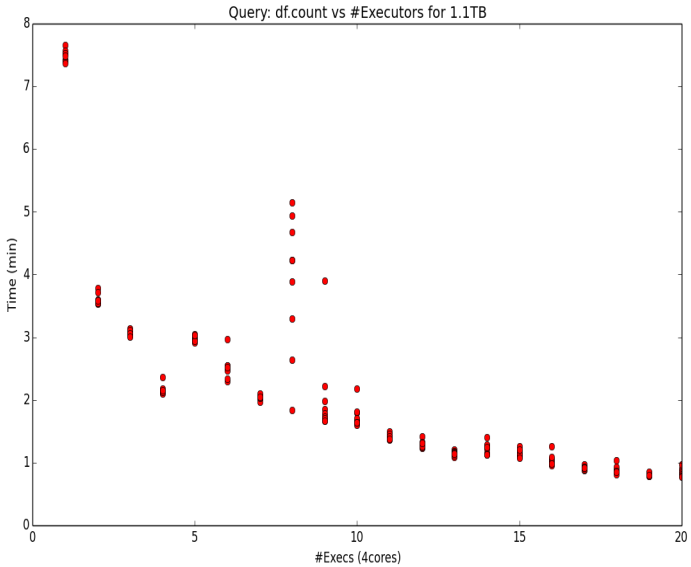
```
./spark-shell --packages
    org.diana-hep:spark-root_2.11:0.1.7

import org.dianahep.sparkroot._

scala> val df = spark.sqlContext.read.root(
    "file:/Users/vk/software/Analysis/files/test/
    ntuple_drellyan_test.root")
```

Intel Lab Cluster

- approx. 20 machines
- approx. 16 physical cores each
- Public Dataset is being copied - 1.2TB soon available
- Suggestions for Benchmark Tests
- My current plan is: a set of basic queries on RDD/Dataframe(count, map, flatMap, groupBy, etc...., aggregate with histogrammar) over Muon objects...
- Mycurrent plan is: execution of these queries must be measured by varying the level of parallelism.
- If there are suggestions - please let me know!!!

Basic Performance

- CMS Public Dataset for benchmarks
- Spark's Listeners to collect performance information.
- **Preliminary Results for 1.2TB (>1K files) for df.count**



Github and Useful Links

- spark-root
- spark-root Scala User Guide
- HowTo for analytix
- Jupyter python-based notebook for processing 1.2TB of public dataset
- root4j

User Guides

More User Guides are to come soon!

Summary

- Huge Huge Thanks to Philippe, Danilo, Axel, Sergey Linev for replying to my questions!
- root4j/spark-root - JVM-based ROOT I/O library. It Works!
- spark-root allows one to view TTree as Spark Dataframe
- spark-root 0.1.7 is available on Maven Central for use
- Limitations do exist, but resolveable!

What's next?!

- There is no I/O Optimization implemented yet
- HDFS Locality - right now only HDFS access is done.
- Tuning Partitioning/Splitting - currently it's file-based
- Name Aliasing - useful for physicists
- Cross-references, a la TRef???
- Overcome the limitations
- In principle, root4j should be rewritten from scratch
- Prepare a decent TestBed - given Scala has a superb support for that!