Laurelin: Java-native ROOT I/O for Apache Spark

vCHEP
19th May, 2021

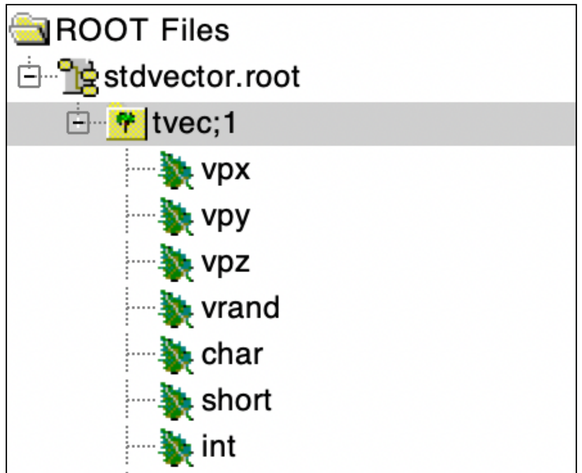
Andrew Melo (Vanderbilt University) and many others

Big Data and HEP

- Foreseen data volumes in HEP are an opportunity for Physics but a challenge to perform analysis
- *Big Data* tools show promise as a key enabler of future analyses
- Historically, no cross-pollination within and without HEP -- new tools can't interact with our data
- "Apache Spark™ is a unified analytics engine for large-scale data processing."

Spark + ROOT = Laurelin

- Widely-used, open-source project
- Processes *DataFrames* (roughly equivalent to a TTree) in parallel
- Native support for many file formats, but not ROOT
- Laurelin is a Spark plugin for the ROOT file format

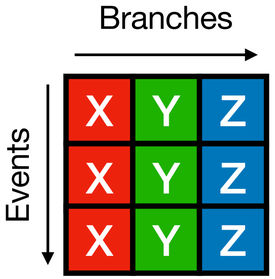


Laurelin

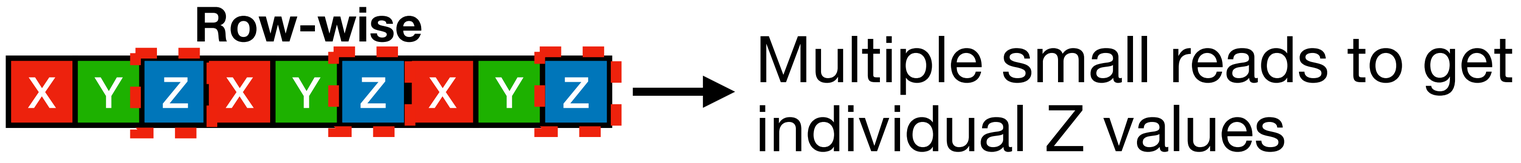
```
spark.read.format('root').load('stdvector.root').printSchema()

root
 |-- vpx: array (nullable = false)
 |     |-- element: float (containsNull = false)
 |-- vpy: array (nullable = false)
 |     |-- element: float (containsNull = false)
 |-- vpz: array (nullable = false)
 |     |-- element: float (containsNull = false)
 |-- vrand: array (nullable = false)
 |     |-- element: float (containsNull = false)
 |-- char: array (nullable = false)
 |     |-- element: byte (containsNull = false)
```

ROOT TTrees

- Stores data by-branch (column-wise) and not by-event (row-wise)
  - Perfect for HEP since we typically don't need to read every branch

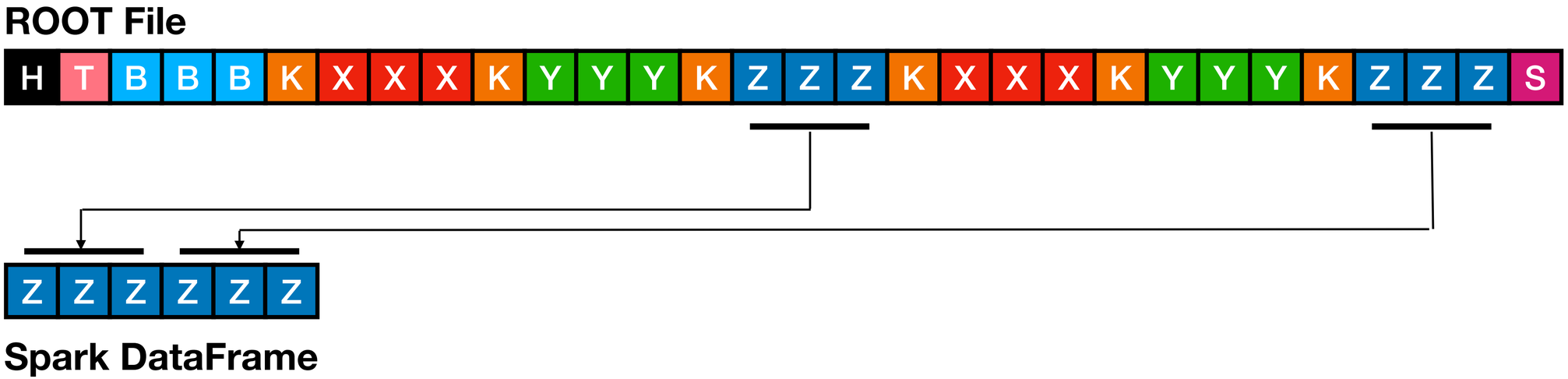


Much more efficient



Spark DataFrames

- Can be stored/processed either row- or column-wise
- The columnar version is almost exactly how ROOT stores data in files
- "Just" need to find the blobs of interesting data and move them to memory

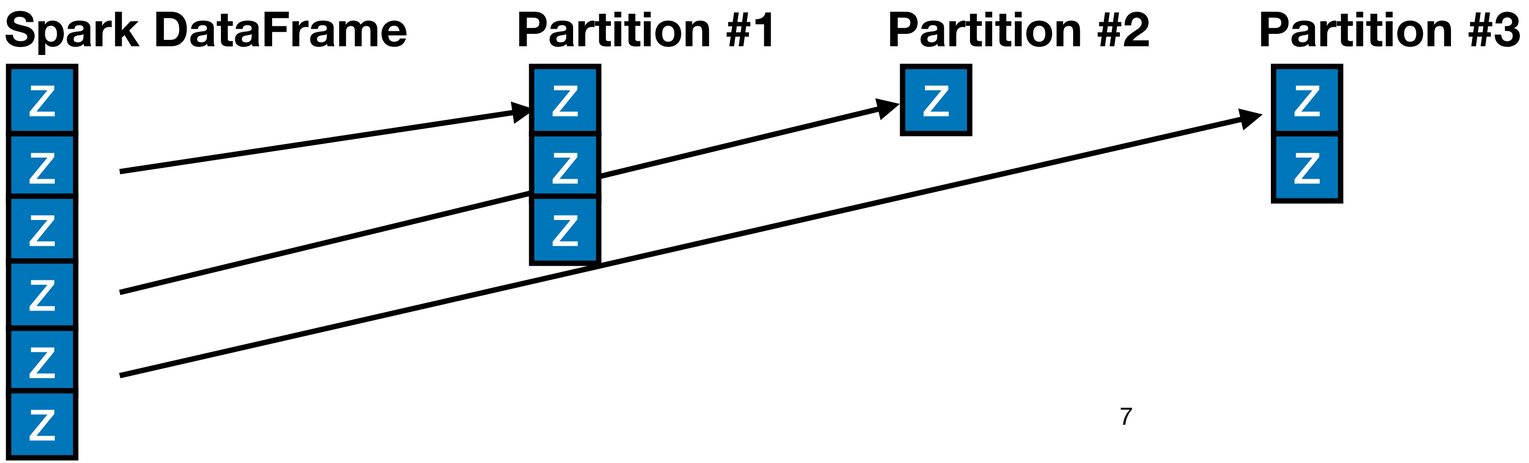


Reading ROOT Files

- Not much documentation, what exists is often outdated
- Different ROOT versions write slightly different files
- ROOT files describe the objects inside using "Streamers"
- Streamers are a step-by-step guide of how to read an object
  - "First read a integer, this is called 'version', now read a float, this is...."
  - All we have to do is load the streamers, then use that to load the TTree!
  - ... can't use the streamer to decode the streamers, so there's still some hardcoded decoders
- With the streamer, we can load the TTree, TBranch, and TLeaf, which lets is find where the TBaskets are located for a particular branch

Reading ROOT Files into Spark

- When a user runs a Spark query, Spark automatically tracks which columns are used and informs the file reader to only read those files
- Laurelin is responsible for partitioning files into smaller chunks (usually 200k events), then loading each partition from disk into memory
- Spark then handles coordinating the cluster, running the code over each partition, handling retries and collecting the results back to the user



Results

- With Laurelin installed, the ROOT format becomes a first class citizen in Spark
- ROOT TTrees store their data in a format that's near identical to Spark's internal representation
- By exposing the baskets directly to Spark in columnar form, this allows Spark to perform vectorized operations w/o an intermediate conversion

Example Spark Query

Configure

Open TTree

Query

```
1  spark = pyspark.sql.SparkSession.builder \
2      .master("local[1]") \
3      .config('spark.jars.packages', 'edu.vanderbilt.accre:laurelin:1.1.1') \
4      .getOrCreate()
5
6  df = spark.read.format('root') \
7                 .option("tree", "Events") \
8                 .load('A3AAE4A7-E384-8449-8C4E-E3473A20D211.root')
9
10 df.select("nMuon", "nElectron",
11            (df.nMuon + df.nElectron).alias("lepSum")).show()
```

Result

```
+-----+---------+------+
|nMuon|nElectron|lepSum|
+-----+---------+------+
|    2|        1|     3|
|    0|        0|     0|
|    2|        0|     2|
|    0|        1|     1|
|    0|        1|     1|
|    0|        2|     2|
|    1|        0|     1|
```

Future Plans / Integrations

Laurelin Plans:

- Write support
- Modify the in-memory format to be Arrow compatible
  - Existing format is very close
- Optimized join support, enabling a usable "friend tree" functionality

Integrations:

- Adding a Spark-based func_adl[1] backend
- Enhancing support in Coffea[2] for Spark
- Explore GPU-based queries with nVidia Rapids[3]

[1] https://github.com/iris-hep/func_adl
[2] https://coffeateam.github.io/coffea/
[3] https://rapids.ai/

Thanks!

https://github.com/spark-root/laurelin