DESIGNING NEW INTERFACES FOR ROOT DATA PROCESSING

EP - SFT
KALLE VUORINEN

AUGUST 11TH 2016

SUPERVISORS: G. GANIS
P. MATO
Tree

- Tree is an (ADT) abstracted data type
- Trees have branches and branches have leaves
- Widely used
  - Also at LHC: ROOT TTrees
Designing new interfaces for ROOT data processing  Kalle Vuorinen

BACKGROUND

Usage of TTree

```python
In [ ]: TFile f('data.root')
   TTree t = f.Get('T')
   t->Draw(">>myEntryList",func1 & func2 & func3,"LEGO");
   TEntryList* myEntryList;
   gDirectory->GetObject("myEntryList",myEntryList);
   t->SetEntryList(myEntryList);
   t->Draw("Var1:Var2", func4, "LEGO");
```
Designing new interfaces for ROOT data processing

Kalle Vuorinen

3

CHAINS OF FUNCTIONAL PRIMITIVES

In [ ]:
(roottree.tree('data.root', 'T')
.filter(func1)
.map(func2)
.filter(func3)
cache()
.filter(func4)
histo('Var1:Var2')
.Draw('LEGO'))

func can be any callable object, a usual function, a callable of a class or a lambda.
**TDATAFRAME**

- New class to ROOT
- Works as a dataset class
  - Describes a TTree
- Possibility to use functional chains
- Possibility to cache

```plaintext
t.filter(func1).filter(func2).cache()
Identifies if the functions stay the same or not
```

```plaintext
t.filter(func2).filter(func1).cache()
Caches the values again, because the order has changed
```

func can be any callable object,
a usual function, a callable of a class or a lambda.
FUNCTION TYPES

Transformations

Lazy functions, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset

- Filter(): Filter out all the elements of the dataset
- Map(): Returns a new dataset with the elements changed by function
- FlatMap(): Map a function over a dataset and flatten the result with one level
FUNCTION TYPES

Actions

Functions, that return a value to the driver program after running a computation on the dataset

- Draw(): Draws the histogram according its options
- Histo(): Creates histogram and fills it with the results
- Cache(): Caches the result for later use.
CACHING

DataFrame.filter(function).map(function).cache()

Caches the filtered and mapped result
- Allows the usage of the cached result to be a lot faster the next time you call for it

DataFrame = dataset, for example a TTree
Using cache

```python
DataFrame.filter(function).map(function).cache().filter(function)
  .Draw('LEGO')
```

Uses the cached values and filters that once and then draws it.
- We save time by having a cache of the first results

DataFrame = dataset, for example a TTree
TEST RESULTS
Designing new interfaces for ROOT data processing

Kalle Vuorinen

Comparison

- data.root, 4 293 120 entries
STATUS OF THE PROJECT

Python implementation

- Python implementation is almost ready
- Needs more transformations and actions

C++ implementation

- C++ implementation in progress
- Waiting the identifying of all the needed transformations and actions
THANK YOU!

Oulu
65°01’N 025°28’E

Temperature: +30 celsius to -30 celsius
Population: ~200'000
BACKUP SLIDES
<table>
<thead>
<tr>
<th>Transformation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>map(func)</td>
<td>Return a new distributed dataset formed by passing each element of the source through a function func.</td>
</tr>
<tr>
<td>filter(func)</td>
<td>Return a new dataset formed by selecting those elements of the source on which func returns true.</td>
</tr>
<tr>
<td>flatMap(func)</td>
<td>Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).</td>
</tr>
<tr>
<td>mapPartitions(func)</td>
<td>Similar to map, but runs separately on each partition (block) of the RDD, so func must be of type Iterator&lt;T&gt; =&gt; Iterator&lt;U&gt; when running on an RDD of type T.</td>
</tr>
<tr>
<td>mapPartitionsWithIndex(func)</td>
<td>Similar to mapPartitions, but also provides func with an integer value representing the index of the partition, so func must be of type (int, Iterator&lt;T&gt;) =&gt; Iterator&lt;U&gt; when running on an RDD of type T.</td>
</tr>
<tr>
<td>sample(withReplacement, fraction, seed)</td>
<td>Sample a fraction fraction of the data, with or without replacement, using a given random number generator seed.</td>
</tr>
<tr>
<td>union(otherDataset)</td>
<td>Return a new dataset that contains the union of the elements in the source dataset and the argument.</td>
</tr>
<tr>
<td>Action</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>reduce((func))</td>
<td>Aggregate the elements of the dataset using a function (func) (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.</td>
</tr>
<tr>
<td>collect()</td>
<td>Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.</td>
</tr>
<tr>
<td>count()</td>
<td>Return the number of elements in the dataset.</td>
</tr>
<tr>
<td>first()</td>
<td>Return the first element of the dataset (similar to take(1)).</td>
</tr>
<tr>
<td>take((n))</td>
<td>Return an array with the first (n) elements of the dataset.</td>
</tr>
</tbody>
</table>
Designing new interfaces for ROOT data processing

Kalle Vuorinen