Alabaster: Autocomplete Letting Apache Beam Applications Succeed Through Exploration Rapidly

Finn Voichick · fvoichick@wustl.edu · Washington University in St. Louis · Department of Computer Science and Engineering
Brad Myers, Mary Beth Kery · mkery, bam @cs.cmu.edu · Carnegie Mellon University · Human-Computer Interaction Institute

Apache Beam

Powerful API for cloud computing
Very scalable
Same API for bounded and unbounded data
Unpopular compared to Apache Spark, especially among data scientists

Goals

Identify usability problems with Beam
Consider ways to mitigate problems
Provide advice about where Beam maintainers should direct their efforts

Word Count in Apache **Beam**

```
with Pipeline() as p:
  p | ReadFromText(input_file) \
    | FlatMap(lambda line: re.findall(r"[A-Za-z']+", line)) \
    | Map(unicode.lower) \
    | Count.PerElement() \
    | Map(lambda word_count: "%s,%d" % word_count) \
    | WriteToText(output_file)
```

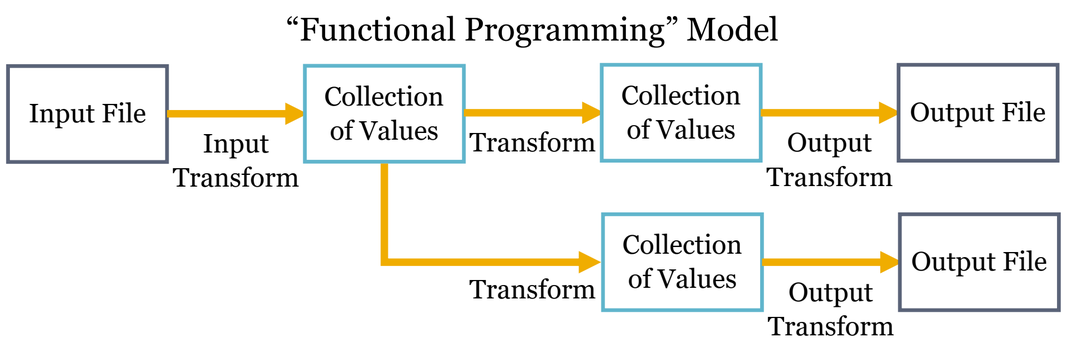
Each transformation is a PTransform subclass.
The pipe operator (“|”) is overloaded to apply the transform to a PCollection.
Usually: applying a PTransform to a PCollection produces another PCollection.
More extensible: users can write their own transformations.

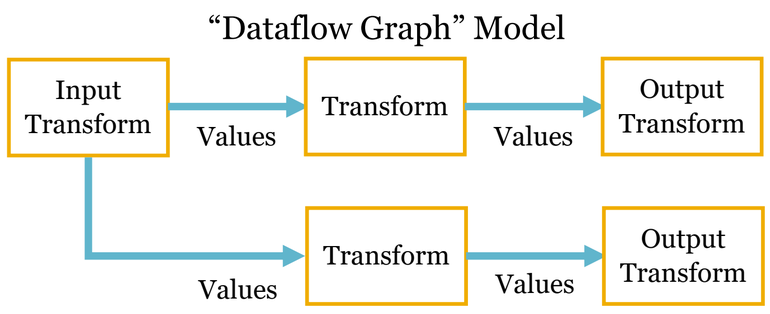
Word Count in Apache **Spark**

```
sc = SparkContext.getOrCreate()
sc.textFile(input_file) \
  .flatMap(lambda line: re.findall(r"[A-Za-z']+", line)) \
  .map(unicode.lower) \
  .map(lambda word: (word, 1)).reduceByKey(operator.add) \
  .map(lambda word_count: "%s,%d" % word_count) \
  .saveAsTextFile(output_file)
```

Each transformation is a method.
Methods are called on RDDs (Resilient Distributed Datasets)
Usually: methods of the RDD class return RDDs.
Less extensible: more difficult to define your own transformations.

Problem: Conflicting Mental Models

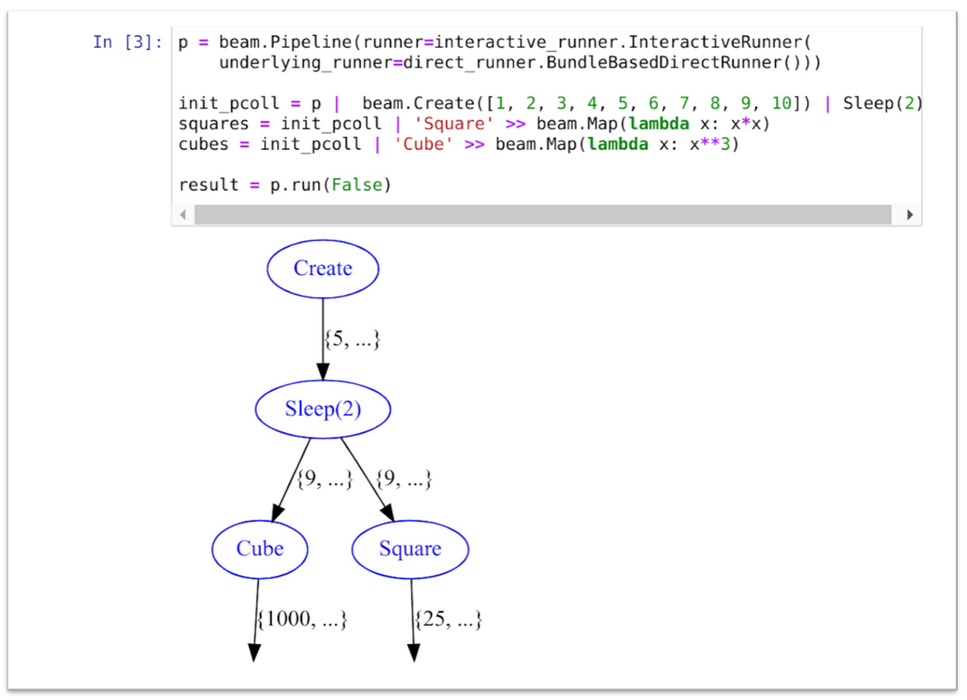




Different areas of documentation use different illustrations.
The API tends to encourage the “functional programming” model.
For programs with joins, the “dataflow graph” model fits better.
Potential solution: use one model throughout the documentation.
Another solution: illustrate and explain the differences in documentation.

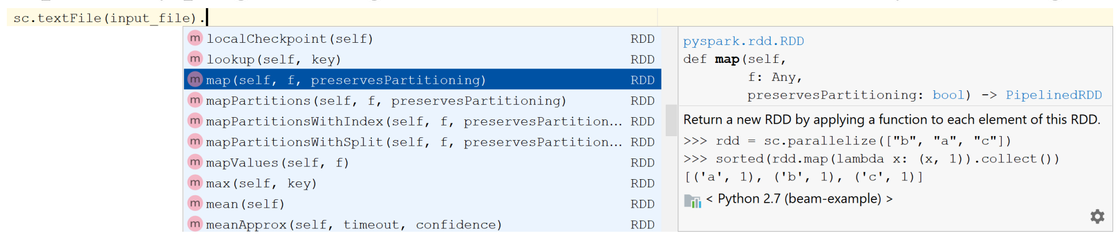
Problem: Lack of Interactivity

Spark has an interactive shell.
In Spark, users can see what’s in their pipeline as they’re building it.
In Beam, users must build up a job and submit it as a whole.
Solution: Beam engineers are creating a Jupyter notebook with visualizations:



Problem: Difficult to Explore

In Spark, IDEs can help with code completion.
Exploratory programming: users don’t need to know what they’re looking for.



With Beam, IDEs don’t understand the pipe operator.
Users must switch between documentation (browser) and code (IDE or editor)
Transform classes are spread between several modules.

Solution: Alabaster

A plugin for PyCharm designed to help newcomers learn faster.
A list of transforms appears upon typing the pipe character.
Upon selection, “boilerplate” code is automatically inserted.
Help is given for transform parameters as well.

This work was funded by a Google Faculty Research Award and National Science Foundation grant #1560137.