Ray Streaming Cross-Language API

1. Motivation

Currently ray streaming has java/python api. Those api serve many use cases for java/python users well. However, current api doesn't cover the use cases for hybrid jobs consisting of java/python workers. Hybrid cross-language streaming jobs can be used to address cases like:

- Python/Java reuses Java/Python connectors. In this way, we don't have to reimplement connectors that we already implemented in java or python. And some source/sink may be easier to implement in java such as hadoop file and kafka source/sink. In this way, all connectors can be shared by java/python api.
- In occasions like online learning, we may need to use streaming java workers to process features/labels, then send features/labels to streaming python workers (which are tensorflow workers).
- In some online recommendation/anti-fraud occasions, we may send features in java to python workers to inference, then send results to java workers for further processing.
- Rewrite some performance-critical parts of a python streaming job using java functions without having to rewrite the whole streaming job using java api.
- Implements some parts of the pipeline that can't be done in one language easily using another language. For example, it's difficult to access hbase/hdfs in python function, we can implement this function using java without rewriting the whole streaming job using java api and vise versa.

All above cases need cross-language streaming api support. So in the document we propose the design and implementation of streaming cross-language api which we mentioned early in the design of streaming python.

2. API

We will provide both java and python cross-language api. Both have exactly the same functionalities, and can describe any sophisticated streaming java/python/cross-lang jobs which can be described by any of those api.

2.1 Java API

```
ds.filter(Example::filter)
  .map(Example::mapper1)
  .asPythonStream()
  .map("module1", "func1")
  .keyBy("module1", "func2")
```

```
  .reduceByKey("module1", "func3")
  .asJavaSAtream()
  .map(Example::map)
  .asPythonStream()
  .sink(new Sink());
```

2.2 Python API

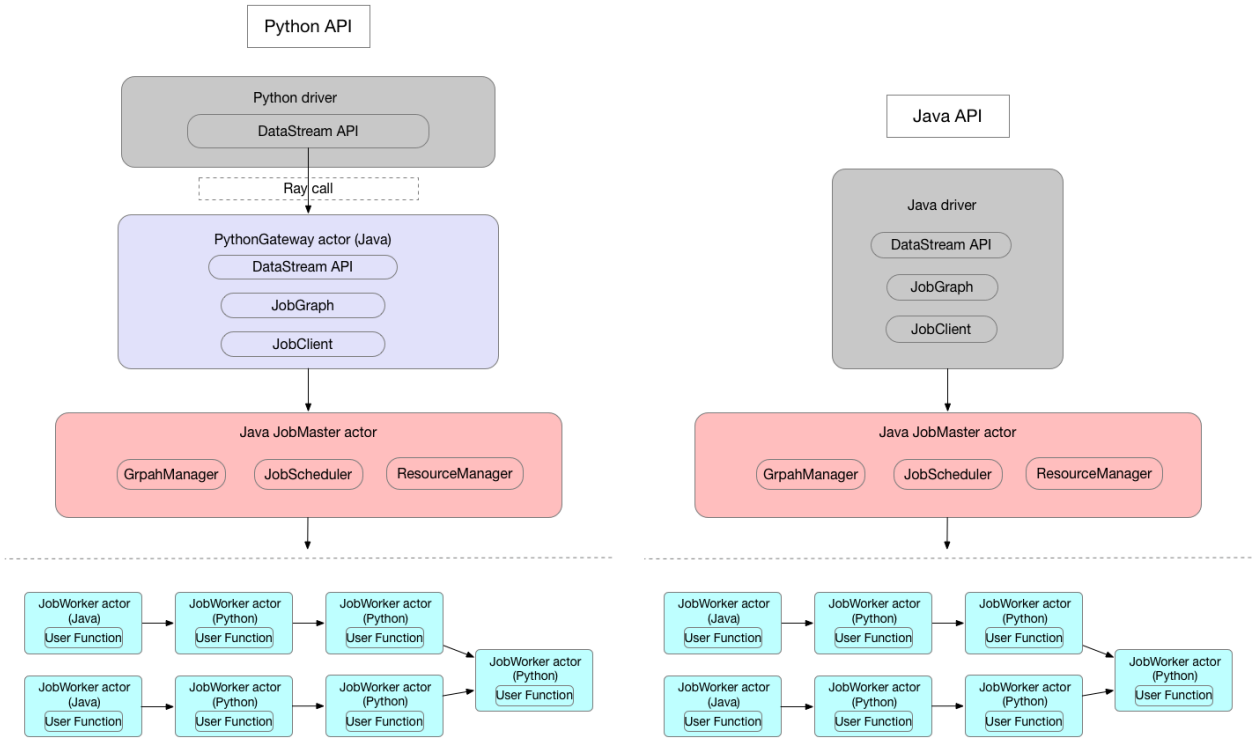
```
ds.filter(func1)
    .map(func2)
    .as_java_stream()
    .map("com.example.Mapper")
    .key_by("com.example.KeyExtractor")
    .reduce_by_key("com.example.ReduceFunction")
    .as_python_stream()
    .map(func3)
    .as_java_stream()
    .sink("com.example.Sink")
```

For now, we only support transfer data of **numeric types, string, list, map and their composition** between java/python workers and let the users take up serialization/deserialization work for other types between java/python in user function. Automatic schema based serialization which discussed in 3.3.2 detailly will be supported later.

Note that **partition related methods in data stream can not be called immediately after converting from java/python stream to python/java stream, such as partitionBy()/keyBy()/partition_by()/key_ky(**). This is because these functions will be executed in a data stream before the converted stream, and a java/python worker can't execute a python/java partition function. Calling these methods on converted streams intermediately will throw runtime exceptions. If the user does need to call partition related methods in a converted stream immediately, the user can call map method with an identity function first, then call partition related methods.

3 Implementation

Users can use java/python api to build cross-lang job pipelines and submit the job graph to the JobMaster for scheduling and execution.
PythonGateway receives ray calls from python driver to map data stream api calls in python to data stream api calls in java. See more details in [streaming python api doc].



3.1 Cross-Lang Graph/Scheduling

Add PythonFunction/PythonPartition/PythonDataStream in streaming-api, and extend JobGraph/ExecutionGraph to express hybrid java/python data flow. Make JobScheduler to support schedules for both java and python workers. This part is designed in streaming python api doc and has been done in cross-lang graph PR and python api PR.

3.2 DataStream Conversion

For java and python api, we both need datastream class to represent java/python datastream. For java, we already added PythonDataStream, PythonKeyDataStream, PythonStreamSource and PythonStreamSink in the cross-lang graph PR. We need to add JavaDataStream, JavaKeyDataStream, JavaStreamSource and JavaStreamSink in python to represent all java streams.

We also need to add steam conversion methods to datastream to convert between java and python data stream, so that we can apply corresponding java/python stream transformations:

- Convert java stream into python stream
    - Add asPythonStream() for stream in java api
    - Add as_python_stream() for stream in python api. When this method is called, it will call asPythonStream() in java api by forwarding call to java PythonGateway actor too, and return a python stream in python.
- Convert python stream into java stream

    - Add asJavaStream() for stream in java api
    - Add as_java_stream() for stream in python api. When this method is called, it will call asJavaStream() in java api by forwarding call to java PythonGateway actor too, and return a java stream in python.

The converted stream and raw stream are the same logical stream which have the same logical id. Changes such as parallelism and config in any of them will be reflected in the other stream.

3.3 Serialization

We will implement cross-language data serialization between java/python workers. The XLangSerializer will handle serialization in `OutputCollector#collect` method.

3.3.1 CrossLangSerializer

All elements in stream can be classified into a Record/KeyRecord//Watermark object, these objects capture extra information that are necessary for streaming runtime. For user data itself, we will use schema or msgpack for serialization. For the Record/KeyRecord/Watermark object, we need to do some extra serialization work. CrossLangSerializer is implemented both in java/python so that a serialized element in java/python workers can be deserialized in python/java workers.

For now, boolean/byte/short/int/long/double/binary/string/collection/map are supported between java and python using msgpack. Custom type serialization is not supported now. Users need to use a map function to handle custom type serialization.

3.3.2 Schema

All datastream conversion methods `asJavaStream()/asPythonStream()/as_java_stream()/as_python_stream()` can accept a schema as an optional argument. If the schema is not passed and the schema can't be inferred from data class, only pure binary is allowed to be transferred between java/python workers. Schema will be used as a data format to serialize/deserialize data between Java/Python workers. For now, we only support transfer of pure binary data between java/python workers and let the users take up serialization/deserialization work. Automatic schema based serialization will be supported in future after we open source our cross-language serialization framework which is used for transferring message objects between java/python automatically.

Actually, there is no need to specify schema manually in python api, because we can extract input argument and return value type from upstream/downstream java function.

4 Work Plan

- Represent java DataStream in python
- DataStream Conversion

- Serialization between java/python workers