Elastic Spark Programming Framework (ESPF)

A Dependency-Injection Based Programming Framework for Spark Applications

Bruce Kuo, Software Engineer, APAC Data, email: bruce3557@yahoo-inc.com

Outline

- Motivation & Related Work
- Prerequisite
- Programming Framework
- Integration with Components
- Conclusion
- Q&A

Motivation & Related Work

Native Spark Application

```
public class GainsChartDataGeneration {
    public static void main(String[] args) {
        String sortedPredictionResultTable = args[0];
        String gainTable = args[1];

        SparkConf conf = new SparkConf();
        JavaSparkContext sc = new JavaSparkContext(conf);
        HiveContext sqlContext = new HiveContext(sc.sc());

        DataFrame dataFrame =
            sqlContext.table(sortedPredictionResultTable)
                .select("target", "score");

        // Generate schema
        ...
        StructType schema = DataTypes.createStructType(newFields);

        long totalCount = dataFrame.count();
        List<Row> seqList = new ArrayList<>();
        for (long i = 100; i <= totalCount; ++i) {
            long curCount =
                dataFrame
                    .limit((int) i)
                    .filter("target=1")
                    .count();

            seqList.add(RowFactory.create(i, curCount));
        }

        JavaRDD<Row> resultRDD = sc.parallelize(seqList);
        sqlContext
            .createDataFrame(resultRDD, schema)
            .write()
            .mode(SaveMode.Overwrite)
            .saveAsTable(gainTable);
    }
}
```

Initialization

Main logic

Output

YAHOO!

Native Spark Application (Cont.)

Handle arguments setting in every application

```
String sortedPredictionResultTable = args[0];
String gainTable = args[1];
```

Native Spark Application (Cont.)

Initialize Spark environment

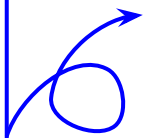
```
SparkConf conf = new SparkConf();
JavaSparkContext sc = new JavaSparkContext(conf);
HiveContext sqlContext = new HiveContext(sc.sc());
```

Submit Spark Application

With spark-submit shell command

- Need to specify application settings every time
- It is kind of wordy
- Not intuitive enough to know the function of an argument

```
spark-submit
--master yarn-client
--driver-memory 12G
--class com.yahoo.ecdata.generation.GainsChartDataGeneration
--num-executors 300
--executor-memory 12G
--conf spark.executor.cores=4
--conf  spark.ui.view.acls=*
--conf spark.kryoserializer.buffer.max.mb=1024
--conf spark.akka.frameSize=1024
--queue adhoc
experiment.jar
mining_predict_result stats_gains_table
```



Environment configuration settings in every submission

If an Application is a Module...

- Change application arguments in command line or script
  - e.g., change the data source and output path for ETL applications or model arguments for machine learning applications

Experment 1 - weight = 0.1, min_val = 1	**Experment 2 - weight = 0.3, min_val=2**
spark-submit --master yarn-client ... ... experiment.jar 0.1 1 path_1	spark-submit --master yarn-client ... ... experiment.jar 0.3 2 path_2

1. Need to read document to know which argument is for weight or min_val
2. Change the value in script is not intuitive

If an Application is a Module… (Cont.)

- When the number of applications is large, there are many triggering scripts in the system
  - It may cause huge maintenance effort if developers want to change a configuration

```
spark-submit
--master yarn-client  mesos
...
--class X
experiment.jar
mining_predict_result stats_gains_table
```

```
spark-submit
--master yarn-client mesos
...
--class Y
experiment.jar
A B C
```

Need to change many scripts one by one

Oozie Spark

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="spark_oozie_wf">
  <start to="spark-node"/>
  <action name="spark-node">
    <java>
      <job-tracker>${jobTracker}</job-tracker>
        <name-node>${nameNode}</name-node>
        <prepare>...</prepare>
          <configuration>...</configuration>
            <main-class>org.apache.spark.deploy.SparkSubmit</main-class>
            <arg>--master</arg>
            <arg>yarn-client</arg>
            <arg>com.yahoo.ecdata.generation.GainsChartDataGeneration</arg>
            <arg>--properties-file</arg>
            <arg>spark-defaults.conf</arg>
            <arg>--num-executors</arg>
            <arg>300</arg>
            <arg>--executor-memory</arg>
            <arg>16g</arg>
            <arg>--driver-memory</arg>
            <arg>16g</arg>
            <arg>--queue</arg>
            <arg>adhoc</arg>
            <arg>experiment.jar</arg>
            <arg>mining_predict_result</arg>
            <arg>stats_gains_table</arg>
            <capture-output/>
        </java>
        ...
    </action>
    <kill name="fail">...</kill>
    <end name='end' />
</workflow-app>
```


YAHOO!

Airflow: Bash Operator + Jinja Template

```
templated_command = (
    "spark-submit --master yarn-client --queue adhoc --num-executors 300 \
    --driver-memory 16g --class {{ params.main_class }} {{ params.jar_file }} \
    {{ params.args }}"
)

def i_am_a_function(param1, param2, **kwargs):
    print(kwargs.get('execution_date')) ##airflow macro

with DAG('dag_name', default_args=default_args, schedule_interval='0 5 * * *')
as dag:
    (
        PythonOperator(
            task_id = 'task1',
            python_callable = i_am_a_function,
            op_args = [param1, param2],
            provide_context = True)
        << [BashOperator(
                task_id = 'task2',
                bash_command = templated_command,
                params = {
                    'jar': 'experiment.jar',
                    'main_class':
'com.yahoo.ecdata.generation.GainsChartDataGeneration',
                    'args': 'mining_predict_result stats_gains_table'
                }
            )
            ]
    )
```

parameterized arguments

YAHOO!

Airflow: Bash Operator + Jinja Template (Cont.)

Template command

```
templated_command = (
    "spark-submit --master yarn-client --queue adhoc --num-executors 300 \
    --driver-memory 16g --class {{ params.main_class }} {{ params.jar_file }} \
    {{ params.args }}"
)
```

Airflow: Bash Operator + Jinja Template (Cont.)

Parameterized arguments

```
params = {
    'jar': 'experiment.jar',
    'main_class': 'com.yahoo.ecdata.generation.GainsChartDataGeneration',
    'args': 'mining_predict_result stats_gains_table'
}
```

Possible Problems in Oozie & Airflow

- Need to change default environment settings
- Hard to know how many arguments and the meaning of these arguments
- Is it possible to generate application configuration automatically for different purpose?

Prerequisite

Dependency Injection

- In short words, objects are configured by an external entity
  - e.g., Dao dao = new HiveDao(...); *// HiveDao extends Dao*
- Benefits
  - Reduced dependencies, e.g., any dao implementation can use in a data-access code
  - More testable code, e.g., Dao dao = new TestDao(...);
  - More reusable code
  - More readable code
- Dependency injection frameworks
  - Spring
  - Google Guice

Java Annotation

- Add metadata to a variable, a method or a class.
- Using reflection can help program know the attributes of the fields listed above and provide basic control at runtime.

```
@Column(length = 32) // Truncate column value to 32 characters.
private String name;
```

Programming Framework

Pseudo Solution

```
public class SparkApp {
    // All class variables are injected automatically
    String input; // args[0]
    String output; // args[1]
    Double weight; // args[2]

    public int execute() {
       // Using arguments and Spark context directly
       sparkContext.load(input);
       // Computing logics
       ...
       output.saveAsText(output);
       ...
    }
}
```

How to Make a Class Knows Parameters?

- Annotation + Runtime Injection!

```
@Input(name="input")
String input;

@Output(name="output")
String output;

@ModelParam(name="weight", type="double")
Double weight;
```

Application Initialization

- Spark Environment Initialization
  - SparkConf
  - SparkContext
  - SQLContext or HiveContext if application needs
- Variable Initialization
  - Inject variables with the corresponding argument
  - Handle type casting

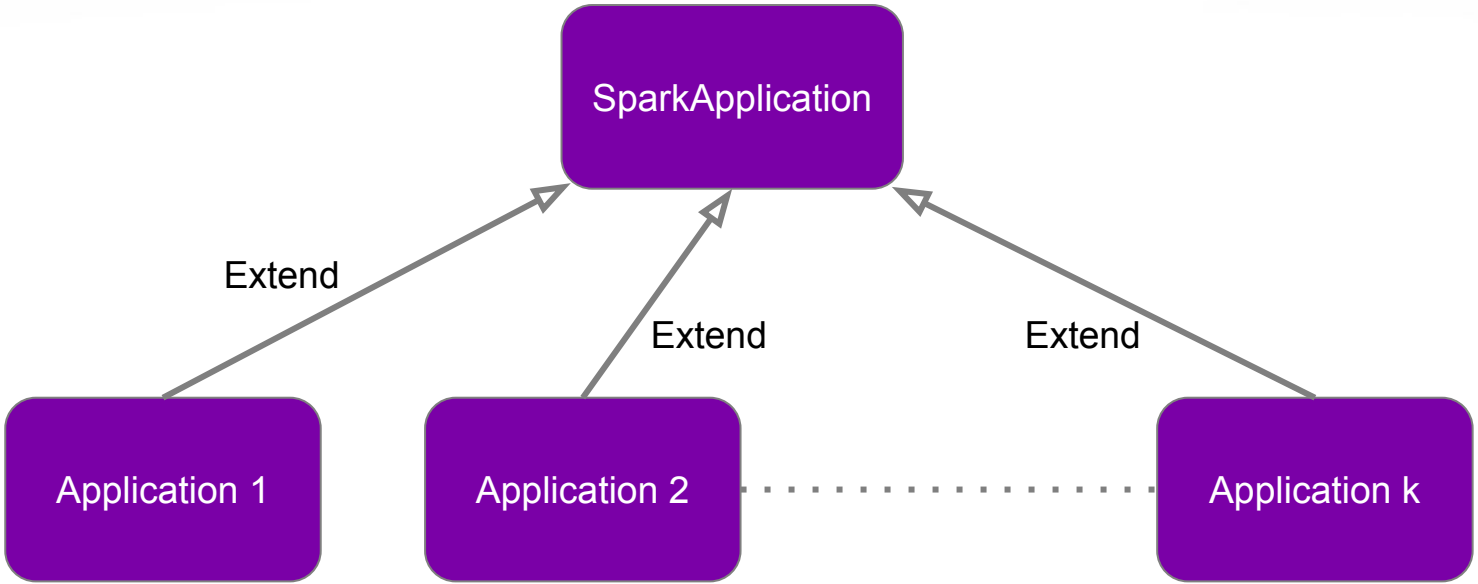
Base Class

- Initialize Spark environment
- Inject variables with corresponding arguments
- Run Spark code section

Definition of Base Class - SparkApplication

```
public abstract class SparkApplication {
    ....

    public void initialize() { // initialize spark related configuration }

    protected abstract int execute() throws Exception; // put your code here

    // use annotation to help user setting configuration
    protected void setArguments(String[] args) throws Exception { ... }

    public static final void main(String[] args) throws Exception {
       // make main function final to avoid override
       initialize();
       setArgs(args);
       execute();
       ....
    }
}
```

Programming Framework



Supported Annotations

- `@Input`
  - name
  - type: table or file path
- `@Output`
  - name
  - type: table or file path
- `@TableParam`
  - table name
  - column
  - datatype: use for type casting
- `@ModelParam`
  - name
  - required

Example: Gains Chart Data Generation

- Definition of gains chart
  - The gains chart plots the values in the Gains(%) column from the table.
  - Gains are defined as the proportion of hits in each increment relative to the total number of hits in the tree, using the equation.

Example: Gains Chart Data Generation (Cont.)

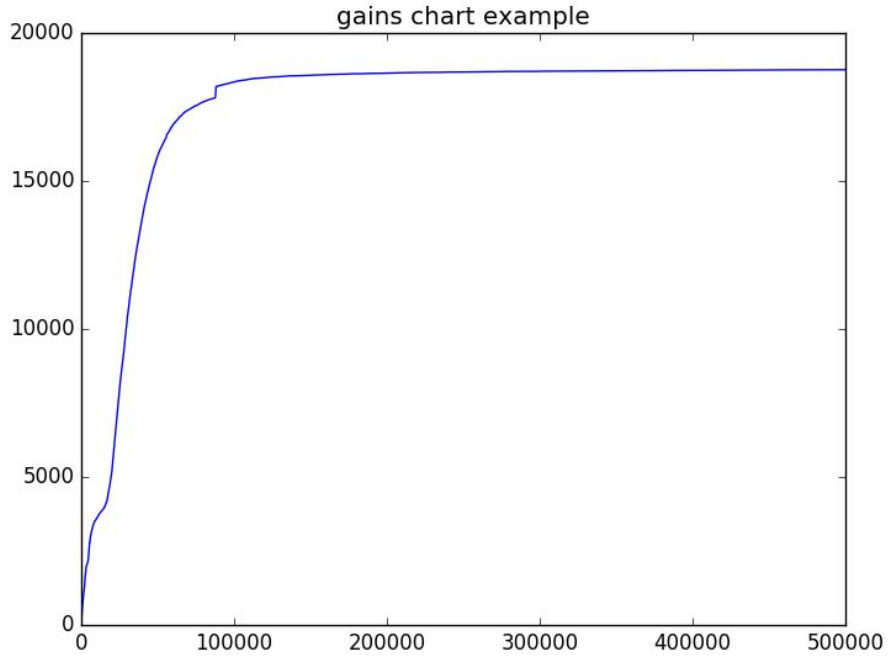
Sorted Predicting Results

Target	Score
1	880
1	724
1	676
1	556
0	480
0	368

Gains Table

Count	Target Count
100	36
200	54
300	66
400	76
500	85
600	90

Example: Gains Chart Data Generation (Cont.)



YAHOO!

Example: Gains Chart Data Generation (Cont.)

```
public class GainsChartDataGeneration extends SparkApplication {
    @Input(name="srcTable", type="hive")
    String sortedPredictionResultTable;

    @Output(name="destTable", type="hive")
    String gainTable;

    @Override
    protected void execute() throws Exception {
        DataFrame dataFrame =
            sqlContext.table(sortedPredictionResultTable)
                      .select("target", score");

        // Generate schema
        ...
        StructType schema = DataTypes.createStructType(newFields);

        long totalCount = dataFrame.count();
        List<Row> seqList = new ArrayList<>();
        for (long i = 100; i <= totalCount; ++i) {
            long curCount = dataFrame.limit((int) i)
                    .filter("target=1").count();
            seqList.add(RowFactory.create(i, curCount));
        }

        JavaRDD<Row> resultRDD = sparkContext.parallelize(seqList);
        DataFrame resultDf = sqlContext
            .createDataFrame(resultRDD, schema)
        writeOutput(resultDf, gainTable, SaveMode.Overwrite);
    }
}
```

Example: Gains Chart Data Generation (Cont.)

Initialize variables with annotation. The framework set variables.

```
@Input(name="srcTable", type="hive")
String sortedPredictionResultTable;

@Output(name="destTable", type="hive")
String gainTable;
```

Example: Gains Chart Data Generation (Cont.)

Only override execute method

```
@Override
protected void execute() throws Exception {
    DataFrame dataFrame =
        sqlContext.table(sortedPredictionResultTable)
                  .select("target", score");

    // Generate schema
    ...
    StructType schema = DataTypes.createStructType(newFields);
    long totalCount = dataFrame.count();
    List<Row> seqList = new ArrayList<>();
    for (long i = 100; i <= totalCount; ++i) {
        long curCount = dataFrame.limit((int) i)
                .filter("target=1").count();
        seqList.add(RowFactory.create(i, curCount));
    }

    JavaRDD<Row> resultRDD =
sparkContext.parallelize(seqList);
    DataFrame resultDf = sqlContext
        .createDataFrame(resultRDD, schema)
    writeOutput(resultDf, gainTable, SaveMode.Overwrite);
}
```

Framework Sugar

- Improve code readability
- Semantic programming
- Ease effort of unrelated logics

Integration with Components

Power of Annotations

- Scan / inject class fields at runtime
- Help other programs easily get arguments

Class - SparkAnnotation

- Save the data and value of the annotation
  - type of annotation (Input, Output…) and its metadata
  - the value of this field

- Serializable

Interface - SparkAnnotationGetter

- Return a map contains all spark annotations

```
public static Map<String, SparkAnnotation>
getSparkAnnotations(SparkApplication sparkApplication)
```

Interface - SparkAnnotationSetter

- Set a variable with corresponding arguments
- Handle type casting
- Used in SparkApplication class

```
public static void setSparkAnnotations
(Field field, SparkAnnotation sparkAnnotation)
```

Class - SparkAppMetadata

- Store application settings
  - Application name
  - Spark environment setting
  - Spark application class
- Store a map of SparkAnnotation from a SparkApplication
- Serializable

Configuration Auto-Generator

- Using SparkApplicationGetter to set SparkAppMetadata with JSON serializer can generate application configuration

```
{
  "name" : "User Prediction Results",
  "input" : [ {
    "name" : "srcTable",
    "value" : "sortedPredictionResults",
    "type" : "hive",
    "fields" : {
         "target": "ta",
         "Score": "sc"
     }
  } ],
  "output" : [ {
      "name" : "destTable",
      "value" : "gainChartData",
      "type" : "hive",
      "fields" : {
          "count": "count",


      }
  } ],
  "sparkConfig" : { },
  "modelArgs" : {},
  "mainClass" : "com.yahoo.ecdata.ml.GainsChartDataGeneration"
}
```

New Spark Application Submitter

- Submit job from configuration
  - Command line submitter
  - Programmatic submitter
- Translate configuration for annotation-based fields to input arguments
- Control resource if needs (or use default setting)

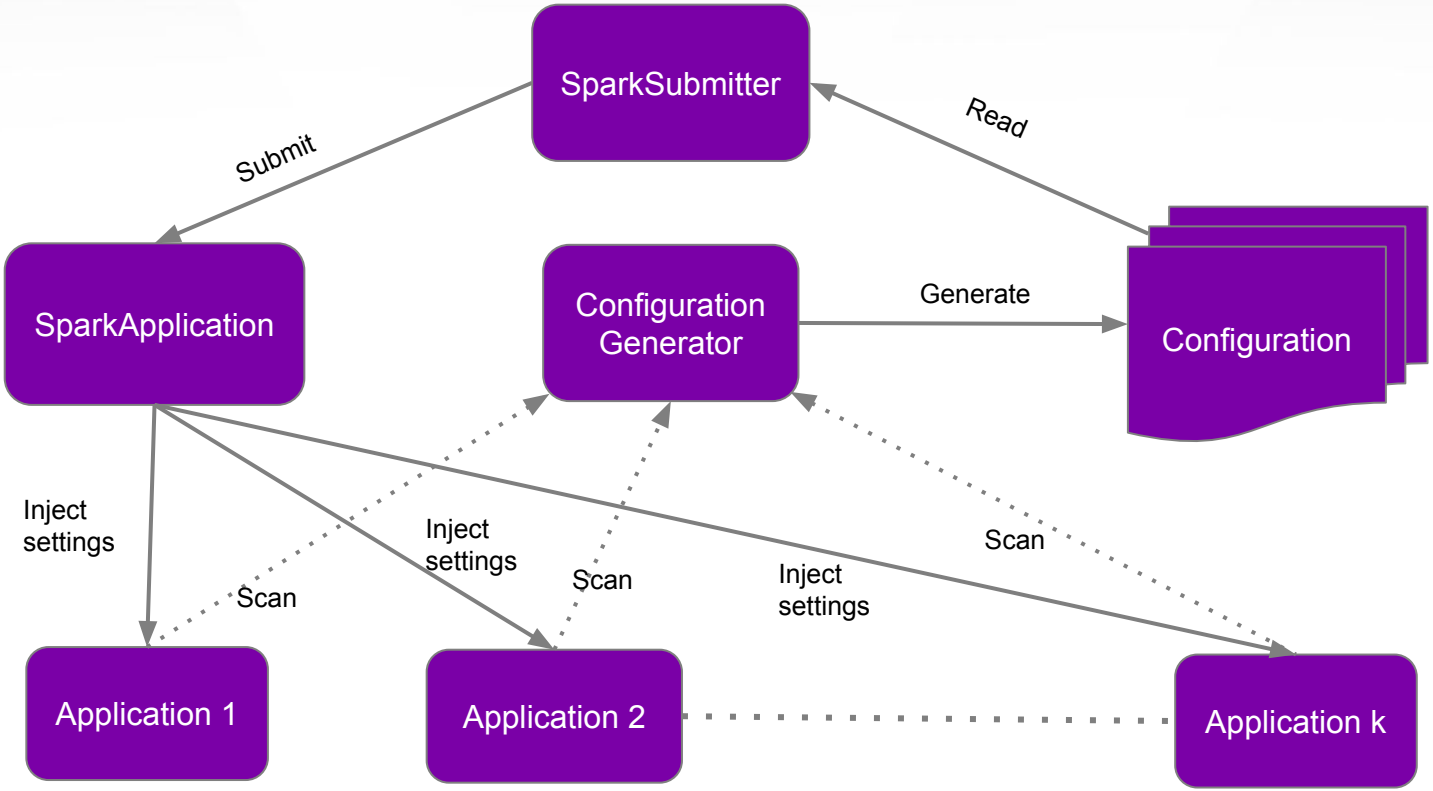
Example: Command Line Submitter

```
SparkSubmitter run -class com.yahoo.ecdata.generation.GainsChartDataGeneration

Missing required options: input, output

Configuration format for com.yahoo.ecdata.generation.GainsChartDataGeneration:
 -class <class>              Spark Job class full name
 -conf <arg>                 JSON file for setting job args and Spark configs
 -sparkConf <key=value>      Spark Configurations
 -output <type=_type,value=_value>         [Output, filepath or hive path]
 -input <type=_type,value=_value>           [Input, filepath or hive path]
```

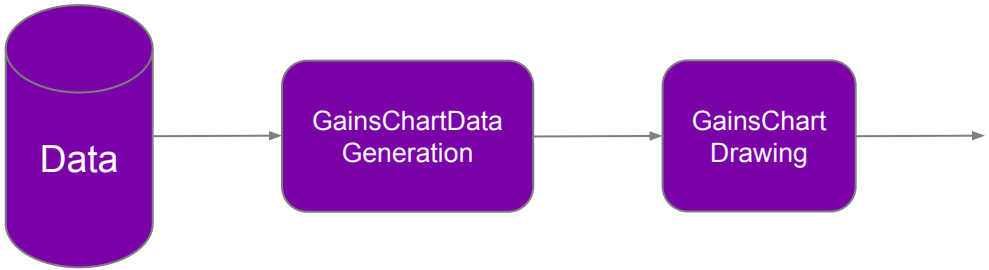
Example: Command Line Submitter (Cont.)

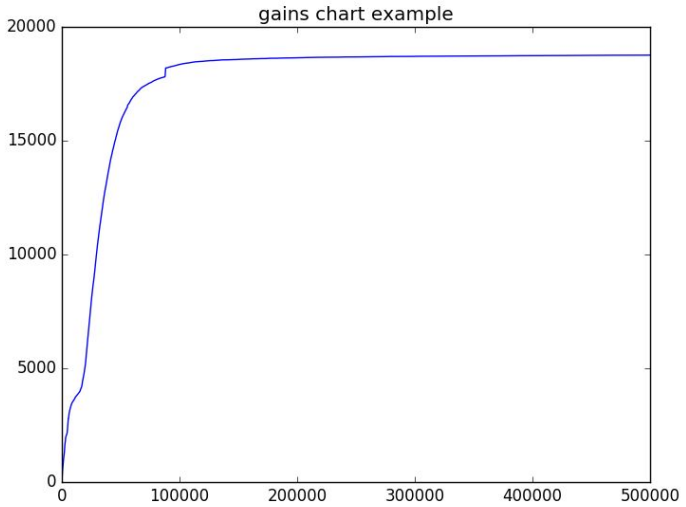
```
SparkSubmitter run -class com.yahoo.ecdata.generation.GainsChartDataGeneration -conf
test_setting.json

The config is invalid for this class

Configuration format:

    {
      "name" : "User Prediction Results",
      "input" : [
        { "name" : "input", "value" : "...","type" : "..." }
      ],
      "output" : [
        { "name" : "output", "value" : "...", "type" : "..." } ],
      "sparkConfig" : { },
      "modelArgs" : {},
      "mainClass" : "com.yahoo.ecdata.ml.GainsChartDataGeneration"
    }
```

Integration Framework Diagram



Example: Gains Chart Drawing Flow

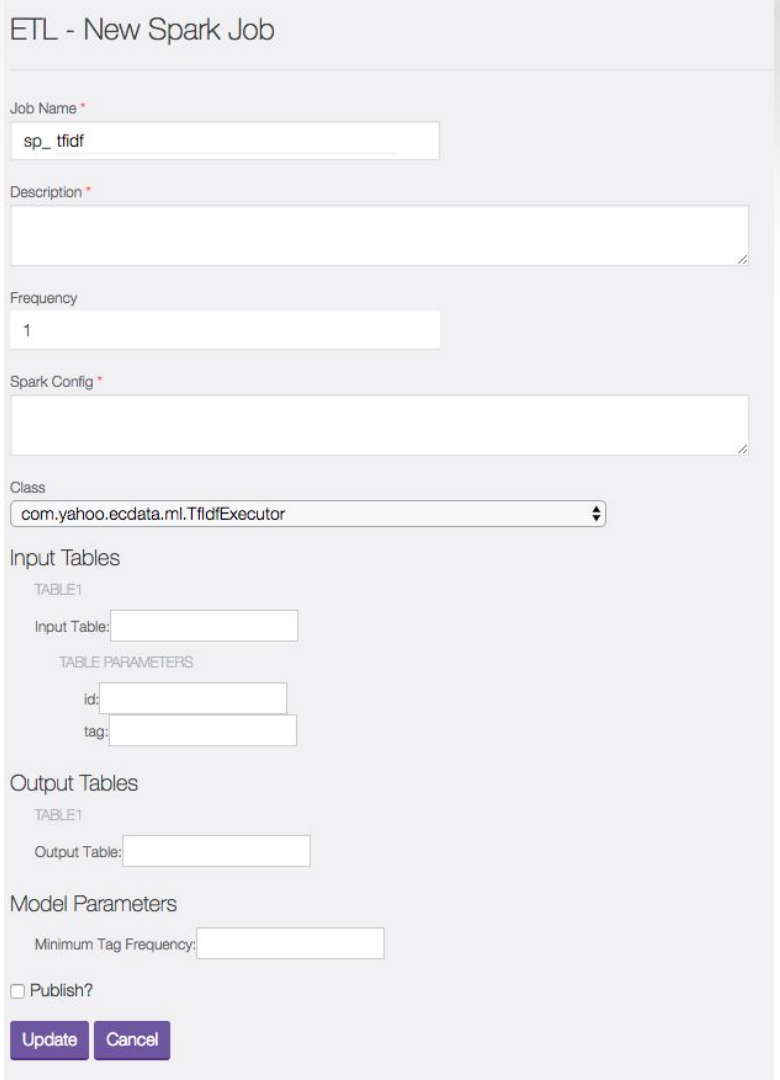




YAHOO!

Example
Spark Web Configurator

- Users can submit applications from web UI



YAHOO!

Conclusion

Comparison of Frameworks

	ESPF	Oozie	Airflow
Configuration Generator	v	x	x
Parameter Understanding	v	x	x
Flow Control	x	v	v
Scheduling	x	v	v
Maintenance	Easy	Native	Native

Future Work

- Provide flow control between applications
- Control resource automatically
  - Collect application statistics to predict the resources
- Open source to community (work in progress)

Conclusion

- Simple and flexible framework for JVM-based languages
  - e.g, Java, Scala
  - Currently not support pySpark
- Ease the maintenance effort in large system
  - Easy for testing with changing configuration
  - Code is the documentation
  - Focus on business logic over configuration

Acknowledgement

- Jason Lin, Lucas Yang, Sas Chen, Norman Huang, Evans Ye
- Yahoo APAC Data Team

Q&A