Rule Engine

Quartic.ai Coding Challenge

Requirements

- Build a rule engine that will apply rules on streaming data. Your program should be able to perform following tasks, at minimum:
  - Allow users to create rules on incoming data stream
  - Execute rules on incoming stream and show the data that violates a rule.

Incoming data stream is a tagged data stream. Each incoming data is a hashmap with following syntax

```
{ 'signal': 'ATL1', 'value': '234.98', 'value_type': 'Integer'}
{ 'signal': 'ATL2', 'value': 'HIGH', 'value_type': 'String'}
{ 'signal': 'ATL3', 'value': '23/07/2017 13:24:00.8765', 'value_type': 'Datetime'}
...
...
...
```

In general, a data unit would have three keys

- **signal:** This key specifies the source ID of the signal. It could be any valid alphanumeric combo. ex: ATL1, ATL2, ATL3, ATL4
- **value:** This would be the actual value of the signal. This would always be a string. ex: '234', 'HIGH', 'LOW', '23/07/2017'
- **value_type:** This would specify how the value is to interpreted. It would be one of the following

Integer: In this case the value is interpreted to be an integer. Ex: '234' would be interpreted as 234
String: In this case the value is interpreted to be a String. Ex: 'HIGH' would be interpreted as 'HIGH'
Datetime: In this case the value is interpreted to be a Date Time.

Rules can be specified for a signal and in accordance to the value_type. Some examples of rules are:
ATL1 value should not rise above 240.00
ATL2 value should never be LOW
ATL3 should not be in future

Your rules need to be stored on a persistent storage format (text/yaml/ini file, database, etc.) independent of the source code.

Output :
For each input data signal, only emit the name of the source that was violated by a rule specified.

Assumptions:

- Build a rule engine that will apply rules on streaming data. Your program should be able to perform following tasks, at minimum:
  - 1 Allow users to create rules on incoming data stream
  - Execute rules on incoming stream and show the data that violates a rule. 2

Incoming data stream is a tagged data stream. Each incoming data is a hashmap with following syntax

3
```
{ 'signal': 'ATL1', 'value': '234.98', 'value_type': 'Integer'}
{ 'signal': 'ATL2', 'value': 'HIGH', 'value_type': 'String'}
{ 'signal': 'ATL3', 'value': '23/07/2017 13:24:00.8765', 'value_type': 'Datetime'}
...
...
...
```

4

In general, a data unit would have three keys

- **signal:** This key specifies the source ID of the signal. It could be any valid alphanumeric combo. ex: ATL1, ATL2, ATL3, ATL4
- **value:** This would be the actual value of the signal. This would always be a string. ex: '234', 'HIGH', 'LOW', '23/07/2017'
- **value_type:** This would specify how the value is to interpreted. It would be one of the following

Integer: In this case the value is interpreted to be an integer. Ex: '234' would be interpreted as 234
String: In this case the value is interpreted to be a String. Ex: 'HIGH' would be interpreted as 'HIGH'
Datetime: In this case the value is interpreted to be a Date Time.

Rules can be specified for a signal and in accordance to the value_type. Some examples of rules are:
ATL1 value should not rise above 240.00
ATL2 value should never be LOW
ATL3 should not be in future

5

Your rules need to be stored on a persistent storage format (text/yaml/ini file, database, etc.) independent of the source code.

Output :
For each input data signal, only emit the name of the source that was violated by a rule specified.

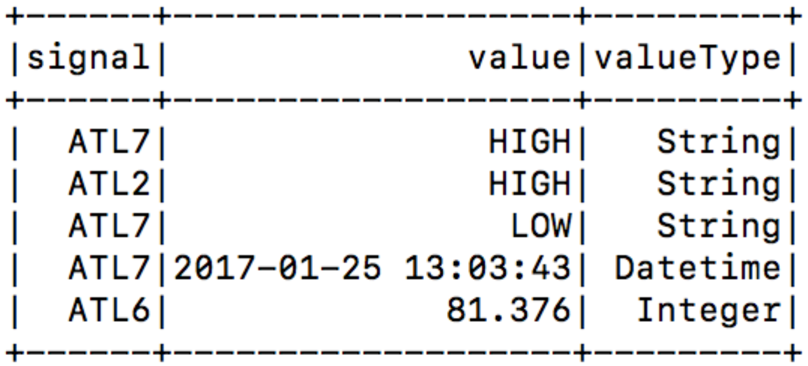
Assumptions:

1 **Allow users to create rules :** Since no specific form of UI/HMI was mentioned for the user to input the rule, I have assumed no particular input mechanism(UI/API service) needs to be developed. The user will now be able to enter the rules through the terminal.

2 **Show the data that violates a rule :** Since no specific form of UI/HMI was mentioned to show the data that violates the rule, the data is now displayed in the console.

3 **Incoming data is tagged data stream:** Since only the nature of the data is mentioned and the range is not mentioned, signals with Integer-typed signal values have been assumed to range between 0 and 1000 and the Datetime is assumed for the year of 2018.
Also, even though the value type is Integer, the value could be float. In which case, for the rule check the number is converted to float and used.

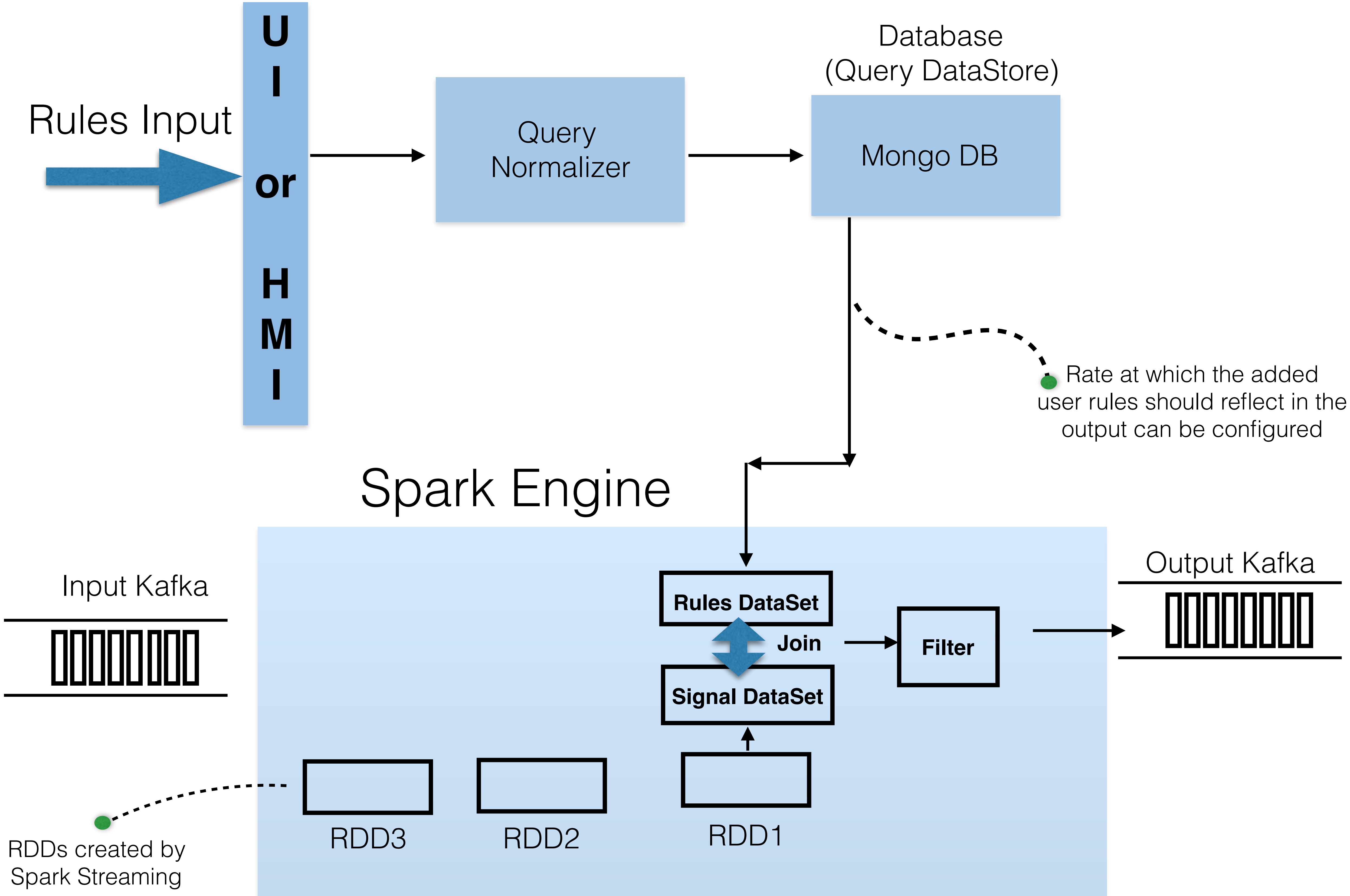
4 **A data unit would have 3 keys :** It has not been mentioned if the mapping between signal and its value_type is one-to-one. It was observed in the input data set, that one signal could have more that one value_type. Therefore, signal to value_type is assumed to be one-to-many.



```
+------+-------------------+---------+
|signal|              value|valueType|
+------+-------------------+---------+
|  ATL7|               HIGH|   String|
|  ATL2|               HIGH|   String|
|  ATL7|                LOW|   String|
|  ATL7|2017-01-25 13:03:43| Datetime|
|  ATL6|             81.376|  Integer|
+------+-------------------+---------+
```

5 **Format of the rules:** The example rules that have been mentioned are "English" like sentences. I have assumed that the there is a fixed format in which the rules can be input and there exists a **"query normalizer"** which validates the input sentences and converts it into appropriate format which can be used for processing the input data stream. I have assumed a particular format for the rules, which will be discussed in the next section.
Also it is assumed that there may be multiple rules applied on the same signal. Though this assumption does not change the design or implementation.

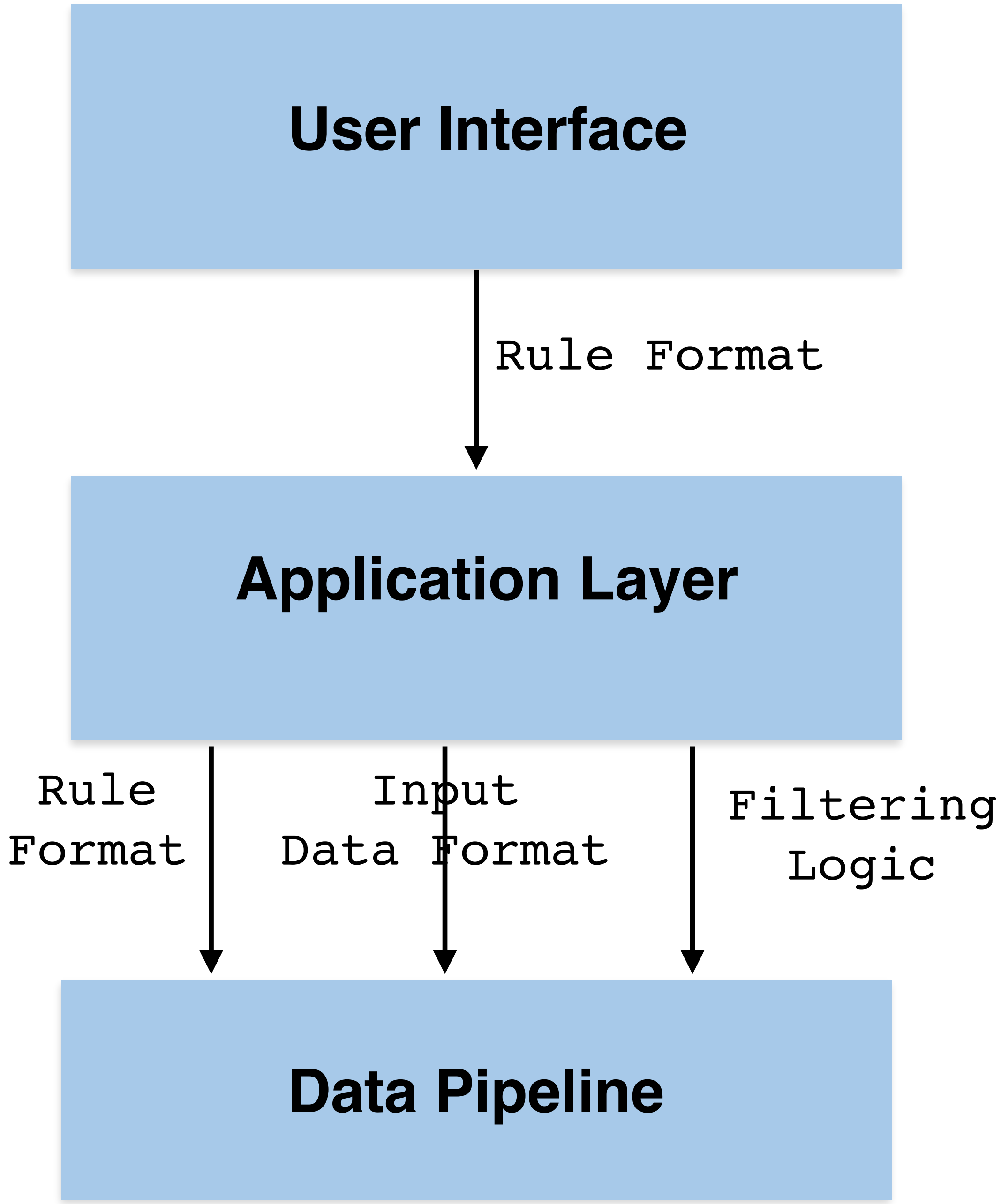
Solution Overview:



Layered Architecture:

- UI or HMI
- Query Normalizer

- Any changes in the way user specifies the rules, only this layer is affected.



User Interface

Rule Format

Application Layer

Rule Format

Input Data Format

Filtering Logic

Data Pipeline

- Input Data Handling
- Rule Data Handling
- Filtering Logic

- Any changes in the number of input signals, nature of the data, nature of the rules or any changes in the filtering logic is handled by this layer.
- This layer is not concerned about how the filtering computation is actually applied on the data.

- Kafka Handling
- Spark Handling
- Computation

- Any changes in the tools or any optimizations in the computation, such as creating more parallelism, or using a more efficient operation is handled in this layer.
- This layer is not concerned with what are fileds in the data and rules and what is the filtering logic implemented.

Testing

Unit testing : Testing was performed using JUnit and Maven-SureFire
The test plans can be found in the test folder.

IntegrationTesting:

Set-up : Zookeeper and Kafka servers initiated.
The kafka messages are sent at the rate of 1 message per second.
Mongo DB loaded with a sample set of normalized queries.
Start rule engine.
The output dataframe is displayed to console and tested.

Complexity

- During each window of the Stream processor, the operations involved are **Join and Filter.**
- Let M be the number of records in the Rules Data Set and let N be the number of records in the Signal DataSet. The value of N depends on the time duration window of the streaming process and QPS of the input data.
- Join, if HashJoin is used the complexity is (N+M). As on now, it would be M*N
- In the filter operation, there are comparisons to make. Even if we consider the worst case, then it would be about ~ 20 operations per record.
- The complexity is upper bounded number of records, which is itself bounded by M*N
- Therefore the complexity is O(MN)

Improvements

- Need to complete the requirements : in terms of required output formating and handling dates
- Need to include the metrics
- Exceptions Handling is not included.
- Currently kafka partitions are not configured.
- The output is just displayed onto console. It needs to be written out to kafka