Distributed Pattern Matching for Apache Flink

Introduction

Pattern matching over event streams is a very important application and currently used in sensitive contexts such as fraud detection and stock monitoring. This project proposes to add basic pattern matching functionality to the existing Flink Streaming API.

Project Goals

Apache Flink already has a Stream Processing API that supports basic operations on streams such as reduce, map, fold, etc. as well as a few advanced features such as grouped transformations and windowing of data streams. Very basic patterns such as, `eventA` followed immediately by `eventB`, `(eventA, eventB)`, can be detecting in a `DataStream` using a simple window-filter-aggregate pipeline. But for more complex patterns, such as `eventA` followed by a series of events that are not `eventB` ending with `eventC`, `(eventA, !eventB[ ], eventC)`, a richer language would be required to be able to express such constructs.

The goal of this project would be to implement a basic pattern matching API, built on the existing functions exposed by the Stream Processing API. We propose a simple API that can detect patterns of reasonable complexity. For more complex pattern queries, we propose a Domain Specific Language that transforms the pattern query into a JobGraph for Flink.

Implementation

The end result of the implemented pattern matching API would look something as follows : Consider the DataStream of events emitted in a RFID-based retail management. The following query attempts to detect shoplifiting activity by reports events where an item is picked and taken out of the store without being registered or checked out.

```
val myDataStream[DataStreamObject] = ...
val eventA = Event(DataStreamObject.name, "Take", EventType.NORMAL)
val eventB = Event(DataStreamObject.name, "Register", EventType.KLEENESTAR)
val eventB = Event(DataStreamObject.name, "Exit", EventType.NORMAL)
val ruleSet = List(
                    Rule(Rule.EQUAL, eventA.tag_id, eventB.tag_id),
                    Rule(Rule.EQUAL, eventA.tag_id, eventC.tag_id),
                    Rule(Rule.NOT, eventB))
val detectedEventStream =
myDataStream.pattern(eventA,eventB,eventC).where(ruleSet).within(Time.of(1,
TimeUnit.HOURS))
```

The components of the API consist of : - `Event` which make up the pattern that is to be detected. A pattern is generally made up of multiple events - `EventType` which specifies the type of `Event` that is detected. Types of events can be either `NORMAL`, `KLEENESTAR` or `KLEENEPLUS` - `Rule` defines each individual rule of the pattern query. They are basically comparison and logical operations. A `Rule` can be comprised of multiple `Rule` instances to satisfy a complex pattern query as follows

```
val mySpecialRule = Rule(Rule.OR, Rule(Rule.EQUAL, eventA.id, eventB.id),
Rule(Rule.EQUAL, eventA.price, eventB.price))
```

This API could be implemented using the existing Windowing API. A very simple translation example is as follows

```
// Using the Pattern Matching API
val myDataStream[DataStreamObject] = ...
val eventA = Event(DataStreamObject.name, "A", EventType.NORMAL)
val eventB = Event(DataStreamObject.name, "B", EventType.NORMAL)
val ruleSet = List(Rule(Rule.EQUAL, eventA.value, eventB.value))
val detectedEventStream =
myDataStream.pattern(eventA,eventB).where(ruleSet).within(Time.of(1,
TimeUnit.HOURS))
```

```
// The Windowing API translated implementation
val myDataStream[DataStreamObject] = ...
val detectedEventStream = myDataStream.window(Time.of(1,
TimeUnit.HOURS)).every(Count.of(2)).filter(EventFilterFunction).reduce(PriceEqualit
yDetectionReduceFunction)
```

The majority of the project will be focused on developing a clean and simple Pattern Matching API resembling that which is described above. Once a stable API is complete, we can think about trying to go ahead with implementing a few optimizations, such as a NFA based pattern evaluation model[1], sharing operators[3] across Flink's TaskManagers, and merging equivalent runs of the same pattern matching fork[1].

Timeline

- Before 25 May : Hash out details of the Pattern Matching API such as supported `EventTypes`, `Rules`etc. Also discuss on how the project will be taken forward, implementation details, communication channels, bi-weekly reviews of progress made, etc.
- 1st and 2nd week: Start implementing the `Event` and `Rule` parts of the API.
- 3rd and 4th week: Add operators that would be required by the API such as the `EventType.KLEENESTAR` along with unit tests and documentation.
- 5th week:(Buffer week: Time set aside to overcome unforeseeable blocks that hindered progress) Try to finish up the previous week's work. Clean up documentation and tests. Add a complex integration test that checks for correctness on all the implemented work.
- 6th week: Mid-term evaluations
- 7th and 8th week: Integrate the pattern matching API into the `flink-streaming` API for both Scala and Java
- 9th and 10th week: Work on a few optimization techniques discussed above to improve the performance of the pattern matching API
- 11th and 12th week: Start fixing remaining TODO's left across the codebase, cleaning up documentation. Get people to try it the pattern matching API out, and make changes according to

their feedback. Fix bugs as they pop up
- 13th week: Finish up documentation, complete writing tests, fix any remaining TODO's.

About Me

I'm a 4th year undergrad at VIT University, Vellore majoring in Computer Science and Engineering. I'm currently interested in distributed systems and large scale graph analytics. For my final semester, I'm interning at IISc Bangalore working on a subgraph-centric framework on large scale graph processing, GoFFish[2]. Although this is my first time applying to GSoC, I wanted to apply last year as well. But I managed to get a summer internship at Google, so I was deemed ineligible to apply.

I'm currently improving my scala skills as I try to solve s-99 here. I have contributed on and off to a few open source projects during my time as an undergrad. I've pushed code to Osquery, SimpleCV, openwayback, and most recently, Flink. To familiarise myself with the contributor workflow for Apache Flink, I sent a small documentation fix pull request . As a part of getting comfortable with the `flink-streaming` code base, I've worked on FLINK-1450 and FLINK-1655.

References

1. Agrawal, Jagrati, et al. "Efficient pattern matching over event streams." Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008.
2. Simmhan, Yogesh, et al. "Goffish: A sub-graph centric framework for large-scale graph analytics." Euro-Par 2014 Parallel Processing. Springer International Publishing, 2014. 451-462.
3. Nagy, Kristian A., "Distributed Complex Event Detection." MEng Individual project report 2012, http://www.doc.ic.ac.uk/teaching/distinguished-projects/2012/k.nagy.pdf

Regards,

Akshay Dixit