COMPLEX EVENT PROCESSING WITH APACHE FLINK

KOSTAS KLOUDAS

COMMITTER @ APACHE FLINK
SOFTWARE ENGINEER @ DATA ARTISANS
ABOUT DATA ARTISANS

Original creators of Apache Flink®

Open Source Apache Flink + dA Application Manager
WHAT IS CEP?
CEP: COMPLEX EVENT PROCESSING

- Detecting event patterns
- Over continuous streams of events
- Often arriving out-of-order
CEP: COMPLEX EVENT PROCESSING
CEP: COMPLEX EVENT PROCESSING
CEP: COMPLEX EVENT PROCESSING

Input

Pattern

Output

=
CEP: EXAMPLES

- **Security**: Raise alert when user a has X consecutive failed logins within Y secs, or he/she prints more than Z important documents within Y seconds.

- **Supply Chain Monitoring**: Detect all intermediate stops of all shipments that started from a contaminated site, or report all shipments which took in total more than Y hours.

- **Stocks trends**: detect stocks that start high, keep a high avg. price for a period, and then volume plummets.
CEP: USE-CASES

- IoT
- Fraud Detection
- Intrusion Detection
- Inventory Management
- Click Stream Analysis
- Trend Detection in financial sector
- ...yours?
WHAT IS STREAM PROCESSING?
STREAM PROCESSING

Computations on never-ending “streams” of events
DISTRIBUTED STREAM PROCESSING

Computation spread across many machines
STATEFUL STREAM PROCESSING

Result depends on history of stream
Stream Processors are a natural fit for CEP
CEP ON FLINK

Input

FlinkCEP

Pattern

Output

© 2018 data Artisans
What does FlinkCEP offer?
PATTERN DEFINITION
PATTERN DEFINITION

• Composed of Individual Patterns

\[ P_1 \text{ (shape == rectangle)} \]
\[ P_2 \text{ (shape == triangle)} \]
PATTERN DEFINITION

• Composed of Individual Patterns
  - $P_1$ (shape == rectangle)
  - $P_2$ (shape == triangle)

• Combined by Contiguity Conditions
  — ...later
FLINKCEP INDIVIDUAL PATTERNS

• Unique Name

• Condition : which elements to accept
  – Simple e.g shape == rectangle
  – Iterative e.g rectangle.surface < triangle.surface

• Quantifiers (or not)
  – Looping/Optional: oneOrMore(), times(#), times(from,to), optional(), until(cond.), greedy()
FLINKCEP COMPLEX PATTERNS

• Combine Individual Patterns
  – Contiguity Conditions: *how to select relevant events* given an input mixing relevant and irrelevant events
  – Matching Semantics: *where to start searching for a new match*
  – Time Constraints: *within(time)* e.g. all events have to come within 24h

• Combine Complex Patterns
  – Grouping Complex Patterns: the “parenthesis” of pattern definition.
FLINKCEP CONTIGUITY CONDITIONS

Pattern

Input
FLINKCEP CONTIGUITY CONDITIONS

**Strict Contiguity**

- matching events strictly follow each other
Pattern

FLINKCEP CONTIGUITY CONDITIONS

Input

Output
Relaxed Contiguity

- non-matching events to simply be ignored
FLINKCEP CONTIGUITY CONDITIONS

Pattern

Input

Output

- Pattern: square and triangle
- Input: green squares and triangles
- Output: red triangles, green squares, and yellow triangle
FLINKCEP CONTIGUITY CONDITIONS

Pattern

Input

Output
Non-Deterministic Relaxed Contiguity

• allows non-deterministic actions on relevant events
FLINKCEP CONTIGUITY CONDITIONS

NOT patterns:

- for strict and relaxed contiguity
- for cases where an event should invalidate a match
FLINKCEP

• Individual Patterns:
  – Quantifiers
  – Conditions: Simple & Iterative

• Complex Patterns:
  – Time Constraints: Event and Processing time
  – Contiguity Constraints: Strict, relaxed, non-deterministic, NOT
  – Matching Semantics: Where to start searching for a new match
  – Grouping Complex Patterns: The “parenthesis” of pattern definition.
What is coming in FlinkCEP?
ROADMAP

• Integration with SQL:
  – `match_recognize()` clause in SQL 2016
  – write SQL and detect CEP patterns on the result

• Dynamic pattern specification:
  – `add()`, `remove()` patterns to be evaluated against your stream
Example
RUNNING EXAMPLE: RETAILER

• Trace all shipments which:
  – start at location A
  – have at least 5 stops
  – end at location B
  – within the last 24h
Trace all shipments which:
- start at location A
- have at least 5 stops
- end at location B
- within the last 24h

OBSERVATION A INDIVIDUAL PATTERNS
OBSERVATION B QUANTIFIERS

• **Start/End**: single event

• **Middle**: multiple events
  - 5 or more
OBSERVATION C CONDITIONS

- **Start** -> Simple
  - properties of the event

- **Middle/End** -> Iterative
  - depend on previous events

\[
\text{ev}.\text{from} == \text{A} \\
\text{ev}.\text{to} == \text{B} \\
&&
\text{size(“mid”) >= 5}
\]
Trace all shipments which:
  - start at location A
  - have at least 5 stops
  - end at location B
  - within the last 24h
OBSERVATION E CONTIGUITY

• We opt for relaxed continuity
Pattern< Event, ? > pattern = Pattern
  .< Event > begin("start")
    . where (mySimpleCondition)
  . followedBy ("middle")
    . where (myIterativeCondition1)
    . oneOrMore()
  . followedBy ("end")
    . where (myIterativeCondition2)
  . within (Time.hours(24))
Pattern<Event, ?> pattern = Pattern
  .<Event>begin("start")
    .where(mySimpleCondition)
  .followedBy("middle")
    .where(mylIterativeCondition1)
    .oneOrMore()
  .followedBy("end")
    .where(mylIterativeCondition2)
  .within(Time.hours(24))

Start
Middle
End
Pattern<Event, ?> pattern = Pattern
  .<Event>begin("start")
    .where(mySimpleCondition)
  .followedBy("middle")
    .where(mylIterativeCondition1)
      .oneOrMore()
    .followedBy("end")
      .where(mylIterativeCondition2)
      .within(Time.hours(24))
Pattern\(<\text{Event}, \, ?>\) \text{pattern} \, = \, \text{Pattern}

\text{.\langle\text{Event}\rangle\, begin("start")}
\text{.\, where(mySimpleCondition)}
\text{.\, followedBy("middle")}
\text{.\, where(myIterativeCondition1)}
\text{.\, oneOrMore()}
\text{.\, followedBy("end")}
\text{.\, where(myIterativeCondition2)}
\text{.\, within(Time.hours(24))}
RUNNING EXAMPLE INTEGRATION

Pattern<Event, ?> pattern = ...

PatternStream<Event> patternStream = CEP.pattern(input, pattern);

DataStream<Alert> result = patternStream.select(
    new PatternSelectFunction<Event, Alert>() {
        @Override
        public Alert select(Map<String, List<Event>> pattern) {
            return parseMatch(pattern);
        }
    });
RUNNING EXAMPLE INTEGRATION

Pattern<Event, ?> pattern = ...

PatternStream<Event> patternStream = CEP.pattern(input, pattern);

DataStream<Alert> result = patternStream.select(
    new PatternSelectFunction<Event, Alert>() {
        @Override
        public Alert select(Map<String, List<Event>> pattern) {
            return parseMatch(pattern);
        }
    });

RUNNING EXAMPLE INTEGRATION

Pattern<Event, ?> pattern = ... 

PatternStream<Event> patternStream = CEP.pattern(input, pattern);

DataStream<Alert> result = patternStream.select (new PatternSelectFunction<Event, Alert>(){
  @Override
  public Alert select(Map<String, List<Event>> pattern) {
    return parseMatch(pattern);
  }
});
CALL FOR PRESENTATIONS IS OPEN

The Apache Flink® Conference
Stream Processing | Event Driven | Real Time

3 SEPTEMBER 2018: TRAINING
4-5 SEPTEMBER 2018: CONFERENCE
BERLIN, GERMANY

Submit your talk at berlin.flink-forward.org

#flinkforward
THANK YOU!

@kkloudas
@dataArtisans
@ApacheFlink

WE ARE HIRING

data-artisans.com/careers
POWERED BY APACHE FLINK