CS 237:
Event Framework for TIPPERS

Qiushi Bai, Avinash Kumar, Jonathan Harijanto

Motivation

- TIPPERS App: “Concierge”
  - Allows users to
    - Locate entities such as people, room, and events
    - Receive a notification when a room / a person is available in the building
  - For example, “Notify me when person X is in the building” OR “Looking for person X”
- However,
  - Only simple predicates defined
  - Duplicate coding for each function
  - Has a backend which does all the work of filtering data and finding the result

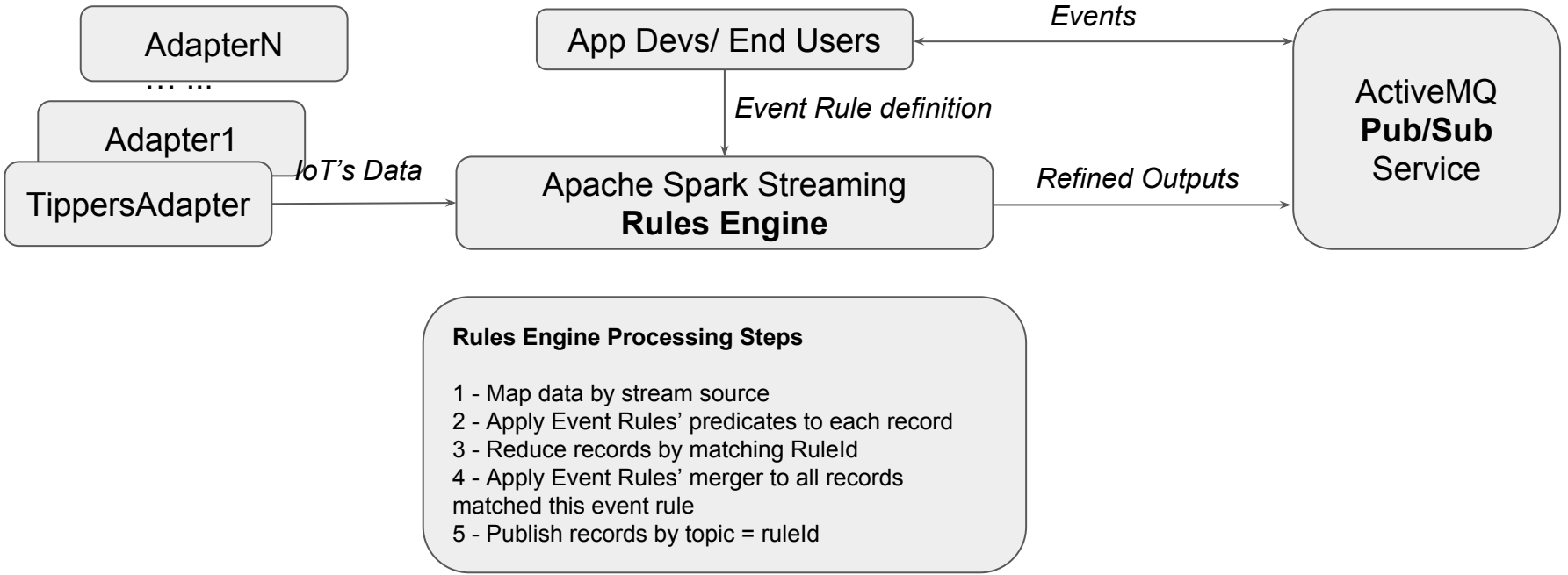
Goal

- Implementing an event framework for TIPPERS IoT data stream, which
  - Allow user define events with more complex predicates.
    - Person X and Person Y appear on the same floor, notify me.
    - Person X appears in the building and Room R is empty, notify me.
  - Very little coding for this applications - declare predicates and rules like using SQL

Related Work

- Research on Stream Processing softwares:
  - Apache Storm, Apache Samza, Apache Spark, Apache Flink
- Research on Rules Engine
- Research on Pub/Sub frameworks:
  - (Google) Cloud Pub/Sub
    - (+) Provides a “push subscription” feature.
    - (-) Charges a small amount of money per gigabyte once exceed the quota.
  - Apache Kafka, ActiveMQ
    - (+) Open source, so it is free
    - (-) Requires a perfectly running server

Prototype Architecture



Prototype Design Specification

- TippersAdapter
  - Pulls the IoT sensor data from TIPPERS' database in a real-time manner
    - Current IoT data:
      - Physical sensor type → WeMo, Thermometer, and WifiAP
      - Semantic sensor type → Presence and Occupancy
  - Pushes the IoT data to the rules engine through a socket.

```
OCCUPANCY|470c9f13-d107-49ba-ab1b-5c19ba0c6278|2100_1|36|2017-11-08 07:02:00|vSensor2
PRESENCE|dd30fe99-98eb-4b7a-87d1-aaa530c23113|2baf0a1f_6d3a_47bd_9137_05ea3692bda9|5100_4|2017-11-08 07:04:00|vSensor1
WiFiAPObservation|0d68ac60-94fe-40ac-baec-881943ea2c95|1d3477bc-4d1c-472c-a3fb-50f0d017c8d6|2017-11-08 07:04:00|3143_cl
ThermometerObservation|258cf974-0261-4c7d-a4e2-55e6286e84dc|24|2017-11-08 07:02:00|24c0a2bb_9dee_4d58_9f10_7df41ac63afa
ThermometerObservation|8bc88da7-7de1-4014-8976-9b017c7c9cd1|3|2017-11-08 07:02:00|70fe2e8f_4baf_4ef0_a133_205d302190eb
WeMoObservation|3aaf2909-0e83-4e4b-816d-8cfc9ae0e33a|23|484|2017-11-08 07:04:00|7503c550_a671_4599_a583_b1d6eefab4e8
```

Example of the IoT data from TIPPERS

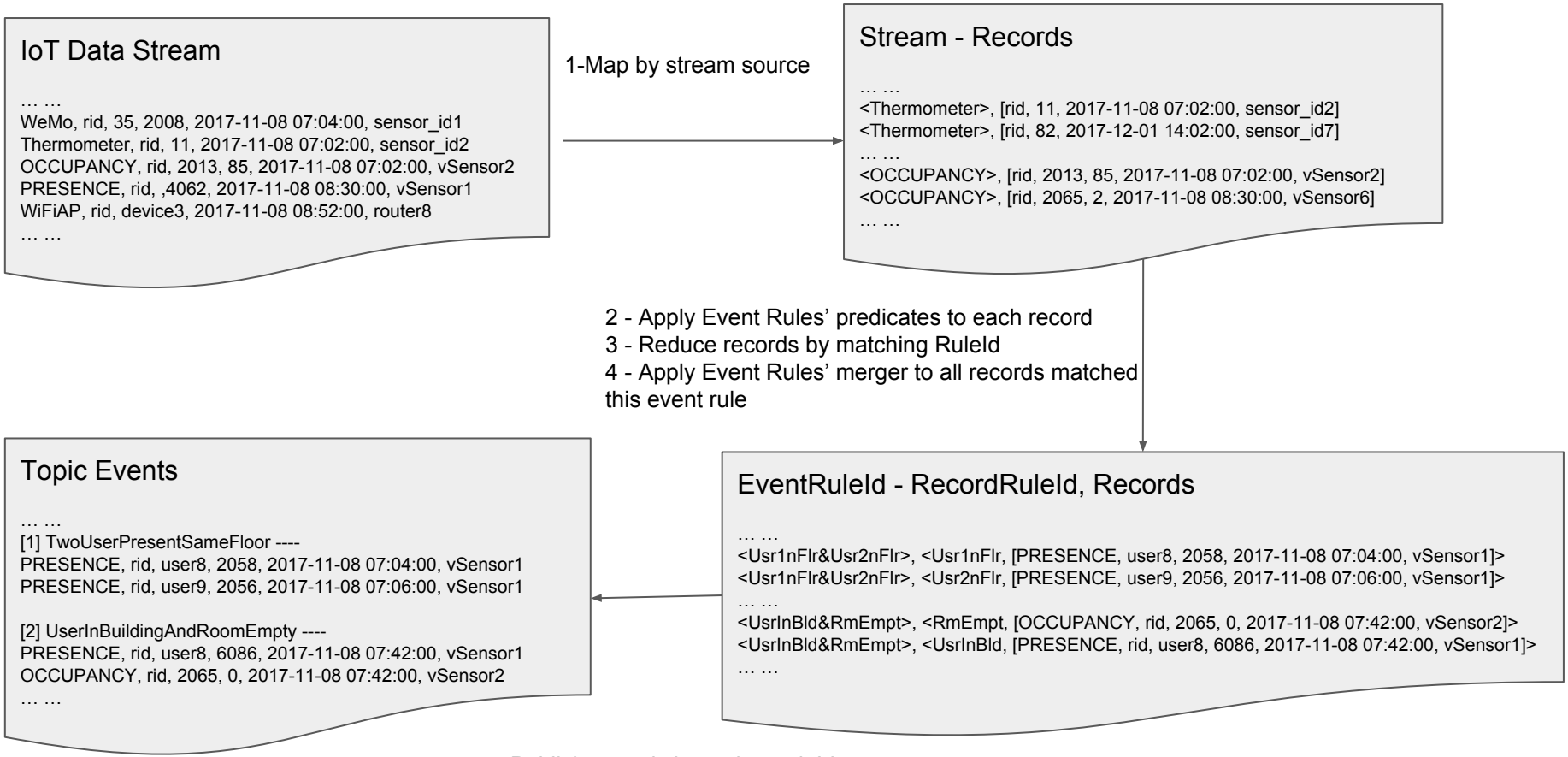
Prototype Design Specification

- Apache Spark Streaming rules engine
  - Processes the incoming IoT data on a socket in a real-time manner.
  - Loads event rules defined by user-defined functions (UDF)
  - Applies the event rules by filtering the IoT data
    - Supports combination of predicates among streams
      - For example, *presence.userId = "X"* AND *occupancy.location = "2065"* AND *occupancy.NumberOfPeople = 0*
      - Meaning: *Person X is in building* AND *Room 2065 is empty*
    - Supports conjunction predicates within a record
      - For example, *presence.userId = "X"* AND *presence.location like "20%"*
      - Meaning: "Person X present on 2nd floor"
  - Outputs a list of records that satisfy the rules

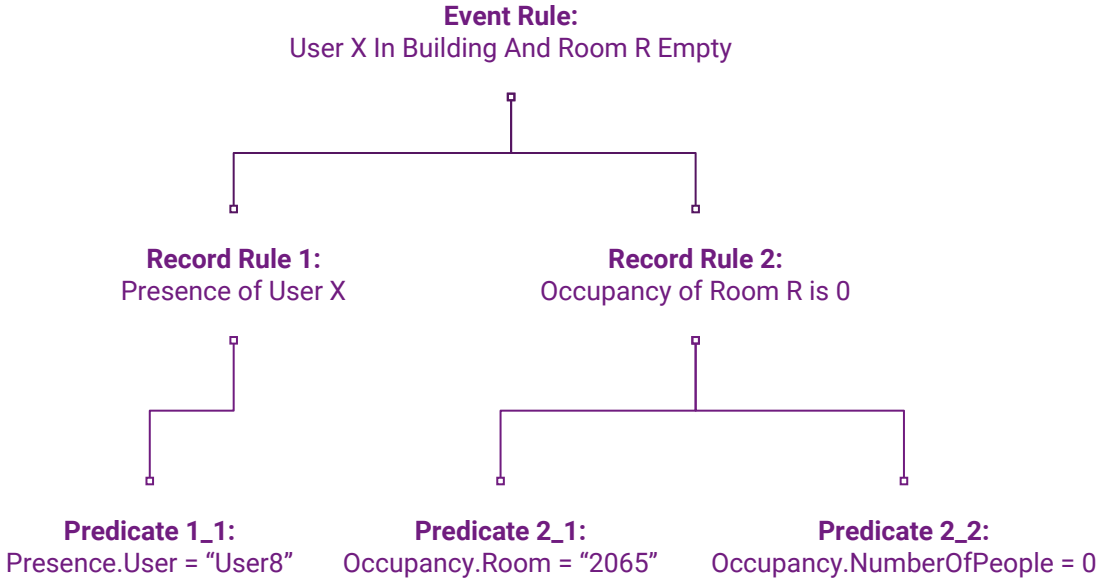
Prototype Design Specification

- Apache ActiveMQ
  - Receives a list of records that satisfies a specific rule from Spark
    - The rule becomes the topic
      - For example, “TwoUsersPresentSameFloor”
    - The records becomes the message(s)
      - PRESENCE,ID1,user8,2058,2017-11-08 07:04:00,vSensor1
      - PRESENCE,ID2,user9,2056,2017-11-08 07:06:00,vSensor1
  - A publisher of a topic pushes the message(s)
  - Subscribers of a topic pull the message(s)

Data Flow


IoT Data Stream
… …
WeMo, rid, 35, 2008, 2017-11-08 07:04:00, sensor_id1
Thermometer, rid, 11, 2017-11-08 07:02:00, sensor_id2
OCCUPANCY, rid, 2013, 85, 2017-11-08 07:02:00, vSensor2
PRESENCE, rid, ,4062, 2017-11-08 08:30:00, vSensor1
WiFiAP, rid, device3, 2017-11-08 08:52:00, router8
… …
1-Map by stream source
Stream - Records
… …
<Thermometer>, [rid, 11, 2017-11-08 07:02:00, sensor_id2]
<Thermometer>, [rid, 82, 2017-12-01 14:02:00, sensor_id7]
… …
<OCCUPANCY>, [rid, 2013, 85, 2017-11-08 07:02:00, vSensor2]
<OCCUPANCY>, [rid, 2065, 2, 2017-11-08 08:30:00, vSensor6]
… …
2 - Apply Event Rules' predicates to each record
3 - Reduce records by matching RuleId
4 - Apply Event Rules' merger to all records matched this event rule
EventRuleId - RecordRuleId, Records
… …
<Usr1nFlr&Usr2nFlr>, <Usr1nFlr, [PRESENCE, user8, 2058, 2017-11-08 07:04:00, vSensor1]>
<Usr1nFlr&Usr2nFlr>, <Usr2nFlr, [PRESENCE, user9, 2056, 2017-11-08 07:06:00, vSensor1]>
… …
<UsrInBld&RmEmpt>, <RmEmpt, [OCCUPANCY, rid, 2065, 0, 2017-11-08 07:42:00, vSensor2]>
<UsrInBld&RmEmpt>, <UsrInBld, [PRESENCE, rid, user8, 6086, 2017-11-08 07:42:00, vSensor1]>
… …
Topic Events
… …
[1] TwoUserPresentSameFloor ----
PRESENCE, rid, user8, 2058, 2017-11-08 07:04:00, vSensor1
PRESENCE, rid, user9, 2056, 2017-11-08 07:06:00, vSensor1
[2] UserInBuildingAndRoomEmpty ----
PRESENCE, rid, user8, 6086, 2017-11-08 07:42:00, vSensor1
OCCUPANCY, rid, 2065, 0, 2017-11-08 07:42:00, vSensor2
… …
5 - Publish records by topic = ruleId

UDF - User Defined Function




```java
@Override
String topicName() { return "UserInBuildingAndRoomEmpty"; }

@Override
List<IRecordRule> recordRuleList() {

    List<IRecordRule> recordRuleList = new ArrayList<>();

    RecordRule r_user_in_building = new RecordRule( parent: this);
    r_user_in_building.id = "r_user_in_building";
    r_user_in_building.stream = "PRESENCE";

    Predicate p_user = new Predicate(r_user_in_building);
    p_user.id = "p_user";
    p_user.attribute = "semantic_entity_id";
    p_user.attributeType = AttributeType.STRING;
    p_user.operator = Operators.EQUAL;
    p_user.valueString = this.user;

    r_user_in_building.predicateList.add(p_user);

    RecordRule r_room_empty = new RecordRule( parent: this);
    r_room_empty.id = "r_room_empty";
    r_room_empty.stream = "OCCUPANCY";

    Predicate p_room = new Predicate(r_room_empty);
    p_room.id = "p_room";
    p_room.attribute = "semantic_entity_id";
    p_room.attributeType = AttributeType.STRING;
    p_room.operator = Operators.EQUAL;
    p_room.valueString = this.room;

    Predicate p_empty = new Predicate(r_room_empty);
    p_empty.id = "p_empty";
    p_empty.attribute = "occupancy";
    p_empty.attributeType = AttributeType.INT;
    p_empty.operator = Operators.EQUAL;
    p_empty.valueInt = 0;

    r_room_empty.predicateList.add(p_room);
    r_room_empty.predicateList.add(p_empty);

    recordRuleList.add(r_user_in_building);
    recordRuleList.add(r_room_empty);

    return recordRuleList;
}
```

Challenges

- Abstraction - Declarative UDF
  - Supports conjunction predicates within record
    - Example - presence.userId = “X” AND presence.location like “20%”;
    - Meaning - Person X present on 2nd floor
  - Supports combination predicates among streams
    - Example - presence.userId = “X” AND occupancy.location = “2065” AND occupancy.NumberOfPeople = 0.
    - Meaning - Person X is in building AND Room 2065 is empty
  - UDF coding is little
- Scalability - Spark Stream
  - High frequency of input stream data
  - Support distributed environment

Challenges

- (Google) Cloud Pub/Sub is very complicated!
  - Authentication issue:
    - Cloud Pub/Sub API Client Library for Java
  - Runtime issue:
    - Google App Engine
  - Dependency issue:
    - Cloud Pub/Sub API Client Library brings a lot of conflicts
- Solution: new Pub/Sub framework called Apache ActiveMQ
  - Works exactly the same way the Cloud Pub/Sub, but easier to implement
    - No authentication, runtime, and dependency issues
  - Drawback → pull subscription based

Contributions

- Exploration on implementing Stream Event Framework for TIPPERS
- Spark + Apache ActiveMQ / Google Cloud Pub/Sub
- Semantically more complex Rule and Predicates flexibility
- UDF support

Thank you.