Low Cost Real Time Complex Event Processing And Stream Reasoning System

Laty Ndiaye Mouhamet, Mamour Diop Serigne, Yacine Ghamri-Doudane, Ismail Benis, Didier Orange, Magali Gerino

Low Cost Real Time Complex Event Processing And Stream Reasoning System

Mouhamet Latyr Ndiaye
L3i, La Rochelle University
La Rochelle, France
mouhamet.ndiaye@univ-lr.fr

El Hadji Serigne Mamour Diop
Gaston Berger University
Saint Louis, Senegal
serigne-mamour.diop@ugb.edu.sn

Yacine Ghamri-Doudane
L3i, La Rochelle University
La Rochelle, France
yacine.ghamri@univ-lr.fr

Ismail Benis
IRIMAS, University of Haute-Alsace
Mulhouse, France
ismail.bennis@uha.fr

Didier Orange
Université de Montpellier
Montpellier, France
didier.orange@ird.fr

Magali Gerino
LEFE/Campus Université Toulouse 3
Toulouse, France
magali.gerino@univ-tlse3.fr

***Abstract*—In IoT applications, data are continuously generated and must be processed in real-time to react to state changing in the environment where devices are deployed. Processing data as generated from sources is inefficient because of the low level of data. In this paper, we propose a generic Complex Event Processing system with reasoning abilities which we have evaluated with case studies.**

***Index Terms*—Complex Event Processing, Stream Reasoning, WaziUp**

I. Introduction

IoT applications are widely used in various domains and large amounts of data are continuously produced. These data are generated as streams and must be processed in real time because the system has to react almost instantaneously to their variations to make decisions, raise alerts or activate actuators. Processing these data in their raw form remains most of the time inefficient. We shoud then be able to have means of processing, not the raw data but a representation with an abstraction level that makes it more tangible and easier to manipulate. In addition, in most of the proposed architectures, the data processing is done in the Cloud.

To avoid drawbacks induced by cloud-based processing and to design a system that can be used in rural areas where internet is not always present, we propose in this paper a Real Time Stream Processing system. This system consists of a combination of *Semantic Web* tools and *Complex Event Processing (CEP)* technologies with a *Semantic Model* to perform reasoning on data streams (*Stream Reasoning*). The implementation is carried out on the *WaziGate* gateway offered by the *WAZIUP* project to ensure processing is performed as close as possible to the data sources. We have evaluated the proposal with a fire detection system and a biological wastewater quality control system.

II. RELATED WORKS

Reis et al. [1] present a semantic model to allow real-time reasoning over data streams by combining CEP and performing continuous over streams of RDF semantic data. Their approach allow deriving RDF data from basic events by using CEP in edge devices.

A. Dhillon et al. [2] propose a CEP based approach for Remote Patient Monitoring. They use a mobile device and a remote IoT Hospital Server (IHS) deployed on the cloud. In their work, complex event detection is performed on the edge and complex event streams are sent to the hospital server for further processing.

Lan et al [3] propose a CEP mechanism for Real-Time Monitoring using hierarchical complex event model to reduce the complexity of event modeling. The CEP system is deployed on the network edge between sensing devices in terminal and applications in the cloud.

However, these propositions do not address all the constrains we aim to lift, which are:

- ***Internet access is not always guaranteed*** In the rural areas of many localities, as in some African countries, Internet access is sometimes poor or even non-existent.
- ***Cost*** To minimize cost, the system should be deployable in low-cost, low-power devices such as Single Board Computers (SBCs).
- ***Genericity*** The system must be generic enough to adapt to different use cases.

III. Proposition

A. System operation

Considering the constraints, we listed in the previous section, we proposed a system that is attached on top of the module provided by the Waziup project [4]. Waziup is a collaborative research project for IoT and Big Data to improve working conditions in the rural ecosystem of sub-Saharan Africa. In our system, messages are sent from sensors and received at the gateway. After a preprocessing stage, data are passed to the stream reasoning system which is composed of three modules: Complex Event Processing module, Reasoning module and Controller module

The end-device consists of a programmable device equipped with sensors and a LoRa connectivity. It is responsible for collecting the values read by the sensors, sending them to the gateway and activating actuators. The gateway has LoRa connectivity to communicate with devices. It receives

data from sensors, performs analysis and sends commands to the end-device to control its state.

B. Architecture

In the proposed system, devices take measurements and send them to the gateway. This one performs operations with the CEP module and produces a result as *facts*, represented by RDF statements. These *facts* are inserted into a reasoner which contains queries to deduce relevant knowledge for the system. The reasoner combines the declarations received from the CEP module with a knowledge base to deduce complex situations defined within the reasoner.

1) CEP module: Data coming from the devices are injected into a CEP engine which contains the rules allowing to extract the useful events. Events undergo a number of transformations such as aggregation, filtering etc. Each unit takes a stream as input and outputs a stream of events satisfying the criteria defined by the processing unit. By making the correspondence between these events and the facts they describe, the module outputs a stream of RDF statements corresponding to these events.

2) Reasoner module: Reasoning consists of deducing new facts from existing axioms and relationships in an ontology. It is used to describe the concepts and entities of a domain as well as the relationships that exist between them. The reasoning module relates the data contained in the knowledge base to the statements resulting from the CEP stage in order to find expected facts.

3) Controller Module: This module is responsible for sending commands to devices in order to act on the monitored environment. These commands correspond to the results obtained from the reasoning module.

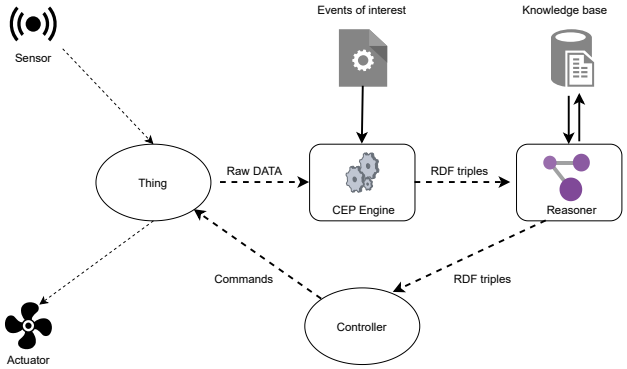

Fig. 1. Global process diagram

C. Implementation

In this section, we will propose an implementation of our architecture.

1) The hardware: The sensing device consists of a microcontroller to which various sensors and/or actuators are connected.

The Gateway is based on a 3 B+ Raspberry Pi running the Raspbian operating system, a Debian-based Linux distribution (Fig. 2).

Fig. 2. WaziUP's WaziGate

For long range communication, the gateway and the sensing devices are equipped with LoRa modules which ensure bidirectional communication. We used the Modtronix inAir9 that operates at 868 and 915Mhz using a *Semtech SX1276* chip.

2) CEP implementation: The CEP module is implemented using Apache Flink which is a framework for processing unlimited data streams in real time. It provides a set of tools for performing operations over data streams. In addition, it has a very rich CEP library and can process a high data rate. Apache Flink offers multiple features such as:

- *Windowing:* Windows split the stream into parts of finite size, over which we can apply computations.
- *Filtering:* Filtering allows defining criteria on the data flow. Apache Flink has a large number of built-in filters and also allows users to define custom filter functions.
- *Aggregation:* Aggregation functions are used to reduce a group of values to a single value. Apache Flink offers many aggregation functions (SUM, COUNT...) and also allows user-defined functions.
- *Pattern Matching:* Pattern Matching allows setting conditions and sequences on data. Apache Flink provides a CEP library, *FlinkCEP*, which allows detecting complex patterns in an endless data stream. CEP rules are defined by setting conditions on the values, their succession etc., and also allow defining time windows (`TimeWindow`) or number of elements (`CountWindow`) to determine the interval of data to which the rules are applied. By mapping patterns to facts they describe, we generate an output stream of RDF statements of those facts. This will allow us to perform reasoning on the data stream.

3) Reasoning implementation: The implementation of this module is essentially based on OWLAPI [5] with the HermiT [6] reasoner. OWLAPI allows creation and manipulation of ontologies. It is an open source JAVA API that provides a large number of tools for handling ontologies. First, a knowledge base made up of A-Box and T-Box of the semantic

model is loaded. The T-Box contains concepts and roles definition while the A-Box describes individuals with assertions using the concepts and roles in the T-Box. The RDF stream produced from CEP rules is received and the RDF triples are combined with the statements of the knowledge base to deduce the expected facts. In this work, the HermiT reasoner is used. HermiT supports a wide range of optimizations that improve its reasoning performance on ontologies. It has some advantages over other popular reasoners such as Fact++ and Pellet. A comparison with Fact++ and Pellet is given in [6].

4) Controller module implementation: This module is responsible for sending commands to sensing devices. The commands sent depend to the results obtained by the reasoning stage and are sent as a string specifying the actuator and the action it should perform. Example: `/@AON/BOFF/C12#` In this command, actions represented by the characters ON, OFF and 12 are sent respectively to A, B and C actuators. In the next section the proposal is evaluated through case studies.

IV. Evaluation

In order to evaluate the system, two case studies were considered: a fire detectionsystem and a wastewater control system.

A. Case study: fire detection

In this case study, we will first perform the detection only with the temperature parameter and, in the second case, temperature and humidity will be used.

1) Case 1: fire detection using temperature: Each room of the building has a temperature sensor which continuously sends the temperature value to the gateway. We define a simple CEP rule which monitors the temperature and generates the RDF triple `(ROOM_ID, hasTemperature, HIGH_TEMPERATURE)` when the latter reaches a fixed threshold value. In the knowledge base, it is defined that a room with `HIGH_TEMPERATURE` value is considered as a room on fire.

CEP rule: With *Apache Flink* we define the rules allowing to detect rooms having a very strong heat. When an event satisfies this rule, the following triple is generated and directed to the output flow (*sink*): `(ROOM_ID, has, VeryHighHeat)`.

Reasoner The ontology consists of a TBox containing an individual of class *Room* and another of class *Temperature*. The relation `(ROOM_ID, hasTemperature,TEMPERATURE)` indicates that the room having the ID `ROOM_ID` has a temperature which value is `TEMPERATURE`. `TEMPERATURE` can take the values `LOW_TEMPERATURE`, `MEDIUM_TEMPERATURE` or `HIGH_TEMPERATURE`

The Reasoner module receives the previous triples and loads the knowledge base. For each triple, it queries the reasoner to find the relations describing a room on fire.

2) Case 2: Fire detection using temperature and humidity: In Case 2, the devices are equipped with a temperature and humidity sensor (DHT11).

CEP rule CEP rules are defined to detect rooms with heat that exceeds the `TMP_THRESHOLD` value and humidity below the `HU_THRESHOLD` value.

Reasoner The Case 1 ontology is completed by the individual `LOW_HUMIDITY` of class `Humidity`.

Here, we ask the reasoner to find the rooms that are on fire by defining its reasoning process.

3) Performance Analysis: We analyse here the performances of the system for the two cases defined in this case study. This analysis is mainly focused on latency and RAM and CPU resources usage.

Data availability Temperature and humidity are measured with an LM35(case 1) or DHT11(case 2) sensor then transmitted to the gateway by *LoRa*. Data are sent as `TC/23` for the first case and `TC/23/HU/50` for the second case. Here we measure the time elapsed between the receiving of messages by the gateway's LoRa module and the availability of raw data. Fig. 3 illustrates the evolution of the availability time for the two cases over a period of 60 minutes.

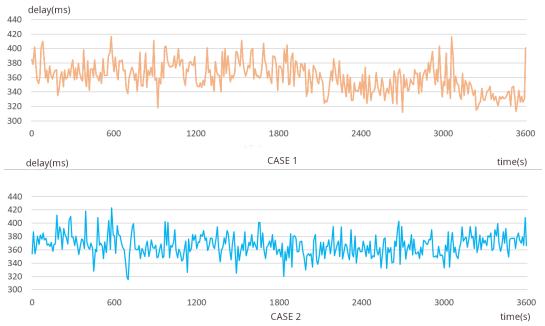

Fig. 3. Availability delays

Complex Event Processing The CEP processing time is the time spent by the data in the CEP module, i.e. from the admission of the data in this module to the creation of the RDF facts that result from the CEP analysis. The evolution of the CEP duration is represented in Fig. 4 for a period of 60 minutes.

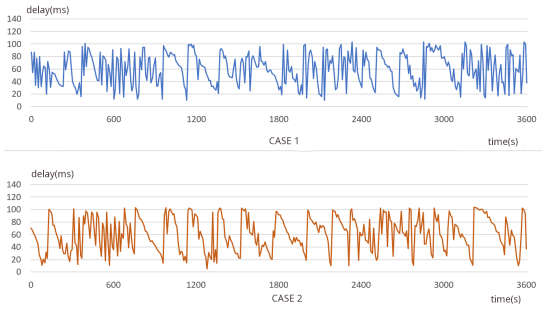

Fig. 4. CEP delays

Reasoning This stage covers from the receiving of data by the reasoner to the generation of new facts as a result of the reasoning process. Figure 5 gives reasoning times measured over a period of 60 minutes.

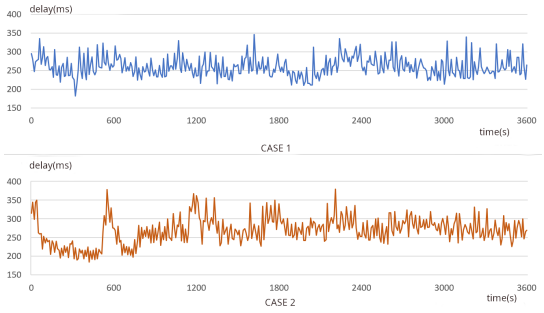

Fig. 5. Reasoning delays

4) Observations:

- **Case 1** The availability delay varies from 312ms to 417ms with an average value of 360.6ms. For the CEP stage, we have a variation in processing times from 10ms to 103ms with an average of 59.60ms. In the reasoning phase, the average execution time is 260.0ms with values varying from 182ms to 346ms.
- **Case 2** In case 2, the availability delay varies from 315ms to 423ms with an average value of 366ms. For the CEP stage, we have the processing times varying from 10ms to 104ms with an average of 61ms. In the reasoning phase, the average execution time is 268.9ms with values varying from 185ms to 379ms.

Analyzes The evaluation shows relatively average delays during the three program execution phases. There is a very slight increase in the total processing time when the number of measurements sent to the gateway doubles. The average CPU and RAM occupation are respectively 18% and 260MB, before program execution.
During execution, the average RAM usage increases from 260MB(28%) to 456MB(49%) i.e. 206MB(22%) of RAM used. CPU usage rose from 18% to 21%.

These performances can be satisfactory in different IoT applications (home automation, smart agriculture, etc.) but this case study may be too simple to evaluate the system. We will, in the next section, study a slightly more complex use case.

B. Case study: filters planted with reeds

This second case study concerns the wastewater treatment unit of Gaston Berger University. This station consists of filters planted with reeds for the treatment and reuse of wastewater. Filters planted with reeds are purification systems allowing monitored reconstitution of natural self-purification phenomena [7].

1) Description of the system: The system analyzes data from sensors installed at water filters and makes decisions based on this data. The objective is to be able to dynamically determine the possible uses of filtered water according to its quality. For this, a set of parameters is involved: pH, Suspended Solids(SS), Biological Oxygen Demand (BOD_5), Chemical Oxygen Demand (COD), Total Nitrogen(TN), Phosphorus and Colony forming unit(CFU). Three possible qualities have been defined for the filtered water: `GOOD`, `MEDIUM` and `BAD`.

The filtered water is used to irrigate three types of crops with different tolerances to wastewater constituents according to the quality of the filtered water by controlling the corresponding valves. Because of the unavailability of some sensors, the study was based on data from laboratory analyzed samples in [8]. These data were used to simulate sensors.

CEP rules for water filtering With *Apache Flink*, we have set the rules to filter relevant events. This step outputs an event as an RDF declaration. The produced event is of the form `(filter, has_quality,QUALITY)` with `QUALITY` $\in$ `{BAD, MEDIUM, GOOD}`.

Knowledge base The knowledge base comprises the resources of the system as well as the relationships between them. Fig. 6 represents an ontology describing the elements of a simplified filter.

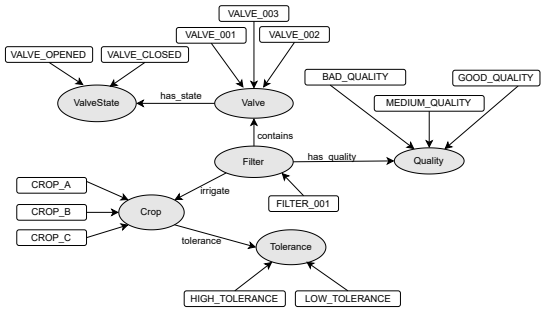

Fig. 6. Knowledge base ontology describing the filter's elements

The filter allows to irrigate crops with different tolerances, depending on the quality of the water it produces, by opening the corresponding valve. Irrigation is done according to the following rules:

- Water classified as *BAD* is discharged into the system.
- Water classified as *MEDIUM* can only irrigate crops with a high tolerance.
- Water classified as *GOOD* can irrigate all crops.

Tables I and II illustrate the correspondences between *Crop*, *Tolerance*, *Valve* and *Quality* classes.

Crop	CROP_A	CROP_B	CROP_C
Tolerance	HIGH_TOLERANCE	HIGH_TOLERANCE	LOW_TOLERANCE
Valve	VALVE_001	VALVE_002	VALVE_003

TABLE I
DIFFERENT CROP TOLERANCES AND CORRESPONDING VALVES

Tolerance	HIGH_TOLERANCE	LOW_TOLERANCE
Quality	GOOD - MEDIUM	GOOD

TABLE II
CORRESPONDENCE BETWEEN CROP TOLERANCE AND WATER QUALITY

Reasoning The reasoning process leads to determining which crops can be irrigated: `(Filter, irrigates,Crop)`. For this, it uses the information obtained from the CEP stage on the quality of the water: `(Filter, has_quality,Quality)`. This

information is used in combination with knowledge base assertions to determine which valves to open. At the end of the reasoning stage, the system produces the triple: `(Valve,has_state, ValveState)`. Using this information, the controller orders the filter to open the valves that serve the crops to be irrigated.

2) Performance Analysis: We measure here the performances of the system with respect to the execution durations of the different phases as well as the system resources (RAM and CPU) usage. The data for the simulation is randomly generated and submitted to the system at regular time intervals.

Processing delays

Data availability delay Here we measure the time elapsed between the reception of data and their availability for the CEP module. The evolution of this delay is given by graph 7 over a period of 60 minutes.

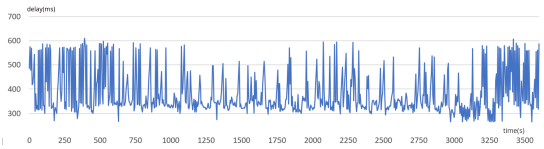

Fig. 7. Data availability delay

Complex Event Processing delay

The duration measured here is the time taken by the pattern detection process at the CEP level, i.e. from the reception of the data in this stage to the creation of the RDF facts resulting from the CEP analysis. These durations are represented in graph 8 for a period of 60 minutes.

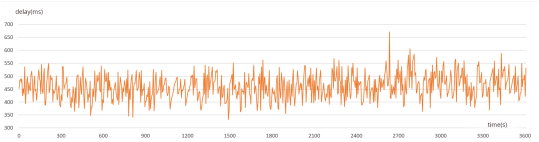

Fig. 8. CEP delays

Reasoning delay Here we measure the time taken by the reasoning process. This time corresponds to the duration which elapses between the generation of the RDF triples by the CEP module and the generation of the triples corresponding to the decision taken. Fig. 9 shows the reasoning delays measured over a 60-minute period.

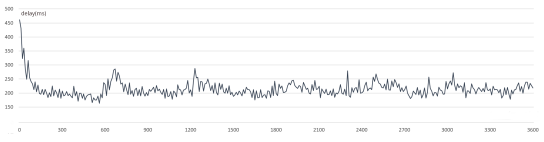

Fig. 9. Reasoning delays

These data show fairly short delays during the different processing phases. The availability delay varies from 265ms to 609ms with an average value of 382.32ms. For the CEP phase, we have a variation in processing times from 333ms to 671ms with an average of 460.60ms. In the reasoning phase, the average execution time is 214.30ms with values varying from 163ms to 461ms.

Resource usage We evaluate here the RAM and CPU usage during the execution of the program with data received every 10 seconds.

Before execution, CPU usage hovers around 20%. The memory occupation stagnates at around 213MB i.e. 23% of the total memory (927.2MB). During execution, the average CPU usage is around 25% with peaks up to 43%. The used memory climbs to 431MB(46.5%), an increase of 218MB(23.5% of total memory). By reducing the data reception interval to one second, there is a significant increase in CPU usage (from 20% before execution to 40% during execution) and RAM (from 213MB before execution to 500MB during execution).

These results show satisfactory processing times as well as an acceptable use of system resources for the considered use case.

V. Conclusion

We proposed a system to identify complex events from raw data from IoT devices in order to execute reasoning processes on these events. Validation tests were thus carried out through case studies in order to test the capability of the system to adapt to various scenarios, on a resource-constrained device. However, defining CEP rules manually can be challenging. Artificial Intelligence techniques should be able to generate business rules automatically or even to predict the occurrence or not of an event.

Acknowledgement Thanks to *SmartCleanGarden concept*, *Prix Convergences du Forum Mondial « Zero Pauvreté, Zero Exclusion, Zero Carbone » (2018-2019)*,*GDRI IRD-SENSE-South (2018-2022)*. https://smartcleangarden.org/

References

[1] R. D. Reis, M. Endler, V. P. de Almeida, and E. H. Haeusler, "A soft real-time stream reasoning service for the internet of things," in *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, 2019, pp. 166–169.

[2] A. S. Dhillon, S. Majumdar, M. St-Hilaire, and A. El-Haraki, "A mobile complex event processing system for remote patient monitoring," in *2018 IEEE International Congress on Internet of Things (ICIOT)*, 2018, pp. 180–183.

[3] L. Lan, R. Shi, B. Wang, L. Zhang, and N. Jiang, "A universal complex event processing mechanism based on edge computing for internet of things real-time monitoring," *IEEE Access*, vol. 7, pp. 101 865–101 878, 2019.

[4] C. Pham, A. Rahim, and P. Cousin, "Waziup: A low-cost infrastructure for deploying iot in developing countries," in *e-Infrastructure and e-Services for Developing Countries: 8th International Conference, AFRICOMM 2016, Ouagadougou, Burkina Faso, December 6-7, 2016, Proceedings 8*. Springer, 2018, pp. 135–144.

[5] M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies," *Semantic web*, vol. 2, no. 1, pp. 11–21, 2011.

[6] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "Hermit: an owl 2 reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.

[7] O. Gilibert, M. Gerino, D.-T. Costa, S. Sauvage, F. Julien, Y. Capowiez, and D. Orange, "Density effect of eisenia sp. epigeic earthworms on the hydraulic conductivity of sand filters for wastewater treatment," *Water*, vol. 14, no. 7, p. 1048, 2022.

[8] M. TOURÉ, "Etude des performances des filtres plantés de roseaux pour le traitement et la réutilisation des eaux usées de l'ugb," 2019.