Marrying Big Data with Smart Data in Sensor Stream Processing

Paula-Georgiana Zalhan
Babeş-Bolyai University of Cluj-Napoca
Cluj-Napoca, Romania *paula.zalhan@econ.ubbcluj.ro*

Gheorghe Cosmin Silaghi
Babeş-Bolyai University of Cluj-Napoca
Cluj-Napoca, Romania *gheorghe.silaghi@econ.ubbcluj.ro*

Robert Andrei Buchmann
Babeş-Bolyai University of Cluj-Napoca
Cluj-Napoca, Romania *robert.buchmann@econ.ubbcluj.ro*

Abstract

Widespread deployments of spatially distributed sensors are continuously generating data that require advanced analytical processing and interpretation by machines. Devising machine-interpretable descriptions of sensor data is a key issue in building a semantic stream processing engine. This paper proposes a semantic sensor stream processing pipeline using Apache Kafka to publish and subscribe semantic data streams in a scalable way. We use the Kafka Consumer API to annotate the sensor data using the Semantic Sensor Network ontology, then store the annotated output in an RDF triplestore for further reasoning or semantic integration with legacy information systems. We follow a Design Science approach addressing a Smart Airport scenario with geolocated audio sensors to evaluate the viability of the proposed pipeline under various Kafka-based configurations. Our experimental evaluations show that the multi-broker Kafka cluster setup supports read scalability thus facilitating the parallelization of the semantic enrichment of the sensor data.

Keywords: Semantic Stream Processing, Sensor data, Apache Kafka, Semantic Sensor Network ontology.

1. Introduction

Although Big Data has been the dominant buzzword in recent years, its research streams are gradually converging with those focusing on data quality and semantic enrichment, typically relying on graph databases with reasoning support - also marketed as "Smart Data" to suggest complementarity to "Big Data". Indeed, numerous events during 2018 branded this year as "the Year of the Graph" [28, 42] while white papers such as Bloor reports regard graph databases as the "fastest growing sector in the database market" [21].

This technological hype is not limited to rethinking traditional data models; it also drives the concept of a "semantic layer" over Big Data and enterprise information [8]. This raises a key requirement for Information Systems development to marry *quantity-driven* with *quality-driven* techniques in streamlined architectures. Sensor stream processing provides relevant application cases for this requirement and is under the scrutiny of our work. This paper is part of a larger effort addressing research challenges that derive from this convergence, approached through the methodological lens of Design Science [43].

The motivating design problem context is to support a Smart Airport with automatic speech and sound recognition - i.e., to detect suspicious sonorous manifestations with the help of geolocated audio sensors distributed across the airport premises. We are currently focusing in setting up the architectural core that streamlines the sensor data collection, semantic annotation and reasoning with the help of a tool pipeline that includes: the Apache Kafka distributed streaming platform [2], the GraphDB semantic database server [20] and the Semantic Sensor Network (SSN) ontology [39]. We position our work in the larger

paradigm of edge computing, as we are trying to assess the viability of the semantic sensor stream processing pipeline.

One key resource of this work is the SSN ontology (plus auxiliary terms from other vocabularies) - we employ it to annotate sensor data input in order to further subject it to linking and semantic processing (e.g., semantic queries, reasoning and integration). Knowledge representation frameworks such as RDF [36] facilitate a semantic layer that can help with the sensor-driven automation of decision making and incident management. Our goal is to facilitate the sensor data analysis by providing a novel semantic stream processing pipeline which includes data collection, semantic annotation, RDF data storage and query processing. This implies more granular tasks such as:

- To collect sensor data that comes from heterogeneous and dynamic sources;
- To distribute the processing of incoming data using parallel processing;
- To annotate the sensor data stream using the SSN ontology in order to enable the detection of certain patterns via reasoning and to achieve semantic interoperability with information systems that rely on the processed streams.

The structure of the paper is as follows: Section 2 comments on previous works related to semantic processing of data streams. In Section 3 we describe the used methodology. Section 4 presents the proposed semantic stream processing pipeline. Section 5 presents a use case scenario for the proposed system. In Section 6 we discuss experimental results regarding the performance of the system. Section 7 concludes the paper and provides an outlook to future work.

2. Related works

The data streams generated in Internet of Things (IoT) environments introduced several challenges related to their heterogeneous and highly dynamic nature. This opened a new research trend in the Semantic Web community, called RDF Stream Processing (RSP) or Linked Data Stream Processing [25], dealing with dynamically changing data that can be modelled by means of the RDF model. In the last decade, RSP engines have been built to model data streams using RDF and to apply continuous SPARQL query processing over the resulted RDF streams. Centralized RSP systems like C-SPARQL [7], CQELS [24], and $\text{SPARQL}_{\text{stream}}$ [12] allow querying RDF streams using extensions of SPARQL [41]. Due to the fact that these RSP engines provide different semantics, efforts are being spent towards a unifying and comprehensive query model that generalizes solutions such as C-SPARQL and CQELS. A unifying query model is proposed in [15] that formally defines the semantics of a RSP system using a SPARQL-extended query language called RSP-QL. However, these RSP systems are not capable of handling massive amounts of data streams, as they do not benefit from task parallelism and the scalability offered by a cluster computing infrastructure. To remedy these limitations and improve the performance of existing RSP systems, distributed RDF streaming systems were designed to enable concurrent queries over the incoming data. For example, the CQELS Cloud system [26] uses Apache Storm [5]; Strider [37] uses Apache Spark [4] to parallelize the continuous execution of queries over RDF data streams in the Cloud.

Several middleware solutions were proposed to transform unstructured streaming data into RDF streams reusing the Semantic Web tools stack. For example, the DataTurbine engine introduced in [18] is a streaming data middleware delivering data from sensors to the Data Center for later analysis following publish-subscribe model. In [27] a Linked Stream Middleware (LSM) platform transforms raw sensory data into RDF streams using W3C's Semantic Sensor Networks Incubator Group (SSN-XG) [40] ontology. The LSM system uses the RabbitMQ [34] publish-subscribe messaging platform as Message Bus and Virtuoso [32] as triple storage. Another middleware solution called Ztreamy [17] has been developed for large scale publishing of semantically annotated data streams on the Web. A recent framework called SEASOR [30] includes features from both centralized and distributed RSP engines providing semantic annotation of the summarized sensor data streams using the SSN ontology.

Our proposed solution for semantic processing of sensor data uses another distributed

messaging system called Apache Kafka, because it has better throughput, built-in partitioning for parallel data consumption than most messaging system have, which makes it suitable to build low-latency processing pipelines. Another aspect that differentiates our solution from other existing middleware solutions is the combined approach for semantic annotation mixing the SSN ontology with other vocabularies (e.g., Schema.org [38]). Moreover, the resulting sensor data graph is semantically integrated with the legacy database to support the development of a Hybrid Semantic System for Incident Management.

Stream Reasoning [16] started to take off in the research community in order to extend traditional stream processing engines with logical, rule-based capabilities. For example, LARS framework was proposed by [8] to formally express and analyze rich stream reasoning primitives under Answer Set Programming foundations. Another system called Streaming MASSIF [10] that uses Cascade Reasoning approach was introduced to perform expressive reasoning and complex event processing over large amounts of heterogeneous IoT data. Also, a similar approach based on stream reasoning models and techniques to process semantically-enriched data streams for supporting decision making in a Smart City was discussed in [14]. A similar effort is highlighted in [13] where authors focus on supervised stream learning from semantics of live traffic data using Description Logic reasoning.

3. Methodology

We are following the iterative treatment development cycle of Design Science, currently focusing on the core mechanisms and architecture that, at the stage hereby reported (Technological Readiness Level of 3), is deployed under laboratory conditions – therefore reported experiments will focus on relative system performance of the core pipeline.

The motivating application case raises a requirement to automate reasoning upon sensor data collected from a smart airport, enabled by a semantic layer that integrates sensor descriptions over the legacy Information System of the airport. Benefits can include prevention of critical events, a more efficient management of crowds and responsiveness to incidents. We aim to generalize this problem to a methodology and architecture for deploying semantic edge computing in problems specific to the management of natural disasters emergency interventions. The current paper focuses on the distributed semantic annotation pipeline that will become the foundation for the reasoning and decision-making components. Some early stage reasoning use cases will also be suggested.

4. Solution Overview: a Semantic Stream Processing Pipeline

Implementing an effective semantic stream processing pipeline architecture requires to address several aspects including data generation, stream processing, data storage and analysis. The main components of the proposed Semantic Stream Processing (SSP) pipeline are presented in Figure 1. This pipeline is based on Apache Kafka to collect and process the streaming data, GraphDB to store the annotated data streams and the SPARQL query language to analyze the resulting graph. In the remainder of this section, we briefly explain each of these components.

Data streams can be obtained from various sensor sources such as temperature, traffic, and location sensors. We focus on geolocated audio sensors to support a Smart Airport scenario (the future works section will also suggest a generalization opportunity reflected in the annotation schema).

The continuous sensor data gathered from heterogeneous data sources is collected and processed by a distributed data ingestion system for later semantic integration. Multiple tools can be used as data ingestion systems in a stream processing system: Apache Kafka, Apache Nifi [3], and Apache Flume [1]. In the current project, Apache Kafka is employed due to its characteristics that make it suitable to handle large-scale data - the biggest benefits are the ability to scale the load as data is ingested into the system and the replication mechanism guarding against data loss during system failures [29]. Kafka runs

as a cluster which connects multiple message producers and message consumers to one or more servers, called brokers. Internally, Kafka uses Apache Zookeeper [6] to store metadata about the Kafka cluster, such as information about topics, brokers and consumers.

The overall distribution mechanism is based on the publisher-subscriber pattern offered by Kafka core APIs. In our proposed SSP pipeline, the incoming data is collected and published into "topics" using the Kafka Producer API. To provide machine-readable and machine-interpretable descriptions of the ingested data, the Kafka Consumer API subscribes to the existing topics and annotates the stream of records using semantic technology, with a preferred frequency. The corresponding stream of records is turned into Smart Data annotated with the SSN ontology. This ontology focuses on describing physical sensor networks, such as sensors, observations that result from sensing, and deployments in which sensors are used. Key concepts are sensor, observation, actuation and sampling, concepts that were adopted from Sensor, Observation, Sample, and Actuator (SOSA) ontology [22]. As a combination of all precursor sensor ontologies, SSN becomes a de facto standard in semantic modelling of sensor data, information related to sensor capabilities and sensor deployment configurations. The data stream values enriched by semantics are persisted into a semantic graph database called GraphDB for reasoning, later analysis or integration with a legacy information system (e.g., a notification system).

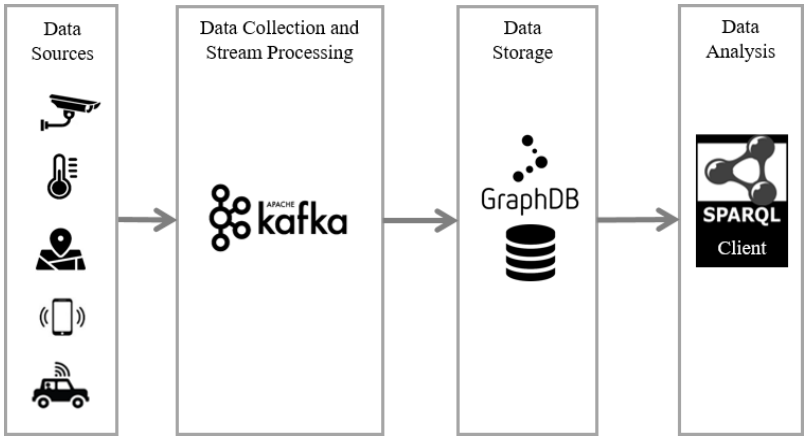

Fig. 1. Proposed semantic sensor stream processing pipeline

5. Scenario Setup and Design Decisions

The motivating context of our design problem is a Smart Airport infrastructure which includes geolocated audio sensors connected to automatic speech recognition (ASR) technology for the purpose of monitoring suspicious conversations of passengers to alert security teams and invoke rapid actions in case critical patterns are detected. The speech recognition component (extracting prominent word sequences uttered in a crowd) is out of scope for this paper, as we focus on the semantic integration architecture and parallelization of the semantic annotation effort. We use previous project [44] experience regarding the building of an ASR system and the development of such a component is not in the scope in this paper (available voice services are being investigated).

To simulate the data streaming from the geolocated audio sensors deployed in a smart airport, we used Producer and Consumer APIs that support custom implementations to write and read streams of data in the Kafka cluster. We have created producer tasks that send JSON messages to the Kafka cluster, published into the *AudioSpeech* topic which contains data-streams from the audio sensor. The stream of records from this topic has the following core schema:

- sensor_id: UUID,
- sensor_type: String in audio sensor,
- station_no: int,
- event_value: String,
- event_time: Timestamp.

The sensor_id field represents the Universal Unique Identifier (UUID) to uniquely identify the deployed audio sensors from the smart airport ecosystem. Raspberry Pi stations identified by a specific station_no are deployed at each airport floor and host several audio sensors. The event_value field of the AudioSpeech topic contains information related to the sound level of the uttered sentence of a specific passenger and the output of the ASR system. The stream of records from the AudioSpeech topic are published with timestamps marking the time when the acoustic data captured by the microphone has been processed and transformed into sentences by the ASR system.

To process the produced stream of records that were previously published, we have created consumer tasks that subscribe to the AudioSpeech topic, read the published stream record, annotate the raw sensor data from the stream of records using a schema derived from the SSN ontology and, lastly, store the resulting RDF descriptions into the semantic database. With SSN, we provide descriptions regarding to individual sensing devices, the relationship with their corresponding platform, their observation values and implied procedures, features of interest, and properties that were observed.

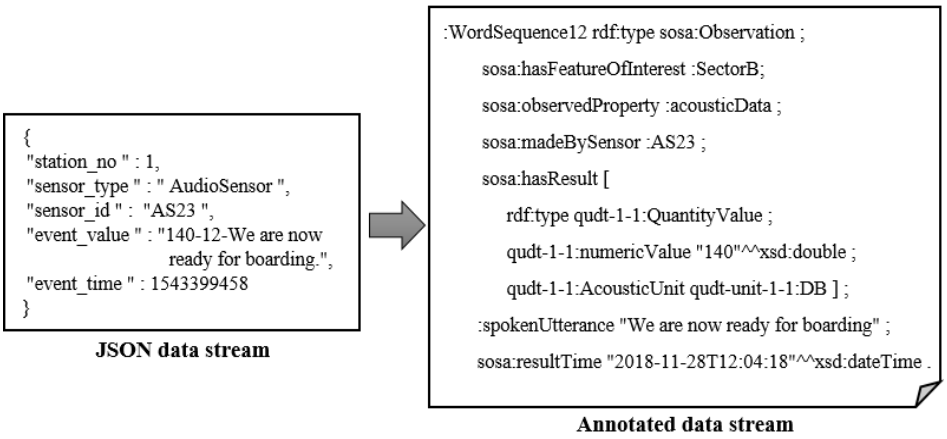

Fig. 2. Description of an audio sensor observation

The semantic model of a raw audio sensor stream originally written in JSON format is shown in Figure 2. The observation value gathered from the AS23 audio sensor is stored in the event_value field and comprises the following information in this order: the sound level measured in decibels, the identifier of the spoken utterance, and the sequence of words uttered by a passenger at a specific time in the airport. In the corresponding annotated data stream, we describe the observation made by an audio sensor and explicitly link the property being analyzed (the acoustic data) with the feature of interest (the airport sector where the audio observation was made). Due to the fact that some aspects such as detailed measure feature and units [23], are not tackled by the existing SSN ontology, we construct the RDF statements by hybridizing the SSN ontology with other schemas such as geospatial vocabularies - GeoSPARQL [19] - to model the location of the sensors in airport, and the Quantities, Units, Dimensions and Data Types Ontology (QUDT) [33] to model quantitative measurements. The resulted RDF descriptions are published into GraphDB for later analysis and querying.

Once the annotated data streams are persisted in the triplestore, we apply SPARQL-based reasoning to categorize the audio sensor observations into four main classes: LowerCritical, LowerNonCritical, UpperNonCritical and UpperCritical based on the sound level expressed in decibels and stored in the audio sensor observation result. In this way, we specify the severity ranges of audio sensor values in order to take rapid actions in case of abnormal operating conditions of the system.

In the following query we generate RDF statements by defining a rule according to which if the result value of an observation exceeds a specified threshold then the sensor observation is considered critical. We describe a rule-based query where all the observation results that contain a sound level value between 110 and 170 decibels are upper critical observations. This can be extended to rules that consider the presence of certain keywords

in the spoken utterance strings, or more advanced text processing rules based on GraphDB's full text indexing (Lucene) features.

```
insert
{
    ?observation a :UpperCritical
}
where
{
    ?observation sosa:hasResult/qudt-1-1:numericValue ?value
    filter ((?value > 110) && (?value < 170))
}
```

To find the latitude and longitude coordinates of a deployed audio sensor that captured a specific sentence of words of a passenger, we query the existing RDF database by following the chain of properties from the specific sentence to the value of the location coordinates as it can be seen in the second query. This query can be useful to detect the place where a suspicious sentence was uttered or an abnormal sound with increased decibel level (such as the sound of a gunshot) has sensed.

```
select ?coordinates
where
{
    :WordSequence12 sosa:madeBySensor/geo:hasGeometry/geo:asWKT ?
}
```

Figure 3 indicates the path of chaining properties for a more complex query where the system notifies the security operators responsible with the sector where a critical observation was made by sending them a message on their telephone.

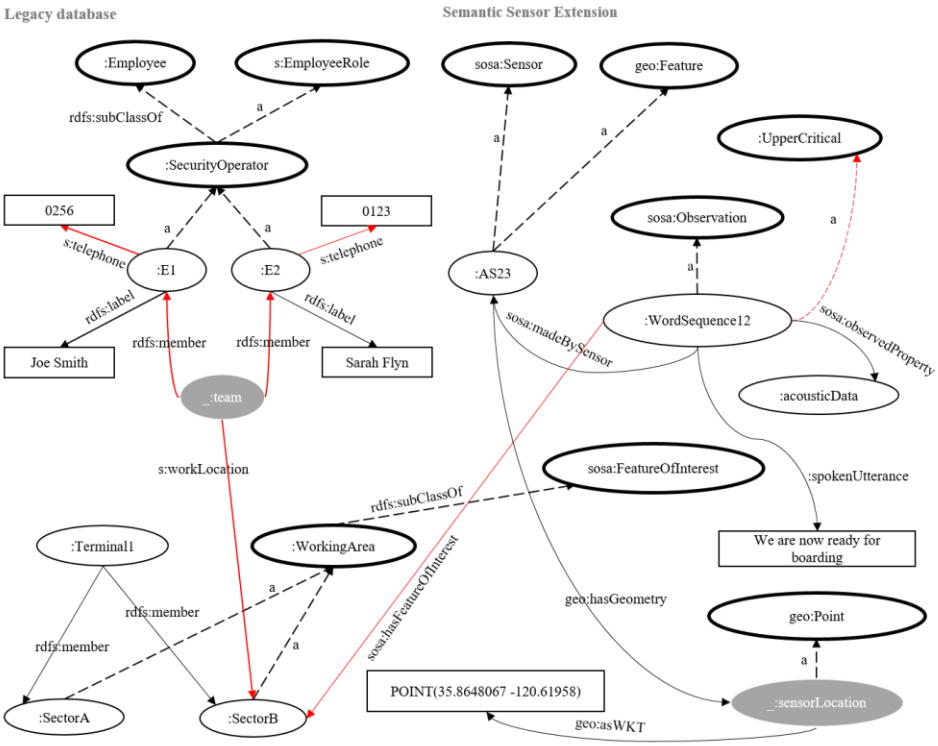

Fig. 3. Complex query to alert the security operator if a critical observation was sensed

In order to accomplish our goal we use the airport legacy information system that stores information about the employees and their working areas. We semantically lift this

traditional database to connect it with the graph that describes the sensor data - in this transformation process we use some classes and properties from different vocabularies: "s:EmployeeRole" class provided by the Schema.org [38] vocabulary to specify employee relationships or the "s:telephone" property to store the telephone number of a specific security operator. We also use the "rdfs:member" super-property of the RDF Schema [35] vocabulary to specify various membership relations (the membership of security operator instances to a team, or the decomposition of an airport terminal into sectors).

Geolocation information is also attached to the audio sensor descriptions for the purpose of future generalization (we plan to also transfer the proposal to generalized emergency interventions for natural disasters, where the granularity of geocoordinates and dynamic location sensors will become relevant for reasoning based on the GeoSPARQL standard).

6. Performance Evaluation and Results

The performance of the deployed SSP pipeline depends heavily on the software and hardware settings. Hence, several tests were conducted to investigate how the system performs with different configuration settings. All the tests were carried out on an Ubuntu 16.04.4 LTS x64-based PC with Intel(R) Core(TM) i7-7500U processor, 2.70 GHz CPU, 8 GB of RAM with Java version 1.8 and Java HotSpot(TM) 64-Bit Server VM. Confluent streaming platform version 4.0.0 was deployed with Apache Kafka version 1.0.0 and Apache Zookeeper version 3.4.10, which were the latest versions available at the time of building the SSP system. For RDF parsing of the JSON streams was used RDFLlib 4.2.2 package with SPARQL 1.1 implementation. To store the annotated streams the GraphDB 8.7 version was used.

6.1. Experiment design

In this section, we describe the Kafka configuration setup of the proposed SSP system. To meet the requirements of a real-time stream processing, the SSP system considers a sliding window to process the continuous sensor data. The window was defined to maintain a limited number of annotated streams into the triplestore. We evaluate the performance of the system by conducting two different Kafka-based scenarios. The following configurations were set up to decide which the most suitable Kafka-based scenario is for achieving the semantic modelling task.

- Kafka configuration 1: This is the minimum configuration consisting of a single node, with one Zookeeper instance and one broker instance, as it can be seen in Figure 4 (a). In this scenario, multiple simulated producers send data streams to the unique broker, which can handle thousands of incoming data seamlessly. These data streams are written to AudioSpeech topic that contains stream of records generated from audio sensors. We created this Kafka topic with a single partition and one replica factor. There is one consumer per topic that processes the data streams previously published.
- Kafka configuration 2: The Kafka cluster configuration consists of a single node with multiple brokers, managed by a single instance of Zookeeper, as it can be seen in Figure 4 (b). To balance the incoming load, the topic is broken down into multiple partitions containing sequences of messages that will be delivered asynchronously to the consumers to ensure parallelism. The consumer instances are grouped into consumer groups, one for each topic. In this configuration setup, the multi-subscriber topics may have zero, one, or multiple consumer instances who can access the data written to them.

In both scenarios, the producers run in their own thread and simultaneously publish data streams to the Kafka topic during a specific period. The timing of the produced data streams follows a Poisson process with a data rate that varies depending on the experiment. The Kafka cluster retains all the published stream of records, without consideration of their consumption. In all tests, a replication factor of one is used because the fault tolerance

measurement is out of scope in this paper. These configurations of the Kafka cluster have been tested to have not only an overview of the possibilities, the number of sensors sources and the amount of data streams that the system can handle, but also to monitor the number of RDF triples that the consumer instances generate over time.

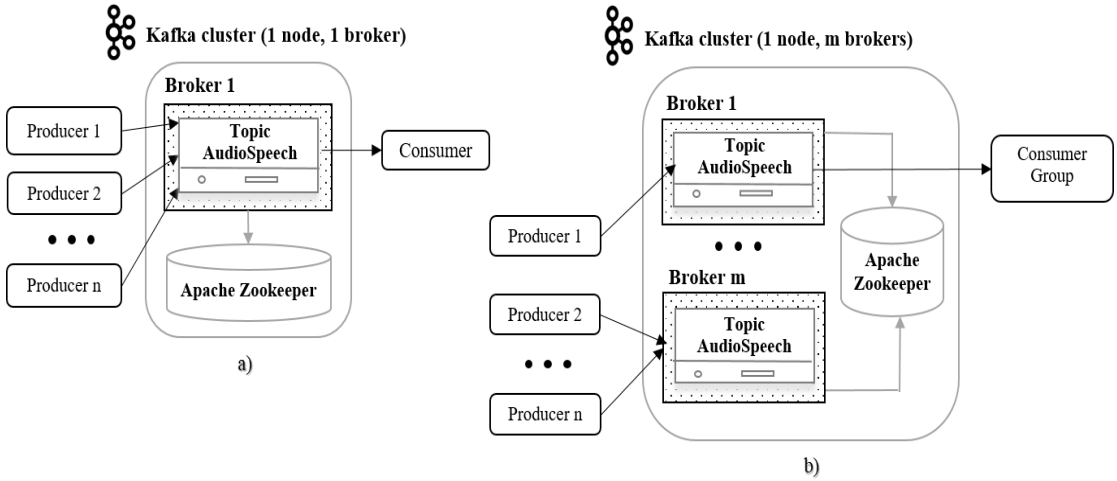

Fig. 4. Different Kafka-cluster configurations

To this end, the first experiment consisted in testing how many producers could be supported to write stream of records in one-broker versus multiple-brokers Kafka cluster setup. Another test was considered to analyze the consumer capability of reading and annotating the stream of data published to the Kafka cluster. Lastly, we evaluate the multiple brokers' Kafka-based cluster in terms of the number of generated RDF triples by increasing the execution time of the consumer instances.

6.2. Result analysis

The results shown in this paper regarding the second Kafka configuration are based on a three-broker Kafka cluster setup. We have varied the number of producers from 10 to 100 and measured the number of published and consumed messages in both Kafka-based configurations, during an execution of 10 minutes time.

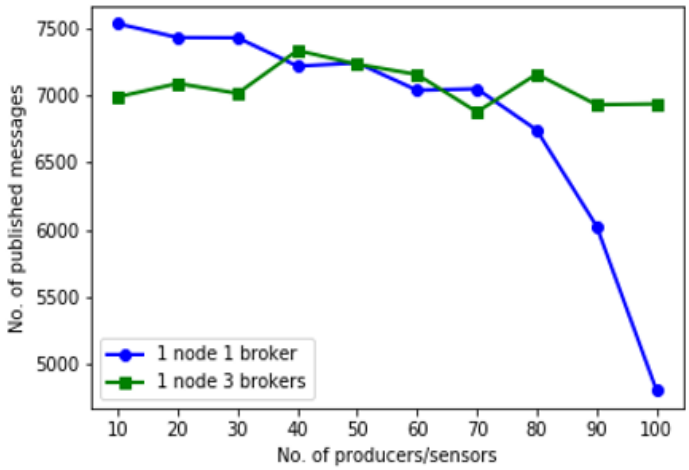

Fig. 5. The number of producers versus the number of published messages in both Kafka-based scenarios

We can see that in the chart of Figure 5 that the amount of published messages in a Kafka cluster consisting of a unique node with one broker instance, drops sharply when the number of producers increases from 80 to 100 producers, while the drop is smoother from 20 to 70 producers. This is caused by the limited network capacity and server write throughput of a single broker Kafka-based setup. In contrast, in the second Kafka-based

scenario, the Kafka cluster manages to cope with ingesting high volumes of data by distributing the write load over the three brokers.

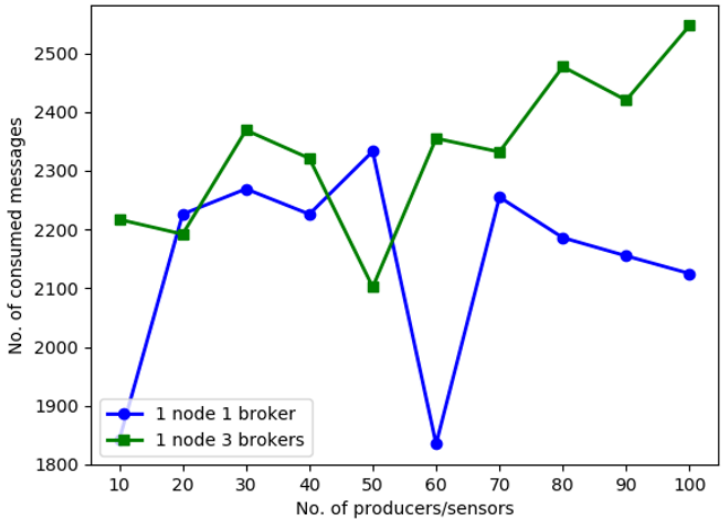

Fig. 6. The number of producers versus the number of consumed messages in both Kafka cluster setups

Besides the low capability of writing the sensor data into the specific topic, the Kafka cluster consisting of one node with one broker also has problems in consuming the previously published data. As there is only one consumer instance, the number of consumed and processed messages from the topic decreases starting with 70 producers. On a single node with three-broker Kafka-based cluster, there are multiple consumer instances (within the consumer group) that concurrently read the messages from the topic. The results presented in the chart from Figure 6 reveal that increasing the number of consumers ensures the parallelization of the semantic enrichment of the sensor data. The chart shows that adding more consumers to read and annotate the data streams improves the processing task.

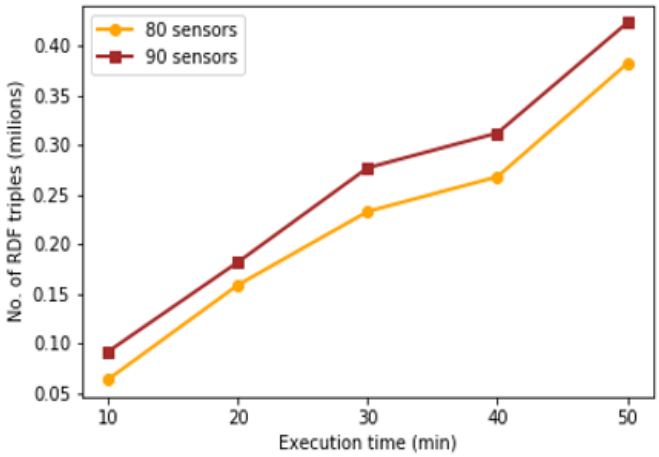

Fig. 7. The number of RDF triples versus the execution time in the multi-broker Kafka cluster setup

Figure 7 shows the performance of the proposed system by analyzing the number of RDF triples generated over time by consumer instances on a single node, three-broker Kafka cluster setup. We evaluate the system's performance using the streams published by 80 and 90 producers, respectively. The number of resulted RDF triples varies from 0.064 to 0.423 million of triples when testing our system. We observe that increasing the execution time from 10 to 50 minutes, the number of annotated data streams also increases.

7. Conclusions

This work at hand took a Design Science approach to develop a semantic pipeline for sensor stream processing. Experiments focus on relative performance and their results

show that the Kafka cluster consisting of multiple broker instances manages to cope with an increasing number of sensors by supporting write and read scalability of the streams. The concurrent reads of the data consumers also facilitates the parallelization of the semantic enrichment of the sensor data. Additional criteria are under consideration as this treatment iterates according to the DSR design and engineering cycle, having in mind the complex tableau of criteria that was systematized by [31]. A current limitation is that the reported experiments do not include the speech-to-text recognition effort, as this component is not to be developed in our project but reused (such components will be the subject to a comparison and selection process that was left out of this paper's scope).

Instead, the focus of our future work is to converge these ideas with the earlier results of [11] where parts of the machine-readable semantics are extracted from diagrammatic enterprise models. For this, a domain-specific modeling language (aligned with the SSN ontology) is necessary to capture a structure and visual overview of the airport premises and sensor network layout, thus facilitating decision support for business stakeholders familiar with their enterprise architecture semantics.

Also, we aim to generalize the proposal beyond the current smart airport scenario - we target emergency interventions and incident management during natural disasters, where dynamic geolocation sensors become more relevant considering the coverage and granularity of geocoordinates that can open additional opportunities for reasoning based on geo-comparison functions (e.g., GeoSPARQL). Further experiments will try to identify in such contexts where it is preferable to execute the semantic annotation in a high-performance architecture for edge computing.

References

1. Apache Flume, https://ume.apache.org/. Accessed June 13, 2019
2. Apache Kafka, https://kafka.apache.org/. Accessed June 13, 2019
3. Apache Nifi, https://nifi.apache.org/. Accessed June 13, 2019
4. Apache Spark, https://spark.apache.org/. Accessed June 13, 2019
5. Apache Storm, storm.apache.org/. Accessed June 13, 2019
6. Apache Zookeeper, https://zookeeper.apache.org/. Accessed June 13, 2019
7. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for C-SPARQL queries. In: Proceedings of the 13th International Conference on Extending Database Technology, pp. 441-452. ACM (2010)
8. Beck, H., Dao-Tran, M., Eiter, T., Fink, M.: LARS: A logic-based framework for analyzing reasoning over streams. In: AAAI'15 Proceedings of the 29th AAAI Conference on Artificial Intelligence, pp. 1431–1438, AAAI Press (2015)
9. Blumauer, A.: Introducing Graph-based Semantic Layers in Enterprises, https://semantic-web.com/2016/08/15/introducing-a-graph-based-semantic-layer-in-enterprises/. Accessed February 28, 2019
10. Bonte, P., Tommasini, R., Della Valle, E., De Turck, F., Ongenae, F.: Streaming MASSIF: Cascading Reasoning for Efficient Processing of IoT Data Streams. Sensors. 18(11), 3832 (2018)
11. Buchmann, R.A., Karagiannis, D.: Domain-specific diagrammatic modelling : a source of machine-readable semantics for the Internet of Things. Cluster Comput. 20(1), 895–908 (2017)
12. Calbimonte, J.-P., Corcho, O., Gray, A.J.: Enabling ontology-based access to streaming data sources. In: Proceedings of the 9th International Semantic Web Conference, pp. 96-111, Springer, Berlin, Heidelberg (2010)
13. Chen, J., Lécué, F., Pan, J. and Chen, H.: Learning from ontology streams with semantic concept drift. In: Proceedings of 26th International Joint Conference on Artificial Intelligence, pp. 957-963, Sierra C (2017)
14. D'Aniello, G., Gaeta, M., Orciuoli, F.: An approach based on semantic stream reasoning to support decision processes in smart cities. Telematics and Informatics. 35(1), 68-81 (2018)

15. Dell'Aglio, D., Della Valle, E., Calbimonte, J.P., Corcho, O.: RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. Semantic Web and Information Systems. 10(4), 17-44 (2014)
16. Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science. 1(1-2), 59-83 (2017)
17. Fisteus, J.A., García, N.F., Fernández, L.S., Fuentes-Lorenzo, D.: Ztreamy : A middleware for publishing semantic streams on the Web. Web Semant. Sci. Serv. Agents World Wide Web. 25(C), 16–23 (2014)
18. Fountain, T., Tilak, S., Shin, P., Nekrasov, M.: The open source dataturbine initiative: empowering the scientific community with streaming data middleware. The Bulletin of the Ecological Society of America 93(3), 242-252 (2012)
19. GeoSPARQL, https://www.opengeospatial.org/standards/geosparql. Accessed June 14, 2019
20. GraphDB, http://graphdb.ontotext.com/. Accessed June 13, 2019
21. Howard, P.: Graph and RDF Databases 2015, https://www.bloorresearch.com/research/ /graph-rdf-databases-2015/. Accessed June 13, 2019
22. Janowicz, K., Haller, A., Cox, S.J., Le Phuoc, D., Lefrançois, M.: SOSA: A lightweight ontology for sensors, observations, samples, and actuators. J. of Web Semantics (2018)
23. Lai, C., Pintus, A., Serra, A.: Using the Web of Data in Semantic Sensor Networks. In: Barolli, L., Terzo, O. (eds.) Complex, Intelligent, and Software Intensive Systems. CISIS 2017. Advances in Intelligent Systems and Computing, vol. 611, pp. 106-116. Springer, Cham (2018)
24. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: Proceedings of 10th International Semantic Web Conference, pp. 370-388. Springer, Berlin, Heidelberg (2011)
25. Le-Phuoc, D., Xavier Parreira, J., Hauswirth, M.: Linked Stream Data Processing. In: Eiter, T., Krennwallner, T. (eds.) Reasoning Web. Semantic Technologies for Advanced Query Answering. Reasoning Web 2012, Lecture Notes in Computer Science, vol. 7487, pp. 245-289, Springer, Berlin, Heidelberg (2012)
26. Le-Phuoc, D., Nguyen Mau Quoc, H., Le Van, C., Hauswirth, M.: Elastic and Scalable Processing of Linked Stream Data in the Cloud. In: Alani H. et al. (eds.) The Semantic Web-ISWC 2013. ISWC 2013. Lecture Notes in Computer Science, vol. 8218, pp. 280-297, Springer, Berlin, Heidelberg (2013)
27. Le-Phuoc, D., Nguyen-Mau, H.Q., Parreira, J.X. and Hauswirth, M.: A middleware framework for scalable management of linked streams. Web Semant. Sci. Serv. Agents World Wide Web. 16(C), 42-51 (2012)
28. Linked Data Orchestration, https://linkeddataorchestration.com/the-year-of-the-graph/. Accessed June 13, 2019
29. Narkhede, N., Shapira, G., Palino, T.: Kafka: The Definitive Guide: Real-time Data and Stream Processing at Scale. O'Reilly Media, Inc., (2017)
30. Pacha, S., Ramalingam, S., Sethukarasi, R.: Semantic annotation of summarized sensor data stream for effective query processing. J. Supercomput. 1-23 (2017)
31. Prat, N., Comyn-Wattiau, I., Akoka, J.: Artifact evaluation in information systems design science research a holistic view. In: Proceedings of the 19th Pacific Asia Conference on Information Systems (PACIS 2014), pp. 23. (2014)
32. OpenLink Virtuoso, https://virtuoso.openlinksw.com/. Accessed June 13, 2019
33. QUDT, http://www.qudt.org/. Accessed June 14, 2019
34. RabbitMQ, https://www.rabbitmq.com/. Accessed June 13, 2019
35. RDF Schema 1.1, https://www.w3.org/TR/rdf-schema/. Accessed June 14, 2019
36. Resource Description Framework (RDF), https://www.w3.org/RDF/. Accessed June 13, 2019
37. Ren, X., Cur, O.: Strider: A hybrid adaptive distributed RDF stream processing engine. In: International Semantic Web Conference, pp. 559-576, Springer (2017)
38. Schema.org, https://schema.org/. Accessed June 13, 2019

39. Semantic Sensor Network Ontology, https://www.w3.org/TR/vocab-ssn/. Accessed June 13, 2019
40. Semantic Sensor Network XG, http://www.w3.org/2005/Incubator/ssn/. Accessed June 13, 2019
41. SPARQL 1.1 Query Language, https://www.w3.org/TR/sparql11-query/. Accessed June 13, 2019\
42. Varangaonkar, A.: 2018 is the year of graph databases. Here's why. https://hub.packtpub.com/2018-year-of-graph-databases/. Accessed June 13, 2019
43. Wieringa, R. J.: Design Science Methodology for Information Systems and Software Engineering, Springer-Verlag, Berlin, Heidelberg (2014)
44. Zalhan, P. -G., Stan, A., Teodorescu, L.-R., Saupe, A.-B., Duma, M.: A Kaldi-based ASR Solution for the Romanian Judicial System. In: Proceedings of the 15th International Conference on Informatics in Economy (IE 2016) Education Research & Business Technologies, 2016, pp. 191-197, Cluj-Napoca (2016)