SCOUTER: A Stream Processing Tool to Contextualize Anomaly Detection

Badre Belabbess[1,2], Musab Bairat[1], Jeremy Lhez[2], Zakaria Khattabi[1] and Olivier Curé[2]

[1] ATOS, F-95870, Bezons, France. {firstname.lastname}@atos.net
[2] LIGM (UMR 8049), CNRS, ENPC, ESIEE, UPEM, F-77454, Marne-la-Valle, France. {firstname.lastname}@u-pem.fr

Abstract. Anomaly detection is a key feature of applications processing Internet of Things sensor measures. In a recent project, we highlighted that to guarantee high quality detections, metadata providing spatio-temporal contexts on sensor measures are needed. These metadata hence become an integral component of the anomaly detection computation and need to be processed using a streaming approach. In this paper, we introduce Scouter, a generic tool that helps in capturing, analyzing, scoring and storing the contextual information of a given application domain. The processing depends on a semantic-based approach that exploits ontologies to score the relevancy of contextual information. The paper provides details on the system's architecture, describes lessons learned and introduces practical aspects.

Keywords: Real Time, Streaming, Anomaly Detection, Annotation, context information, Topic extraction, Sentiment Analysis

1 Introduction

Big data and stream processing are nowadays a big hype. Large amounts of data, which are generated every day by streaming resources obtained from the Internet of Things (IoT), are continuously accumulated and stored in Big Data platforms. The analysis of these data supports intelligent software functionalities usually based on machine learning or semantic-based methods. Among these features, anomaly detection[1] is predominant and is tackling domains as diverse as medicine (*e.g.*, to identify malignant tumors in MRI images), finance (*e.g.*, to discover cases of credit card transaction frauds), information technology (*e.g.*, to detect hacking situations in computer networks). In the Waves project[3], we are interested in detecting anomalies in a large network (over 100.000 km) distributing drinkable water to over 12 million clients. On some experiments, we found out that to guarantee high accuracy in anomaly detection, a contextualization of the measures is needed. For instance, abnormal high pressure and flow measures are typical of a water leak. But consider these values are measured

[3] http://www.waves-rsp.org/

on a week-end day for a given residential area. Then, high water consumptions could be explained by a sport or cultural event happening in that area or simply by some hot weather conditions implying garden watering. Hence, an efficient approach for anomaly detection has to integrate a spatio-temporal context on analyzed measures.

The tasks related to the capturing, analyzing, scoring and storing of contextual metadata are demanding since they require interactions between software components that may be hard to set up, especially in a distributed environment. These components generally handle stream, natural language and ontology processing as well as the storing of complex data structures, etc.. The Scouter[4] tool aims at simplifying these tasks by proposing implementations of the main functionalities and by taking care of installation and configuration aspects. Designed as a generic tool, Scouter relies on the declaration of the data sources one needs to retrieve contextual information from, *e.g.*, tweets, RSS feeds. Then it parses these textual information to discover occurrences of terms which are stored in a set of ontologies. Thus, these texts are annotated with ontology concepts. The scoring module takes advantage of user defined weights,*i.e.*, a real value in the $[0,1]$ range, associated to ontology concepts to provide an overall scoring for each text. Since, the texts have been edited at a precise date and concern a given location, we obtain spatio-temporal, scored contexts for a given domain that can be used to confirm an anomaly detected from IoT measures. To the best of our knowledge, Scouter is a first attempt at providing a semantic-based, stream processing oriented contextualization for anomaly detection. We consider that the different modules implemented in Scouter, can help in IoT projects.

2 Architecture

Scouter was developed to be generic, fully configurable and easy to use. As a summary, data is fetched from different web sources, processed and stored in a NoSQL database to be read later on and fed to a messaging broker for integration with other systems. The main components of our system are: a set of Web data connectors, an analytics unit, storage units, a messaging broker and a web service unit. Figure: 1 provides the architecture of Scouter.

The web connectors unit includes connectors that consume data from different data sources at a certain frequency, and based on predefined configurations which can be specified using a web interface. These sources include: Twitter stream feed using Twitter4J[5] library, Facebook graph API via RESTfb[6], RSS feeds from different providers configured before running, Open Weather Map[7], Open Agenda[8] as a source of events. All of these data sources are consumed in

[4] Scouter can be obtained at https://badrebelabbess.github.io/Scouter-1/
[5] Twitter4J: http://twitter4j.org/en/
[6] RESTfb: http://restfb.com/
[7] OWM: https://openweathermap.org/api
[8] Open Agenda: https://openagenda.zendesk.com/hc/fr

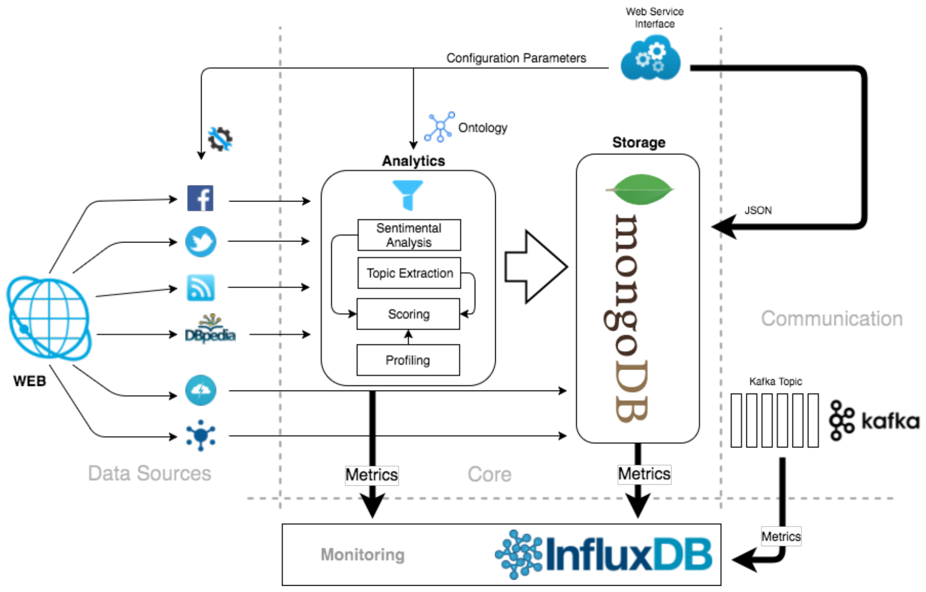

Fig. 1: Scouter Architecture

an efficient multi-threading mechanism using rest APIs or HTTP connections. Keywords used in fetching data are passed to the system through an ontology.

The profiling component provides geographical characteristics for the use case zone, it determines the type of terrain (expressed as an ontology concept) in the zone studied by Scouter. Using geographic data (points of interest, terrain areas) for a given zone, a profile is generated to assist the user in configuring the web connectors. Fetched data from previously mentioned sources are pipelined to a scoring unit to be processed and recorded as events, having location, start date, end date and description. The events are processed in order to guess relevancy to the use case, events which are irrelevant are dropped. For this purpose, sentiment analysis and topic extraction algorithms are implemented in the scoring unit. For topic extraction, behind the scenes, KEA[9] algorithm is used to determine topics from an input text, it supports English, French & Spanish languages, it's simple but yet powerful. KEA algorithm needs training to build a proper model that can be used for evaluation, and our Scouter provides an easy way for preparing the model without the need to code. The last step of scoring involves sentiment analysis that is done based on Apache OpenNLP [10] Library.Leveraging on the maximum entropy algorithm [2], Open NLP's Document Categorizer is able to classify text into predefined categories. It builds a model using multinomial logistic regression to determine the right category for a given text. The user should enter a training data set containing multiple texts

[9] KEA: http://www.nzdl.org/Kea/
[10] https://opennlp.apache.org/

related to his use case along with a positive score (set to 1) or a negative score (set to 0). After scoring, events are recorded into a MongoDB database instance. We chose MongoDB due to the following facts: easy to install an configure, rich indexing facilities, geographical features, scalability and distributed capabilities. Scouter also provides metrics monitoring tool to track the performance of the system, different metrics including query times, event processing times, events count and topic extraction training time are stored in InfluxDB, which is a time series database with very high reading and writing speed.

Recorded events in the database are ready to be consumed by platforms this system was developed to support. Apache Kafka[11] is becoming more popular and widely used in stream processing engines, it can be connected to most of the open-source stream processing frameworks including Apache Spark Streaming and Apache Storm. It is a distributed messaging broker component based on the publish-subscribe approach, and hence, it was our choice for implementing messaging broking. Events are retrieved from MongoDB and written into a Kafka topic, on the other end of this messaging bus, stream processing platforms consume these events from that topic. Before writing these events to a topic, it should become RDF compatible. Apache Jena[12] is used for converting these events into RDF events.

The Web services component is used for the sake of configuring the system and when it's run in a standalone mode. It provides Rest web services that can be integrated with an user interface to deliver configuration parameters in an easy and human readable way. Moreover, it can be used to fetch RDF events from the database if it runs in a standalone mode. Vertx[13], an event-driven, non-blocking web development framework, was used to develop this component, it's a highly scalable framework with very high performance when it comes to implementing web services.

3 Scouter in practice

In order to use Scouter, a web interface is used to build a workflow and to configure different components. This workflow comes with a default flow option that connects all components together. If you want to create a new workflow, you start by defining a geo-location area on a map (for our running example, the Versailles (France) area was chosen), this will create a bounding box for the data to be fetched, adding geo-filtering to our system.

Adding a data source simply requires a drag and drop interaction and some configuration. For instance, the DBPedia data source should have at least a query to execute for fetching data, and the time frequency at which the system should retrieve data. Scouter listens to Twitter streams API (and Facebook) for tracking a list of keywords, but it can also track special accounts and hash-tags in case of interest (in our use case: MonVersailles and Versailles.officiel). Other

[11] Apache Kafka: https://kafka.apache.org/
[12] Apache Jena: https://jena.apache.org/
[13] Vertx: http://vertx.io/

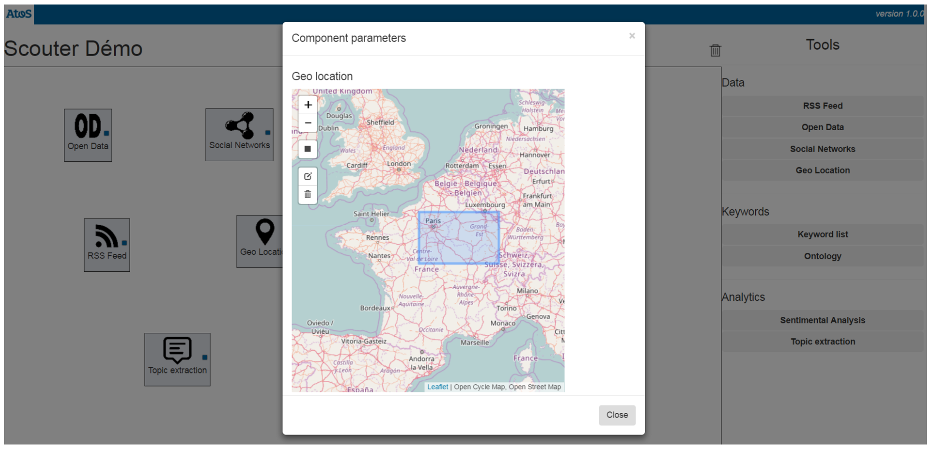

Fig. 2: Scouter Screenshot

data sources parameters that can be accessed in the same way are: start date and frequency for Open Agenda, frequency for Open Weather Map. For RSS feeds, the frequency and a list of links and resources names (*e.g.*, newspapers) should be provided (in our use case, Le Parisien and versailles.fr were subscribed).

These texts need to be scored according to the relevancy of the anomalies to detect. This scoring is based on annotating the texts. This is performed by parsing the texts for predefined keywords which are defined in a set of ontologies. Each ontology concept (a concept could be water, fire, event ...etc. In our case: a leak) is associated with a relevancy weight, *i.e.*, a [0,1] real value. Intuitively, a concept describes an anomaly and a weight qualifies to what extent this concept may justify the anomaly. Moreover, each concept has a list of keywords, and their variations (misspelling, etc.),which are also weighted terms, *i.e.*, seep, leakage.

In addition, the system provides several analytics tools such as topic extraction and sentimental analysis to improve annotations. For topic extraction, a simple training is needed to process the events properly and extract terms. To do so, Scouter provides the user with the possibility to upload a set of files with *.txt* and *.key* extensions. Each pair of files {*.txt*,*.key*} should be of the same name. Text files should include content related to the domain, while key files should include the main topics that text files are about. For sentimental analysis, no configuration is needed (more details on Scouter's github page).

4 Semantic Benefits and Challenges

4.1 RDF

Scouter aims at processing heterogeneous data from various types of sources in the web, therefore we need to ensure syntactic unification using a flexible, graph-based data model and a common vocabulary to validate interoperability. The RDF data model fulfills this gap for us.

Only a small amount of available data in the web are actually used. The primary reason comes from the lack of simple approaches to interconnect the data from various source types (JSON, XML, HTML, etc). By leveraging on RDF, Scouter is able to alleviate some of these problems by providing a set of standards for representing and connecting the data. Consequently, other applications can discover the results provided by our system since the format (JSON in MongoDB) can be easily handled. Moreover, to generate valid linked data we need data to be in a shape that is easy to manipulate and convert to RDF. The serialization format used in our system is JSON-LD, amongst its main advantages, it has a compact data format to exchange data between applications without overloading the network. In addition to that, there is a good tool support where almost every programming language supports JSON and less overhead than XML is applied when parsing and serializing processes are performed. However, several strong challenges appear when facing the conception and implementation of such system: (1) the availability of appropriate vocabularies to fully develop a complex natural language process related to our use case, (3) the availability of tools to ease the task of data acquisition, verification and annotation, (3) the lack of semantic annotations in the different resources especially for RSS feeds, news web sites, (4) the potential semantic mismatch when processing topic extraction and sentiment analysis

4.2 Linked Data

The Linked Open Data (LOD), an initiative of the World Wide Web consortium, allows the publication of knowledge bases in an interlinked way and the querying of different sources with the SPARQL query language. We studied and used several sources of linked data for our project, considering their advantages and drawbacks. Thus, we adopted OpenStreetMap[14] (OSM), a collaborative project for a free world map, to establish the geographical profiling of Scouter, because of its completeness and the structure of its data. Besides the data from the sources, *i.e.*, the facts part of a knowledge base, users of the linked data also have access to the ontology structuring it, allowing better comprehension and reuse. Another advantage is the community: being open means that any user can enrich the content, by adding new information or updating obsolete one. OSM for example, has over two million registered users; Dbpedia[15], another famous part of the LOD structuring information from Wikipedia, regroups over three billions RDF triples. Such a quantity of data can be fetched in different ways: most sources can be queried via SPARQL endpoints, and there are APIs developed specifically in each case, *e.g.*, Overpass for OSM.

The availability and mass editing of knowledge bases also cause some specific problems: because of frequent updates, consistency and homogeneous quality cannot always be verified or guaranteed. Geonames[16], another geographical

[14] https://www.openstreetmap.org/
[15] http://wiki.dbpedia.org/
[16] http://www.geonames.org/

database, still has places where very few data have been annotated yet, in spite of its ten millions of geographical names. Another recurring problem is the interconnection of similar knowledges: even with RDF, such a representation is not always efficient. The use of the property *owl:sameAs*, for example, has controversial uses [3] which can be a semantic issue for interlinking knowledges.

5 Lessons Learned

In this section, we present lessons learned along several distinct dimensions.

Modularity: Instead of trying to adapt our implementation to heterogeneous technologies in the Big Data world, the best approach was to create a simple but powerful bridge that would make the integration seamless. The right choice was to go with a messaging queue system that is widely used and known for its robustness. Soon, Apache Kafka appeared to be the fittest candidate, especially since many Big Data platforms rely heavily on it for its distribution capabilities.

Relevancy: We found out that few data sources with high quality content outperformed multiple data sources with medium to low quality content. Therefore, we decided to keep only 4 different categories and for each one we selected the most suited source. The key component here is to interact with a domain expert, in our use case a water consumption expert, in order to assess thoroughly the relevancy of a data source.

Semantics: To deal with various types of data source one must ask the question of vocabulary consistency and concepts definition. At the beginning, we tried to encapsulate each data source within its own semantic implementation, but soon we faced the problem of genericity. Since we are aiming at a tool adjustable to multiple use cases, going the ontology way seemed to be a win solution. Hence, we could give the possibility to every domain expert to use his own ontology with the specific concepts, properties and keywords that suit his needs.

NLP: The key component for a successful implementation is to find the right models and the proper scores. Even tough a lot of libraries were available for sentiment analysis and topic extraction, many of them lacked robustness, flexibility, ease of use and sometimes they had limitations (commercial license above a certain threshold). Instead of investing much time on finding the fittest technologies, we are quite positive that we can highly improve the results by investing more time on ontology modeling and scoring.

Packaging: Our first approach was to provide a robust and well rounded module with an already built-in implementation for Kafka messaging queue system. However, after reviewing the users' experience from multiple other reusable resources, we found out that the best way to remove complexity was to package the code into a user friendly web application. Moreover, to enhance the maintainability of the code, we decided to go for a system based on flexible connectors that can easily be activated or deactivated depending on the needs of the use case.

Deployment: As for the deployment part, we found out that using containerization technologies such as Docker[17] remove the burden of deployment, can be launched within a minute and can start displaying results without caring about the technical implementation. In addition, docker containers are easily configurable and pluggable to other containers that hold different processes, which can then constitute a real complex set of components for a much advanced global system.

6 Conclusion

In this paper, we presented Scouter, a tool that demonstrated its usefulness in the Waves project for improving the contextualization of identified anomalies. The primary goal was to offer a generic system that can adapt to any Big Data platform whatever the engine used and without losing the capability to process various types of data sources. To achieve such target, we chose an approach based on data connectors relying heavily on messaging queue system and we offer multiple NLP functionalities such as topic extraction and sentiment analysis.

Because of its easy-to-use Docker package, Scouter is already used in other prototypes at Atos and we are aiming to extend it with novel features such as ontology enrichment based on a dictionary of concepts and the identification of duplicate events coming from different data sources. Finally, we plan to improve the implementation by supporting various ontology formats (*e.g.*, ttl, N3, RDF/XML, etc.) and adding new data sources to fit most use cases (*e.g.*, traffic information, etc.).

Acknowledgments

This work has been supported by the WAVES project which is partially supported by the French FUI (Fonds Unique Interministriel) call #17. The WAVES consortium is composed of industrial partners Atos, Ondeo Systems and Data publica, and academic partners ISEP and UPEM.

References

1. V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv.*, vol. 41, pp. 15:1–15:58, July 2009.
2. S. D. P. Adam Berger and V. D. Pietra, “A maximum entropy approach to natural language processing,” *Computational Linguistics-MIT*, pp. 22–1, 1996.
3. H. Halpin, P. Hayes, J. McCusker, D. Mcguinness, and H. Thompson, “When owl: sameas isnt the same: An analysis of identity in linked data,” *The Semantic Web–ISWC 2010*, pp. 305–320, 2010.

[17] https://www.docker.com/what-docker