Horizon 2020 Program (2014-2020)

Big data PPP

Research addressing main technology challenges of the data economy

Industrial-Driven Big Data as a Self-Service Solution

D4.3: Streaming Analytics and Predictions[†]

Abstract: This deliverable describes several aspects of the real-time streaming reference architecture of I-BiDaaS. The architecture is compatible with PMML, which enables rapid specification and development of new functionality. It also contains interconnection primitives between the batch and stream processing layers within I-BiDaaS, as well as a proper visualization front-end that allows the real-time display of analytics via auto-updated visual elements. Overall, I-BiDaaS provides the planning, design considerations, and best practices for implementing real-time streaming solutions. To demonstrate its capabilities, we provide a use-case that focus on the processing of Call-Center transcription logs. By processing this data at real-time, enterprises can create models for QoS prediction as well as derive insights into customer metrics and incidents.

Contractual Date of Delivery	30/06/2020
Actual Date of Delivery	30/06/2020
Deliverable Security Class	Public
Editor	*Giorgos Vasiliadis (FORTH)* *Christoforos Leventis (FORTH)*
Contributors	FORTH, SAG, BSC, AEGIS
Quality Assurance	*Gerald Ristow (SAG)* *Raul Sirvent (BSC)*

[†] *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780787.*

The *I-BiDaaS* Consortium

Foundation for Research and Technology – Hellas (FORTH)	Coordinator	Greece
Barcelona Supercomputing Center (BSC)	Principal Contractor	Spain
IBM Israel – Science and Technology LTD (IBM)	Principal Contractor	Israel
Centro Ricerche FIAT (FCA/CRF)	Principal Contractor	Italy
Software AG (SAG)	Principal Contractor	Germany
Caixabank S.A. (CAIXA)	Principal Contractor	Spain
University of Manchester (UNIMAN)	Principal Contractor	United Kingdom
Ecole Nationale des Ponts et Chaussees (ENPC)	Principal Contractor	France
ATOS Spain S.A. (ATOS)	Principal Contractor	Spain
Aegis IT Research LTD (AEGIS)	Principal Contractor	United Kingdom
Information Technology for Market Leadership (ITML)	Principal Contractor	Greece
University of Novi Sad Faculty of Sciences (UNSPMF)	Principal Contractor	Serbia
Telefonica Investigation y Desarrollo S.A. (TID)	Principal Contractor	Spain

Document Revisions & Quality Assurance

Internal Reviewers

1. *Gerald Ristow, (SAG)*
2. *Raul Sirvent (BSC)*

Revisions

Version	Date	By	Overview
5.0.0	25/06/2020	Giorgos Vasiliadis	Final version
4.0.0	18/06/2020	Giorgos Vasiliadis	Corrections and fixes
3.0.1	12/06/2020	Gerald Ristow	Revised version of Section 3
3.0.0	10/06/2020	Giorgos Vasiliadis	Third revision based on comments received from Gerald Ristow
2.0.0	05/06/2020	Giorgos Vasiliadis	Second revision based on comments received from Raul Sirvent
1.0.0	31/05/2020	Giorgos Vasiliadis	First revision
0.0.5	28/05/2020	Leonidas Kallipolitis	Provided text for Section 5
0.0.4	19/05/2020	Giorgos Vasiliadis	Provided text for Section 4
0.0.3	18/05/2020	Raul Sirvent	Provided text for Section 2
0.0.2	11/05/2020	Gerald Ristow	Provided text for Section 3
0.0.1	27/04/2020	Dusan Jakovetic	Comments on the ToC
0.0.0	21/04/2020	Giorgos Vasiliadis	ToC

Table of Contents

List of Figures

List of Abbreviations

CORS	Cross-Origin Resource Sharing
DFA	Deterministic Finite Automaton
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
KNIME	Konstanz Information Miner
MQTT	Message Queuing Telemetry Transport
ML	Machine Learning
MVP	Minimum Viable Product
PMML	Predictive Model Markup Language
QoS	Quality of Service
SVM	Support Vector Machines
UM	Universal Messaging
WP	Work Package
XML	Extensible Markup Language

Executive Summary

Streaming analysis systems calculate statistical analytics periodically while moving within the stream of data, typically, by maintaining the data in memory, on a time-windows basis. As a result, they rely on the notion of rolling data streaming analysis, in which the results are incrementally updated as new data arrives.

In this document, we describe several aspects of the real-time streaming reference architecture of I-BiDaaS. First, we show the interconnection mechanisms between the batch and stream layers – even though these layers typically operate independently, the architecture enables the potential of connecting both layers. We also describe the runtime environment for predictive analytics models that is supported by I-BiDaaS, based on the open standard Predictive Model Markup Language (PMML) of Data Mining Group. With PMML, it is easy to develop a model on one system using one application and deploy the model on another system using another application, simply by transmitting the PMML file. The use of PMML enables the rapid specification and development of new functionality. We also show the corresponding front-end visualizations that have been implemented to allow the display of analytics via real-time auto-updated visual elements.

To demonstrate its capabilities, we provide a use-case that is focused on the processing of call center transcription logs. By streaming this data and processing at real-time, enterprises can create models for QoS and customer experience predictions, as well as derive insights into customer metrics and incidents.

1 Introduction

Many approaches have been focused on designing scalable data analysis platforms where the computing and storage resources can be scaled out for the processing and analysis of the ever-growing big data [1] [2] [3] [4]. Typical platforms for such processing are based on a batch-oriented distributed computing framework [3] [4]. However, this model of "store-and-process" approach has several limitations. First, it requires data to be stored and managed, which requires substantial storage resources, especially for high-velocity data. In addition, due to the batch-oriented design of these programming models, measurement results cannot be provided in hard real-time. This may have an impact on latency-critical scenarios, such as anomaly detection, accounting and QoS. To overcome these constraints and be able to analyze data at real-time while coping with such big amounts of data, many systems rely on the notion of rolling data streaming analysis, in which the results are incrementally updated (rolled-over) as new data arrives [5] [6]. These systems, typically, maintain the data in memory, on a time-windows basis and produce the appropriate summary statistics periodically.

This document is the outcome of Task 4.1 (namely, "Customization and advancements of the complex event processing engine"), Task 4.2 (namely, "Distributed Complex Event Processing") and Task 4.3 (namely, "Real-time predictive Analytics"), and it includes the technological advancements of WP4 after D4.2 [7] that was submitted at M24. In particular, it shows several aspects of the streaming analytics components of I-BiDaaS, and the different techniques we have implemented to provide flexibility and acceleration. Overall, I-BiDaaS can process incoming streams at real-time and provide aggregated results that can optionally be processed further by external analysis modules or visualization frameworks. It also simplifies development as it is compatible with the PMML, and enables the rapid specification and composition of new processing functionalities, allowing the developers to focus solely on their application logic. We also show interconnection primitives between the batch and stream processing layers within I-BiDaaS.

The I-BiDaaS reference architecture for streaming analytics and predictions, which is part of the I-BiDaaS platform shown in Figure 1. It can be used across several enterprise use cases, including healthcare industry, retailers, transportation, cybersecurity, etc. For our case, we focus on call center transcription logs. These logs include transcripts data from calls received from three different call centers. By streaming this data and processing at real-time, enterprises can create models for QoS prediction as well as derive insights into customer metrics and incidents.

The remainder of the deliverable is organized as follows. Section 2 shows the interconnection mechanisms between the batch and stream layers of I-BiDaaS. Section 3 describes the runtime environment for predictive analytics models that is supported by I-BiDaaS, based on the open standard Predictive Model Markup Language (PMML) of Data Mining Group. Section 4 describes the use of text analytics and predictions in the call centers domain, in order to improve the QoS and the customer experience. Section 5 shows the corresponding visualizations that have been implemented to display streaming analytics via auto-updated visual elements. Finally, Section 6 provides the conclusions of this deliverable.

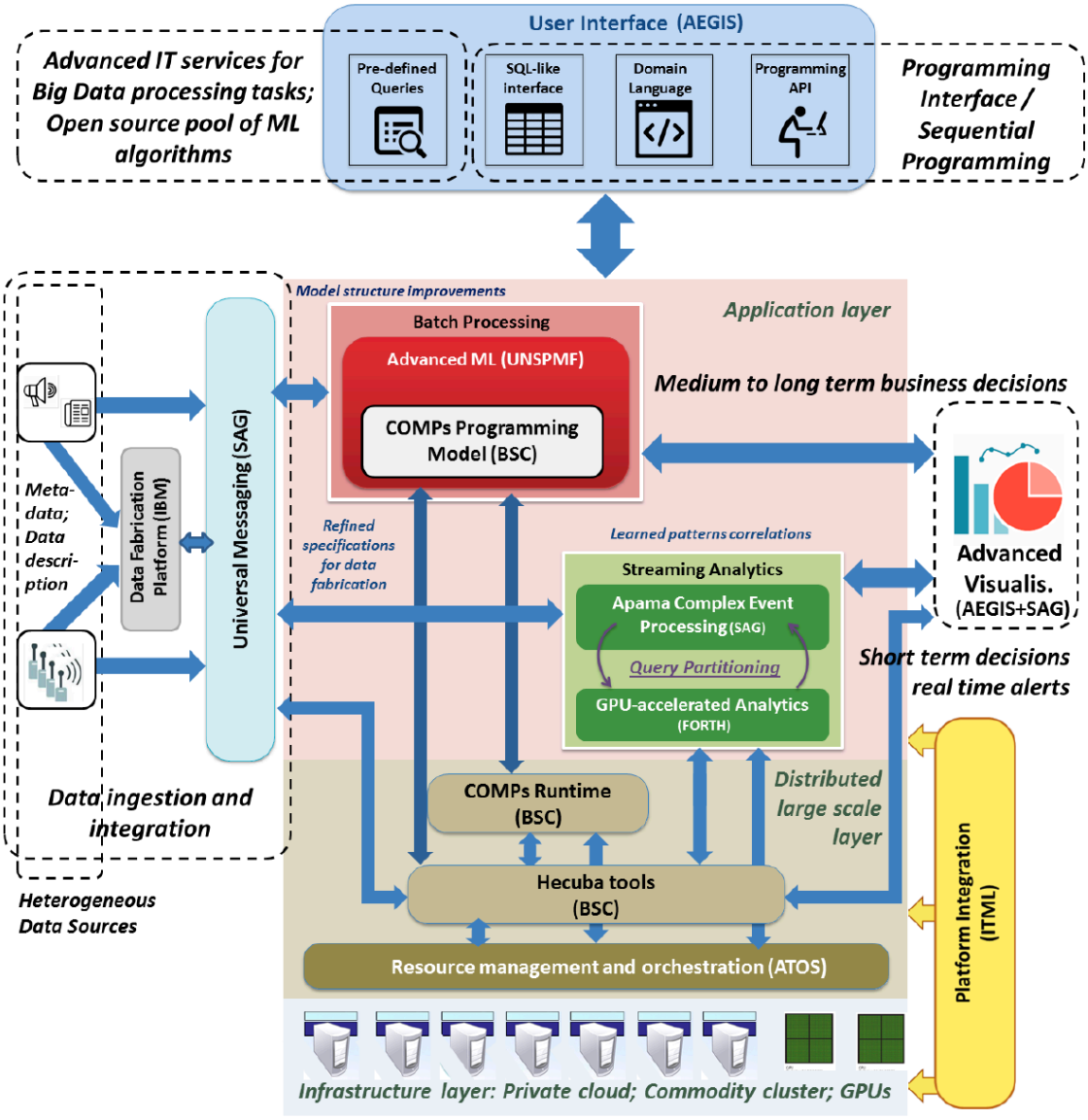

Figure 1: The I-BiDaaS architecture

2 Interconnection of Batch and Stream Processing Layers

Although the batch and stream processing layers in the I-BiDaaS architecture operate independently in each of the use cases of the project [8] [9], the architecture designed in the project provides the capability of connecting both layers. It is clear that streaming and batch data analytics have very different requirements in terms of data processing. However, many real-life use cases are willing not only to process streams of data but also to store parts of them to run batch processes time after the streams have been processed. These processes typically would try to verify the conclusions extracted from an initial analysis done in real-time with the streams. Once the approach is shifted from streaming to batch, a more time-consuming processing of the data can happen (e.g., increasing accuracy, sensitivity, etc.).

With that objective in mind, we have implemented a connection layer between the batch and stream layers in I-BiDaaS, using the Hecuba framework [10]. The objective of this connector is to listen to an MQTT queue, and write the data to a persistent storage (i.e., Apache Cassandra [11]) for later analysis. Hecuba simplifies the usage of the database in the connector, since data can be treated as regular in-memory objects that can be made persistent on demand. The code is then free of any vendor lock-in, since there is no programming directly towards the Cassandra interface, only to Hecuba. This also means that the underlying database (Cassandra) could be changed for a different one, without changing a single line of code in the developed connector.

The connector is implemented in Python, using the paho-mqtt library [12], to achieve the connection to the Mosquitto server [13] (testing) or the Universal Messaging component, that handles the queue messages, as shown in Figure 1. Once a message is received, the objects are serialized using pydantic [14], which allows defining a schema of the object to be serialized using Python3 hint system, enabling its validation before storing the object to the database in the Apache Cassandra cluster deployed for that purpose. The messages are encoded in JSON format and a data model has been defined for the structure of the message.

A representative example of this interaction can be depicted with an imaginary financial use case, in which a producer (`bank_producer`) would emit log information about possible bank activities performed by the users. Then, a receiver (`bank_receiver`) could receive the messages, validate their format and store them in the Cassandra database.

3 Run-time Environment for Predictive Analytics

In this section, we describe the runtime environment for predictive analytics models that I-BiDaaS provides, which allows data scientists to develop prediction models in their familiar environment and languages (e.g., Python, KNIME [15], etc.) and deploy them to the runtime engine.

In order to analyze data via machine learning algorithms, data scientists use many different languages, techniques and algorithms for their models. To exchange and conserve these models, one can use the Predictive Model Markup Language (PMML) [16], which is the leading standard for statistical and data mining models and supported by over 30 vendors and organizations. It is around for more than 20 years. With PMML, it is easy to develop a model on one system using one application and deploy the model on another system using another application, simply by transmitting the PMML file. PMML uses XML to represent mining models and the general structure is as follows (more details are given on the Data Mining Group web site, which contains many PMML sample files with reference to the underlying data sets):

```
<xs:element name="ExampleModel">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="MiningSchema"/>
        <xs:element ref="Output" minOccurs="0"/>
        <xs:element ref="ModelStats" minOccurs="0"/>
        <xs:element ref="Targets" minOccurs="0"/>
        <xs:element ref="LocalTransformations" minOccurs="0" />
        ...
        <xs:element ref="ModelVerification" minOccurs="0"/>
        <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
```

For most use cases in I-BiDaaS, we use Python as our programming language of choice to implement the required machine learning algorithms. For Python, one can use the Nyoka library [17] to export a large number of Machine Learning and Deep Learning models from popular Python frameworks into PMML. Nyoka supports the latest version of PMML, version 4.4. It is also used in the freely available ML Workbench [18], which allows to generate PMML files from scratch or from a dataset. In addition, the ML Workbench allows to test the PMML models locally on different data sets and also allows to execute Python code either stand-alone or in Jupyter notebooks. A screenshot of the Models view is shown in Figure 2.

Another free tool to generate PMML files is KNIME, which has a graphical editor where one can create workflows consisting of different nodes that are interconnected. A sample flow is presented in Figure 3, where the model was trained to predict the outcome of an industrial Etch process. The workflow is visualized by colored nodes where the colors indicate the category the node belongs to, e.g., yellow for string operations. Normally, the flow goes from left to right, indicated by the arrows along the lines. The three colored boxes under each node show the status of the node execution, green means successful execution, yellow means that this node was not yet executed and red would mean that an error occurred during execution, further details

are then given in the log file, e.g., that an input file is missing. The PMML file is written by the red node at the top right of the flow.

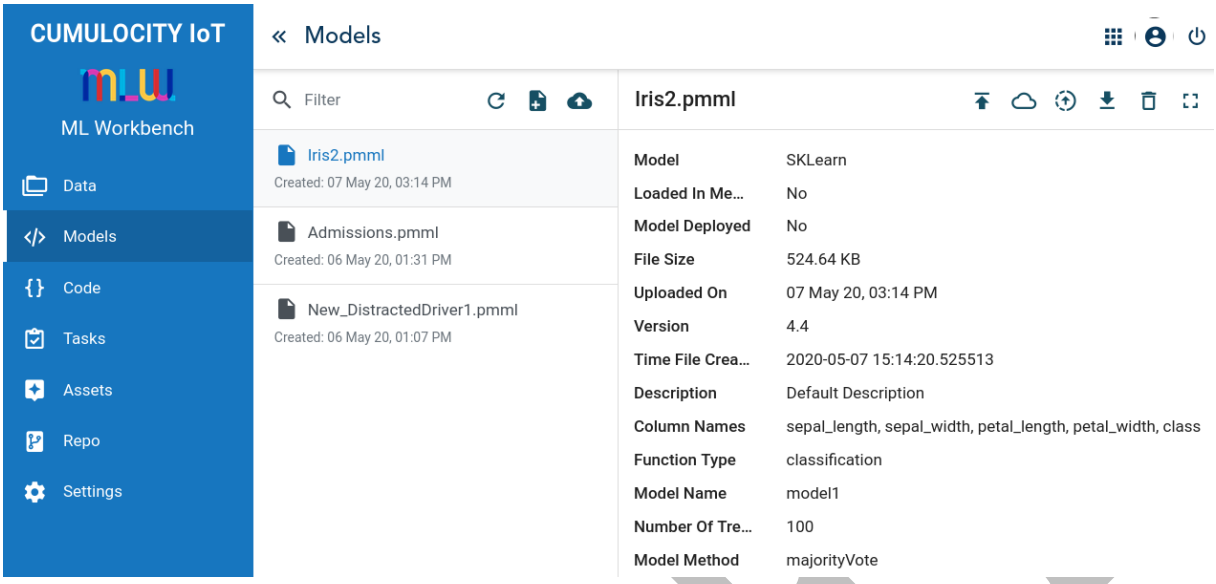

Figure 2: Models view of the ML Workbench

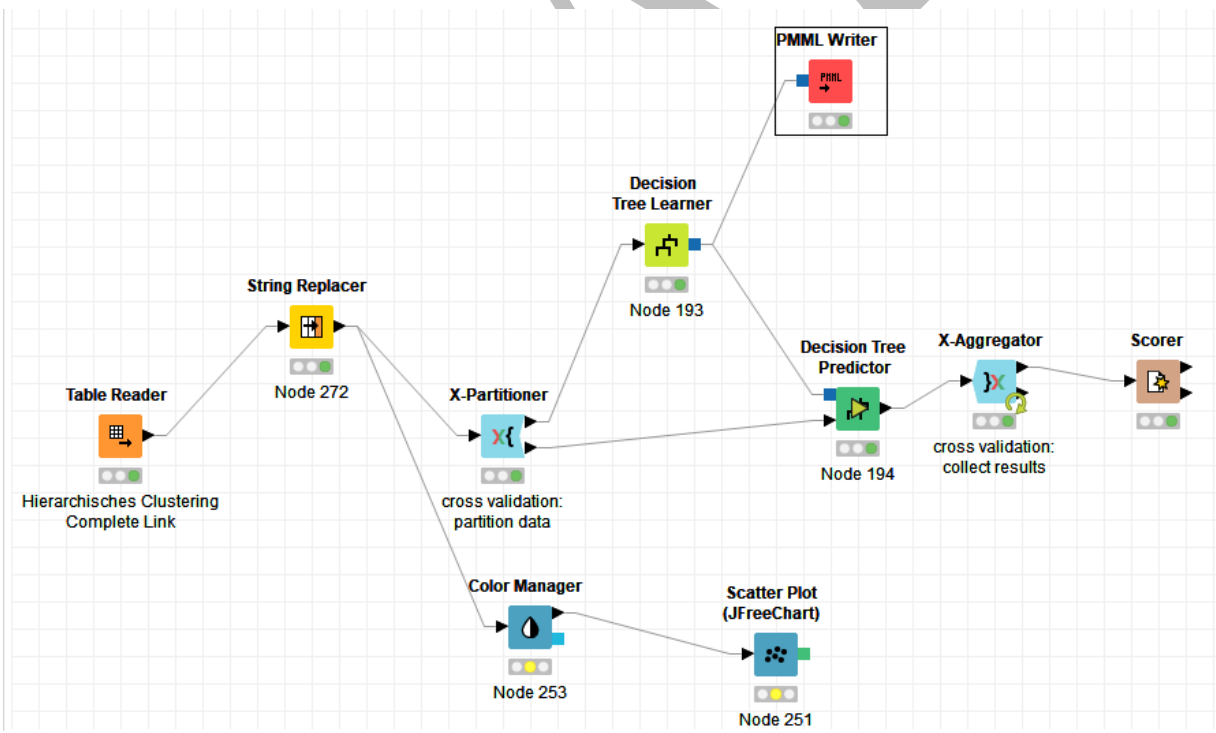

Figure 3: Sample KNIME workflow to generate PMML

We used KNIME already several times successfully, such as in the national research project BigPro [19]. KNIME can also read PMML files and uses this XML format internally.

So far, we do not have a use case in I-BiDaaS that was modelled in PMML. However, it is very likely that many of the planned future users (practitioners) are familiar with this standard and would like to use it in the I-BiDaaS platform. Therefore, we include in the I-BiDaaS platform the possibility to use PMML.

This is possible with Software AG's Zementis Predictive Analytics [20] solution, which can either run stand-alone as Zementis Server or as Plug-ins for different environments, e.g., Apama [21]. Zementis server and Zementis Plug-ins support PMML versions 2.0 through 4.3. They do that by incorporating a PMML converter, which is able to convert older versions of PMML to its latest. The converter is also tasked with the correction of known issues with automatically exported PMML code from different model building tools.

Zementis supports an extensive collection of statistical and data mining algorithms, such as:

- Anomaly Detection Models (Isolation Forest and One-Class SVM)
- Association Rules Models (Rectangular or Transactional format)
- Clustering Models (Distribution-Based, Center-Based, and 2-Step Clustering)
- Decision Trees (for classification and regression) together with multiple missing value handling strategies (Default Child, Last Prediction, Null Prediction, Weighted Confidence, Aggregate Nodes)
- Deep Neural Networks (MobileNet, VGGNet, ResNet, RetinaNet)
- K-Nearest Neighbors (for regression, classification and clustering)
- Naive Bayes Classifiers (with continuous or categorical inputs)
- Neural Networks (Back-Propagation, Radial-Basis Function, and Neural-Gas)
- Regression Models (Linear, Polynomial, and Logistic) and General Regression Models (General Linear, Ordinal Multinomial, Generalized Linear, Cox)
- Ruleset Models (Each rule contains a predicate and a predicted class value)
- Support Vector Machines (for regression and multi-class and binary classification)
- Scorecards (point allocation for categorical, continuous, and complex attributes as well as support for reasoncodes)
- Time Series Models (Univariate non-seasonal and seasonal ARIMA with `conditionalLeastSquares` as forecast method)
- Multiple models (model ensemble, segmentation, chaining, composition and cascade), including Random Forest Models and Stochastic Boosting Models

Zementis Server also implements the definition of a data dictionary, missing and invalid values handling, outlier treatment, as well as a myriad of functions for data pre- and post-processing, including: text mining, value mapping discretization, normalization, scaling, logical and arithmetic operators, conditional logic, built-in functions, business decisions and thresholds.

The Zementis predictive analytics plug-in for Apama is a correlator plug-in to score predictive models directly from within Apama applications. It also supports adding, updating, activating, and removing a PMML model at runtime. Finally, it can accept and process PMML data received from messaging services like Digital Event Services, MQTT, Universal Messaging and so on. Since we use Universal Messaging in the I-BiDaaS platform as message broker, this makes the integration easier. By default, it is installed in the I-BiDaaS distribution, but it remains optional to be installed and will work seamlessly, without any changes in I-BiDaaS.

4 Predicting the Performance of Call Centers

In this section, we present our I-BiDaaS-based solution for computing streaming analytics in call centers. This solution will be utilized by TID, as part of its "Quality of Service in Call Centers" use case, in order to improve the customer experience. As shown in Figure 4, our solution is able to process the recorded text transcripts, which can originate from different call centers at real-time. Given the vast amount of data that needs to be processed, I-BiDaaS is able to demonstrate faster end-to-end text analytics with GPU acceleration (more details can be found in D4.2 [7]), and provide different types of analytics (such as sentiment score, word frequencies, and most frequent words) for different time intervals (e.g., last minute, last hour, and last day). These analytics can be used for real-time predictions, such as QoS, and help a company to take appropriate actions in order to better understand or control a given situation. We note that our solution uses a dataset that has been captured from real call centers[1] to demonstrate the capabilities of the I-BiDaaS platform, however it can be used in many other different domains that require real-time processing of big volume of data, as well.

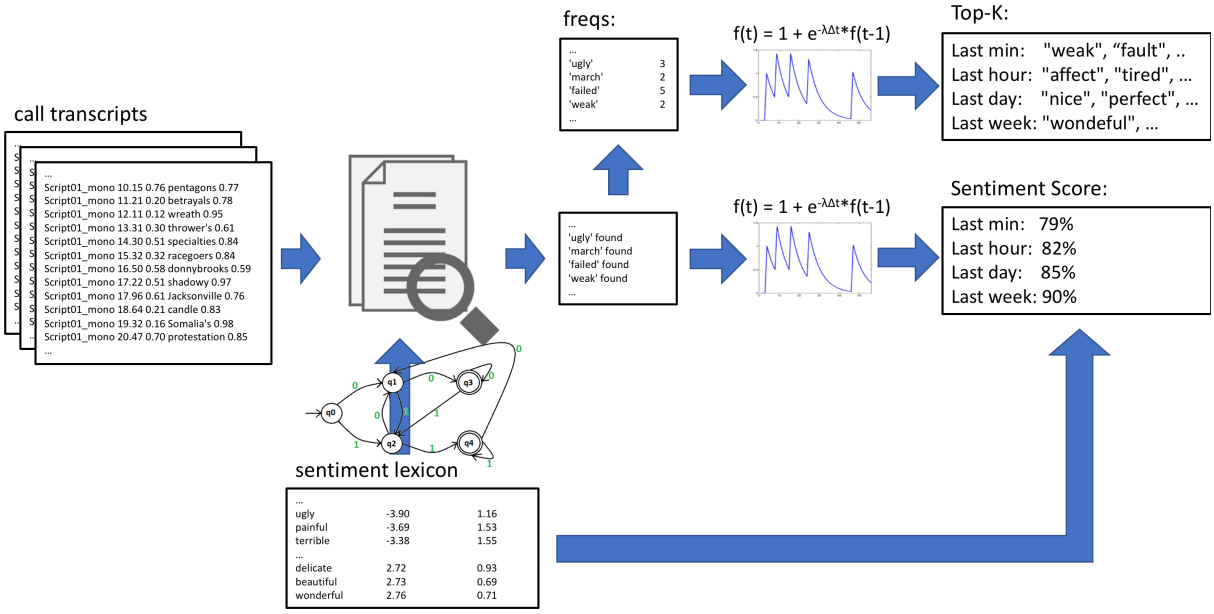

Figure 4: The diagram for processing the incoming call transcripts at real-time.

4.1 Word Searching

As shown in Figure 4, our analysis component takes as input the transcripts as they have been recorded from the corresponding call centers and a lexicon that contains a list of words and a marking if the word has a positive or a negative sentiment. In some cases, the words within the lexicon have a weighted score [22]. Such lexicons can be typically created using machine learning techniques or manually (e.g., by an expert).

The lexicon is used as an input by our event processing component shown in Figure 1, which extracts the corresponding words and constructs a deterministic finite automaton (DFA). This DFA is used to find each and every occurrence of them within the stream of transcripts that is received at real-time from the call centers. This processing is accelerated using GPUs, and will

[1] A detailed description of the Call Centers dataset can be found in D2.1 [20] and D8.2 [21].

produce as output a constant stream of matches (similar to [23], [24], [25]). These matches are then used for two purposes: (i) to compute a universal sentiment score per call center for each time window specified, and (ii) to compute the frequency of each word for each time window specified.

4.2 Rolling Analytics

To cope with the big amounts of data, many stream processing systems use the notion of rolling data analysis, in which the results are incrementally updated (rolled-over) as new data arrives [5] [6]. These systems, typically, maintain only the data that correspond to specific time-windows (e.g., last minute or last hour) and produce summary statistics only for these time intervals. The key idea is to collect a bunch of data for a specific time range, and compute the requested statistics for that data.

We note that the input events can be either synchronous or asynchronous: in the latter case, the events can arrive or stop arriving at any time, while in the former, the events are arriving periodically. When processing synchronous events, the time window is typically programmed as a special mechanism that periodically collects a number of tuples for the requested time window interval, which then is further processed in order to compute the desired statistics. In our solution, we follow a different approach. Each item in the stream processing pipeline is always coupled with a timestamp that is derived either directly from the raw data (e.g., if the data files contain a valid timestamp) or at the time it is received by our system.

Obviously, computing the desired statistics over a time-window is simple, but it can lead to huge memory requirements when the incoming data increases or when it is highly dispersed. As such, they can operate only on small time intervals (e.g., up to a couple of minutes), due to the excessive memory requirements. This is a major constraint for many applications that need to detect long-term behaviors (e.g., events that occur periodically in an hour- or even in a day-granularity). To overcome the high memory requirements, there have been many different methods and data-structures proposed [26] that can be used to compute various types of statistics, which do also provide very efficient memory requirements guarantees, at a cost of precision. For instance, as we explain in more detail below, the majority of these approaches are probabilistic, hence the provided statistics are approximate. Obviously, this results in approximation errors in the statistics themselves, although they do not significantly affect the accuracy of the query results. The reason is that exactness is not necessary for the types of queries that we process at real-time, which are related in detecting unusual item trends in the stream or finding the ranking of the top most active items in the stream. Put shortly, the role of the real-time view is to monitor interesting behaviors and provide a more fine-grained dataset for the offline batch processing phase.

To overcome the large memory consumptions of rolling windows, we use the notion of time-decayed counters [27]. The idea behind this is that the value of a counter decays automatically over time using a mathematical formula. Exponential decay is one such popular formula due in part to the relative simplicity with which simple counters can incorporate exponential decay. In particular, it allows to continually reduce, to half, the value of the counter over time. As such, we only keep a simple counter for each feature that is incremented every time we have a new occurrence, and is also decayed using the following formula:

$$cnt = e^{-a \cdot dt} \cdot cnt, \text{ where } e = 2.718,\ a = \frac{\log(2)}{halflife}, \text{ and } dt = t_{current} - t_{previous} \quad (1)$$

By using this formula, there is no need for manually sliding time windows or explicitly maintaining rolling counts, as observations naturally decay away over time. The *halflife* parameter defines the time required for the counter to reduce to half of its initial value. In addition, it is designed for heavy writes and sparse reads, as it implements decay only when necessary or at read time.

Obviously, the counters are not exactly the same as if we have used fixed time windows, even though are sufficient to exhibit the feature trends and reveal momentums over time, without requiring to keep large amounts of data (for each specific time frame), but only a few numbers per feature. In addition, the time-decayed formula provides lifetime flexibility by adjusting the lifetime accordingly, in an ad-hoc manner: we can track trends over just a few minutes or hours, as opposed to more long-term events, in which we are interested in trends over weeks and months.

4.2.1 Sentiment Score Computation

In our scenario, we are not interested to compute the sentiment of a certain entity within a transcript, but rather predict the sentiment out of a whole transcript, which will then be further aggregated to predict the sentiment of the whole call center. As such, the sentiment values of the words that have been found in the text stream of each call center are accumulated into a single score using the mathematical formula described in Section 4.2. This score is actually the sentiment score of a specific call center for a given time window, and is actually an indicator of whether the overall customer sentiment is positive or negative.

To keep the score, we use a separate exponential-decayed counter for each of the specified time windows. This score is increased by one when a positive word appears or decreases by one if a negative word appears. In case we use a lexicon with different weights per word, we add the corresponding weight each time. We wanted to keep it as simple as it could be, but on the other hand, if a user has his own dataset with a sentiment score for each word, he/she can easily use them to increase/decrease the score accordingly. We would like to add that the score can be also negative, and this is also realistic since a specific call center might face a huge amount of complaints for many reasons like long time waiting in line to be served. Separate scores might take place since more than one call center will be available in the database along with a percentage score to ideally visualize how many positive words occurred in this time window. Once the aggregation finishes, we move to the final step, which is to extract the Top-K words occurring from the Hashmap aggregation (described in the next Section).

4.2.2 Most Frequent Words

Besides the total sentiment score for each call center, we also compute the Top-K emotion words for each of the selected time windows. These words can be very helpful, as they give extra context and more details regarding the sentiment score.

To aggregate the occurrences of each of the words found, we use a Hashmap. The size of the Hashmap in the worst-case scenario will be as large as the whole lexicon that was fed inside the pattern file. Each entry in the Hashmap is of the form `<word_id, counter>`, where `word_id` is the unique identification of the word and `counter` is the corresponding count for this word for a specific time-window, as described in Section 4.1. The use of exponential-decayed counters allows the user to configure appropriate time-windows as desired (e.g., for the last minute, hour, day or even week). By doing so, our approach is capable to monitor the frequencies of each word occurring while processing the incoming transcripts, and increase the

frequencies (counters) of each word along with a decay function to decrease these counters on different time windows.

Every time a specific word appears and it is not present in the Hashmap, a new entry in the Hashmap is created, with the current epoch timestamp (in milliseconds) and a count value equal to one. If the word exists in the Hashmap, then we only increase its counter by one or as many times as the word has been found and update its corresponding timestamp. This counter helps us to keep track of each word's frequency occurrence.

The aggregated results are published periodically, every few seconds (configurable). To do that, we first need to decay the counters based on the current epoch. This step is necessary because word occurrences change asynchronously within different time-windows. In case a specific word has stopped appearing, its corresponding counter will eventually approach to zero and will be removed from the Hashmap when exceeding a certain threshold (e.g., 0.00001). After the counters have been decayed, we sort them by their corresponding frequencies to find the Top-K entries.

Data : {"CallCenterID":"CallCenter2","TopKWords":[{"beckoned":"0.664626934984473"},{"accusations":"0.3708308909212282"},{"affirmative":"0.2070737617272439"},{"calming":"0.11565637273772882"},{"abort":"0.06455324068304688"},{"absurdly":"0.03603018825545449"},{"aggrieve":"0.020123150461091005"},{"afordable":"0.011232321924475175"},{"arduously":"0.006265158252880099"},{"apprehension":"0.003495908967612896"}],"SentimentScore":7,"TimeWindow":60,"Epoc":1588941163951}
Consumed event ID :29711 with TAG : TAG1
Data : {"CallCenterID":"CallCenter2","TopKWords":[{"abounds":"1.416308866517227"},{"abuses":"1.3282353957503392"},{"calming":"1.2712364364155389"},{"awesome":"1.216273291811153"},{"beauty":"1.2140179115853666"},{"adaptive":"1.1267476791481033"},{"asperse":"1.0982931805320764"},{"arduously":"1.088288070770989"},{"afordable":"1.0848454464556776"},{"beautifully":"1.0609678892609855"}],"SentimentScore":7,"TimeWindow":3600,"Epoc":1588941163951}
Consumed event ID :29712 with TAG : TAG2
Data : {"CallCenterID":"CallCenter2","TopKWords":[{"abounds":"1.416308866517227"},{"abuses":"1.3282353957503392"},{"calming":"1.2712364364155389"},{"awesome":"1.216273291811153"},{"beauty":"1.2140179115853666"},{"adaptive":"1.1267476791481033"},{"asperse":"1.0982931805320764"},{"arduously":"1.088288070770989"},{"afordable":"1.0848454464556776"},{"beautifully":"1.0609678892609855"}],"SentimentScore":7,"TimeWindow":86400,"Epoc":1588941163951}
Consumed event ID :29713 with TAG : TAG0
Data : {"CallCenterID":"CallCenter1","TopKWords":[{"assure":"3.026154375617 47"},{"adjustable":"2.748603237380398"},{"abrupt":"2.7213118988176594"},{"beckoning":"2.589989786799762"},{"adoring":"2.529729581645662"},{"audacious":"2.4991035178104233"},{"amiss":"2.4144481501177304"},{"aches":"2.3043348009582 14"},{"aground":"2.2853696610468495"},{"bravery":"2.26138025924143"}],"SentimentScore":-222,"TimeWindow":60,"Epoc":1588941163951}
Consumed event ID :29714 with TAG : TAG1
Data : {"CallCenterID":"CallCenter1","TopKWords":[{"apocalypse":"65.85283450483358"},{"antagonism":"65.1201861225178"},{"annihilation":"64.96158265959102"},{"astutely":"64.62897407150372"},{"attune":"64.45829980227835"},{"antiproliferation":"64.27254629330105"},{"beauteous":"64.2600397640414"},{"awesome":"64.12597883390407"},{"aimless":"63.28399382410632"},{"attractively":"62.962388422047475"}],"SentimentScore":-222,"TimeWindow":3600,"Epoc":1588941163951}
Consumed event ID :29715 with TAG : TAG2
Data : {"CallCenterID":"CallCenter1","TopKWords":[{"apocalypse":"65.85283450483358"},{"antagonism":"65.1201861225178"},{"annihilation":"64.96158265959102"},{"astutely":"64.62897407150372"},{"attune":"64.45829980227835"},{"antiproliferation":"64.27254629330105"},{"beauteous":"64.2600397640414"},{"awesome":"64.12597883390407"},{"aimless":"63.28399382410632"},{"attractively":"62.962388422047475"}],"SentimentScore":-222,"TimeWindow":86400,"Epoc":1588941163951}
Consumed event ID :29716 with TAG : TAG0
Data : {"CallCenterID":"CallCenter3","TopKWords":[{"abomination":"0.664626934984473"},{"aggressor":"0.3708308909212282"},{"awesomeness":"0.2070737617272439"},{"afflictive":"0.11565637273772882"},{"assurances":"0.06455324068304688"},{"appall":"0.03603018825545449"},{"beautify":"0.020123150461091005"},{"brave":"0.011232321924475175"},{"amusingly":"0.006265158252880099"},{"awe":"0.003495908967612896"}],"SentimentScore":-7,"TimeWindow":60,"Epoc":1588941163951}
Consumed event ID :29717 with TAG : TAG1
Data : {"CallCenterID":"CallCenter3","TopKWords":[{"accidental":"1.8192481383446144"},{"baffling":"1.7465472329583052"},{"audaciousness":"1.7254176367225307"},{"benifits":"1.563918214932667"},{"agonizing":"1.4990672685792"},{"agony":"1.479905498496719"},{"afflictive":"1.3334005664525994"},{"baffled":"1.320156055347466"},{"aggrieved":"1.263132887761136"},{"brainiest":"1.1015723617241289"}],"SentimentScore":-7,"TimeWindow":3600,"Epoc":1588941163951}
Consumed event ID :29718 with TAG : TAG2
Data : {"CallCenterID":"CallCenter3","TopKWords":[{"accidental":"1.8192481383446144"},{"baffling":"1.7465472329583052"},{"audaciousness":"1.7254176367225307"},{"benifits":"1.563918214932667"},{"agonizing":"1.4990672685792"},{"agony":"1.479905498496719"},{"afflictive":"1.3334005664525994"},{"baffled":"1.320156055347466"},{"aggrieved":"1.263132887761136"},{"brainiest":"1.1015723617241289"}],"SentimentScore":-7,"TimeWindow":86400,"Epoc":1588941163951}

Figure 5: The aggregated results for different time windows.

4.3 Data Model

The aggregated analytics are produced periodically, every 5-10 seconds (configurable), using JSON format. The structure is depicted in Figure 5 and, as we can see, we use basic types to represent a single value (e.g., integers, floating point, epoch times, strings), and arrays for bundled values (e.g., sets and tables). In particular, each tuple is of the form:

```
{ccid, epoch, time window id, sentiment score, top K words}
```

where `ccid` is the call center id where the entries came from, `epoch` the timestamp of our processing stage, `time window id` is which window was processed, the total `sentiment score` of the specific time window along with the `top-k words` occurred.

The use of JSON allows post-processing modules to easily interpret the produced results using arbitrary schemes for different purposes. Furthermore, each JSON message can be easily enriched with contextual data. For instance, it can be extended with other data, or tagged by certain modules in the processing pipeline.

4.4 Performance Evaluation

In this section, we show the performance of our approach. For the evaluation, we use synthetic data that we feed into Terracotta [28], in a streaming fashion. The synthetic data are generated from a set of 500 English words, 50% positive and 50% negative, randomly distributed to represent transcripts from three different call centers.

Figure 6 shows the end-to-end time required (median value) of our approach over different numbers of transcripts. The time increases proportionally to the volume of data, most of which is spent on fetching the data from Terracotta. The exact percentage of each of the operations performed during our analysis is presented in Figure 7. As we can see, reading the data from the database consumes a significant amount of the total time (over 90%), which we plan to optimize as part of our future work.

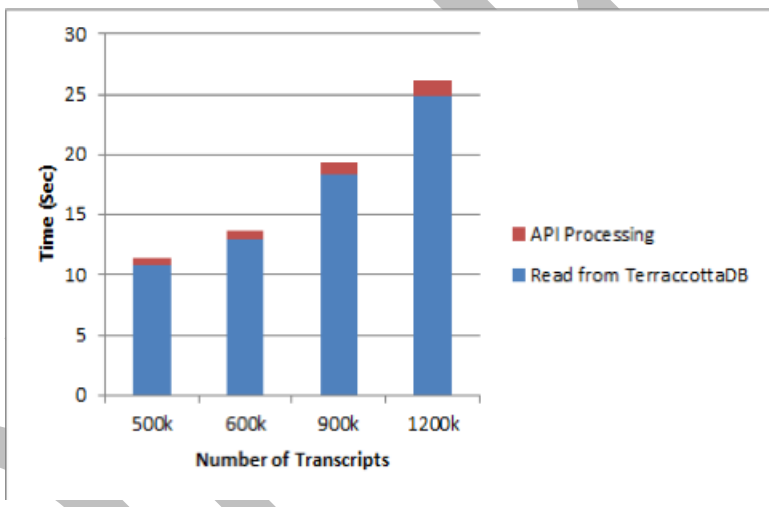

Figure 6: Sustained end-to-end time for processing call transcripts

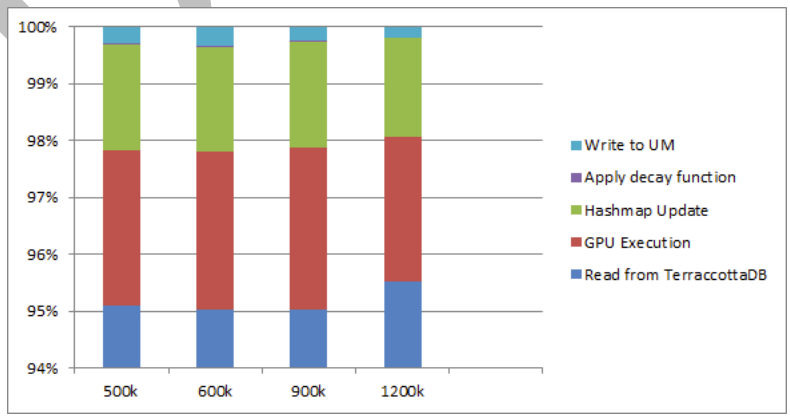

Figure 7: Breakdown of each of the operations performed during processing call transcripts

5 Visualizing streaming analytics

Based on the work conducted during the development of the MVP (see D4.1 [29]) and the 1st integrated prototype, visualisations of streaming analytics provide a robust mechanism of receiving a constant flow of analytics results and displaying them to users via real-time auto-updated visual elements. The implementation of this mechanism relies on the Universal Messaging JavaScript API [30]. This API is a pure JavaScript solution that allows developers to use JavaScript and HTML-based clients, which can publish and subscribe to Universal Messaging channels, and asynchronously receive events in real-time. The communication takes place via a set of JavaScript communication drivers and protocols using streaming techniques or long polling according to requirements and browser capabilities. An indicative list of the first drivers that a JavaScript client application will try using follows:

- `WEBSOCKET`: Streaming driver for browsers supporting HTML5 Web Sockets.
- `XHR_STREAMING_CORS`: Streaming driver for browsers supporting `XMLHTTPRequest` with CORS (Cross-Origin Resource Sharing). Intended for Chrome, Firefox, Safari, IE10+ and Microsoft Edge.
- `XDR_STREAMING`: Streaming driver for browsers supporting `XDomainRequest` (Microsoft Internet Explorer 8+). Intended for IE8 and IE9.

The list includes subsequent options of other drivers [28] to use, but the vast majority of clients settle on one of the three mentioned above.

The I-BiDaaS visualisations [32], being developed as JavaScript-based web applications (built on Angular.io [33]), take advantage of the UM JavaScript API to consume data output from the streaming analytics functions. Using the `subscribe()` method of the API, the visualisations get connected to the relevant UM channels and listen for new results generated by the analysis modules. Received data is afterwards fed to the various visualization elements, which are automatically updated in real-time, thus allowing users to view updates without having to refresh their browser page or making any other interaction. Figure 8 depicts the communication flow. This functionality allows the monitoring of analytics in a time-based manner, which is particularly exploited in use cases, such as the prediction of call centers performance that is described in Section 4.

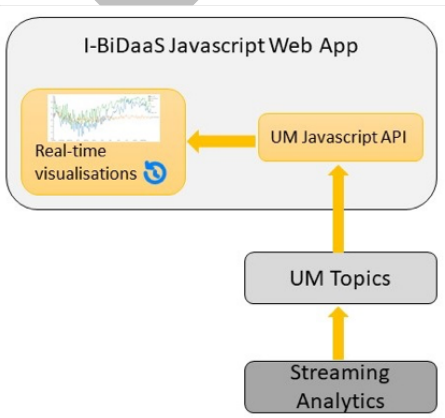

Figure 8: The interconnection of real-time visualisations via Universal Messaging (UM)

More specifically, the visualisations, in this case, are able to listen to data coming from the UM and display a number of real-time updated visualisations to show analysis of data in various time windows, namely 1-min, 1-hour and 1-day periods. The visualisations are constantly updated with newly received data and the displayed time window shifts as time progresses so as to always show only the latest data. Figure 9 and Figure 10 show the capture of two moments in time, namely *t1* and *t2*, for the visualisations of the call centers case.

Figure 9: Sentiment Averages for time *t1*

Figure 10: Sentiment Averages for time *t2*

Apart from the sliding time windows on the lower part of the screens, an interactive map showing aggregated results per call center is available on the top left part of the screen and, besides this, an array of the latest most important words extracted by the analysis is also presented in a time-specific separation following the same intervals of 1 min, 1 hour and 1 day.

The result is that end users only have to access the page, and real-time information is presented to them without any other action by their side. Switching from one call center to another is the only interaction available, as it was considered the best solution for having a concise interface which can, at the same time, support multiple call center information at one page.

6 Conclusions

In this deliverable, we showed several aspects of the streaming analytics components of I-BiDaaS, as well as the different techniques we have implemented to provide flexibility and acceleration. Overall, I-BiDaaS is capable to process incoming streams at real-time and provide aggregate results that can optionally be processed further by external analysis modules or visualization frameworks. It also provides interconnection primitives between the batch and stream processing layers. I-BiDaaS is compatible with PMML, hence it enables the rapid specification and composition of new processing functionalities, allowing the developers to focus solely on their application logic.

7 Bibliography

[1] C. Mitch, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing and S. . B. Zdonik, "Scalable Distributed Stream Processing," *CIDR,* vol. 3, pp. 257-268, 2003.

[2] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan and S. Zdonik, "Aurora: a data stream management system," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007.

[5] B. Arian, A. Finamore, P. Casas, L. Golab and M. Mellia, "Large-scale network traffic monitoring with DBStream, a system for rolling big data analysis," in *Proceedings of the 2014 IEEE International Conference on Big Data (Big Data)*, 2014.

[6] P. Bhatotia, U. A. Acar, F. P. Junqueira and R. Rodrigues, "Slider: incremental sliding window analytics," in *Proceedings of the 15th International Middleware Conference*, 2014.

[7] I-BiDaaS Consortium, "D4.2: Distributed event-processing engine," http://www.ibidaas.eu/sites/default/files/docs/ibidaas-d4.2.pdf.

[8] I-BiDaaS Consortium, "D2.1: Data assets and formats".

[9] I-BiDaaS Consortium, "D8.2: Data Management Plan".

[10] "Hecuba," [Online]. Available: https://github.com/bsc-dd/hecuba.

[11] "Apache Cassandra," [Online]. Available: https://cassandra.apache.org.

[12] "paho-mqtt," [Online]. Available: https://pypi.org/project/paho-mqtt/.

[13] "Eclipse Mosquitto: An open source MQTT broker," [Online]. Available: https://mosquitto.org.

[14] "pydantic," [Online]. Available: https://pydantic-docs.helpmanual.io.

[15] "KNIME: End to End Data Science," [Online]. Available: https://www.knime.com/.

[16] "Data Mining Group," [Online]. Available: http://dmg.org/.

[17] "NYOKA," [Online]. Available: https://github.com/nyoka-pmml/nyoka.

[18] "ML Workbench," [Online]. Available: https://github.com/SoftwareAG/MLW.

[19] "BigPro," [Online]. Available: https://www.fir.rwth-aachen.de/en/research/research-work-at-fir/detail/bigpro-01is14011/.

[20] "Zementis Predictive Analytics," [Online]. Available: https://www.softwareag.com/br/products/az/zementis/default.

[21] "Apama Community Edition," [Online]. Available: http://www.apamacommunity.com.

[22] W. L. Hamilton, K. Clark, J. Leskovec and D. Jurafsky, "SocialSent: Domain-Specific Sentiment Lexicons for Computational Social Science.," [Online]. Available: https://nlp.stanford.edu/projects/socialsent/.

[23] G. Vasiliadis, M. Polychronakis and S. Ioannidis, "Parallelization and characterization of pattern matching using GPUs," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, 2011.

[24] E. Papadogiannaki, L. Koromilas, G. Vasiliadis and S. Ioannidis, "Efficient software packet processing on heterogeneous and asymmetric hardware architectures," *IEEE/ACM Transactions on Networking,* vol. 25, no. 3, pp. 1593-1606 , 2017.

[25] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. Markatos and S. Ioannidis, "Regular expression matching on graphics hardware for intrusion detection," in *Recent Advances in Intrusion Detection*, 2009.

[26] M. Gurmeet Singh and R. Motwani, "Approximate frequency counts over data streams," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, 2002.

[27] G. Cormode, F. Korn and S. Tirthapura, "Exponentially Decayed Aggregates on Data Streams," in *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE)*, 2008.

[28] "Terracotta: Translytical platform for in-memory data," [Online]. Available: http://www.terracotta.org.

[29] I-BiDaaS Consortium, "Real time complex event processing engine - design and approach".

[30] "Universal Messaging JavaScript API," [Online]. Available: https://documentation.softwareag.com/onlinehelp/Rohan/num10-3/10-3_UM_webhelp/um-webhelp/Doc/JavaScript/files/nirvana-js.html.

[31] S. AG, "Universal Messaging Developer Guide Version 10.2 (Innovation Release)," https://documentation.softwareag.com/onlinehelp/Rohan/num10-2/10-2_Universal_Messaging_Developer_Guide.pdf, 2018.

[32] I-BiDaaS Consortium, "D5.4: Big-Data-as-a-Self-Service Test and Integration Report v2".

[33] "Angular: One framework. Mobile & desktop.," [Online]. Available: https://angular.io.

[34] "ML Workbench," [Online]. Available: https://github.com/SoftwareAG/MLW.

[35] "DATA MINING GROUP (DMG)," [Online]. Available: http://dmg.org/.

[36] "Zementis Predictive Analytics," [Online]. Available: https://www.softwareag.com/br/products/az/zementis/default.