EVOTION

727521 – EVOTION

DELIVERABLE No: D5.5

Big Data Analytics Engine

Author(s): Marco ANISETTI, Valerio BELLANDI, Marco CREMONINI, Ernesto DAMIANI (UNIMI), Nikos DIMAKOPOULOS, Panagiotis KOKKINAKIS (ATC), Pascal PAPAGRIGORIOU, Michail SMYRLIS (EMP), Niels Henrik PONTOPPIDAN (OTC),

Dissemination level	
PU	PU - Public

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 727521

Project acronym and GA no	EVOTION- 727521
Project full Title	EVidenced based management of hearing impairments: Public health pOlicy making based on fusing big data analytics and simulaTION.
Project Type	RIA
Start date – end date	01.11.16 – 31.10.19
Website	www.h2020evotion.eu
Deliverable type:	DEM+R
Delivery date	M18 (30-04-2018)
Authors/contributors:	Marco ANISETTI, Valerio BELLANDI, Marco CREMONINI, Ernesto DAMIANI (UNIMI), Nikos DIMAKOPOULOS, Panagiotis KOKKINAKIS (ATC), Pascal PAPAGRIGORIOU, Michail SMYRLIS (EMP) and Niels Henrik PONTOPPIDAN (OTC)
Reviewers:	George Spanoudakis (CITY), Niels Henrik PONTOPPIDAN (OTC)
Contact:	Marco Anisetti (*marco.anisetti@unimi.it)*
To be cited as:	**Anisetti et al. (2018), Big Data Analytics Engine , Deliverable D5.5 to the EVOTION-727521 Project funded by the European Union, UNIMI, Italy**
Subject and keywords	EVOTION Big data Engine, workflows, analytics, security/privacy
Disclaimer:	This document's contents are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

Table of Contents

Table of Figures

List of Tables

Executive Summary

This document provides a report of the Big Data Analytic (BDA) engine of the EVOTION platform, the subcomponents composing the system, the mechanisms that it provides to enable a tight integration with the EVOTION data repository by exploiting native technology, and an integration based on application interfaces with other EVOTION subsystems.

The overall design of the BDA engine was informed by an architectural model called Lambda Architecture, which consists of three layers, respectively for processing batch data, for computing incremental data updates, and for aggregating views on data. This architectural model satisfies the requirements of the EVOTION BDA and provides the flexibility needed to accommodate the heterogeneity of runtime environments. Relevant design concepts are described in this document, like the notions of Workflow Catalogue and Task Catalogue, with corresponding entities called Data Analytic Workflow (DAW) and Executable Data Analytic Workflow (EDAW), which are at the core of the overall engine design and management of EVOTION data analytic processing.

The BDA implementation is described in terms of technology stack adopted from the open source Apache family of solutions for big data processing, and as specific solutions realized, for instance to process EDAWs, to manage Workflow and Task catalogues, to interact with external applications like the Ontology Reasoner or the DSS. In addition, the EVOTION BDA Engine is also structured to be compatible with the H2020 TOREADOR Methodology for fast roll-out of big data analytics, and the EVOTION BDA engine is capable of executing TOREADOR based analytics for fast prototyping and extension of EVOTION workflows.

Specific technologies are presented in their key characteristics and discussed with respect to features and limitations. Where possible, alternatives have been presented with the rationale that guided our final selection. A section with some use cases serves the purpose of presenting EVOTION scenarios where the BDA engine is processing data and producing results. Some examples of implemented Tasks and Executable Workflows are also presented in the appendix A. These examples are mainly to show some of the characteristic of the BDA Task and Workflows in relation to EVOTION scenarios.

A section specifically devoted to highlight security and privacy issues has been produced. First a threat model for the BDA engine is presented, compliant with the overall threat classification adopted in EVOTION. Then, security objectives and functionalities have been specified to match the selected threats.

Finally, we wish to specify that the EVOTION BDA is structured to be compatible with the H2020 TOREADOR Methodology for fast roll-out of big data analytics. More specifically, the EVOTION analytic workflows execution can be triggered with the TOREADOR methodology given the transformation between the EVOTION Language and Models in WP3 and WP4 to the TOREADOR workflows. This transformation will be implemented as part of the WP4 Transformation tool.

1. Component overview

The Big Data Analytic Engine (in the following, *BDA*), is a central component of the EVOTION platform, mainly focused on the execution of policy model's instances over the data collected during the EVOTION clinical trial and aimed at supporting policy makers in the definition of Public Health Policies (PHP). The execution of policy model instances is based on the definition of Data Analytic Workflows (*DAWs*) according to the PHP language developed in WP4 and PHP models of WP3.

A DAW is an ordered sequence of Data Analytic Tasks (just Tasks in the following) of the following types:

- *Data Processing task:* focused principally on data preparation like data source selection for feature reduction, data cleaning, or data type transformation.
- *Statistical Analysis task:* focused on performing statistical analysis on a given dataset like ANOVA, Breusch-Pagan Test, etc.
- *Data Mining task:* focused on more elaborated analysis (e.g., clustering, machine learning) involving supervised or unsupervised algorithms like Random Forest, K-means, etc.

The output of one DAW could be the input of a consecutive one. For instance, one DAW can implement a PCA to select features, and the consecutive one can use the output to compute classification.

The peculiarity of DAWs in EVOTION is that they could mix automatic and human actions. Stakeholders, like policy makers or clinicians, could be part of a workflow when, supported by the EVOTION DSS, could request and specify consecutive refined analysis.

A DAW is expressed as a detailed procedural workflow and requires to be translated into executable form for the BDA Engine (Executable DAW, in the following, *EDAW*). This transformation is carried out by the PHPDM Transformation tool that will be developed in Task T4.3. The BDA Engine presented in this deliverable is compatible with H2020 Toreador[1] workflows (Damiani at al., 2017). H2020 TOREADOR is one of the biggest Big Data project funded by EU focused on providing Model-based Big Data Analytics as a Service. This compatibility allows to use TOREADOR methodology for the fast prototyping and deployment of analytics on the EVOTION platform.

In this section, we summarize the EVOTION platform architecture emphasizing the role of the BDA Engine component. We first briefly recap the set of requirements driving its definition and development starting from those defined in deliverable D2.1 (Dimakopoulos et al., 2017). We then introduce the architecture presented in deliverable D2.2 (Ye et al., 2017) focusing on the components strictly related to the BDA Engine. We finally present an overview of the BDA internal architecture and interfaces.

1.1 Big Data Analytic Engine requirements

Deliverable D2.1 presented a list of requirements elicited from the EVOTION scenarios. They are generically related to the EVOTION platform as a whole or to the Mobile App. A list of preliminary relevant functional requirements for the BDA was also presented in D2.1. We recap the ones related, directly or indirectly, to the BDA in the following Table 1.

[1] http://www.toreador-project.eu/

Table 1: List of EVOTION functional requirements relevant for the BDA.

Functional req. ID	Description	Priority of accomplishment
FR(PHAS)2	Discover factors of low HA usage	Must have
FR(PHAS)6	Characterize data to define the size of the dataset	Should have
FR(PHAS)7	Support different types of data analysis	Should have
FR(PHAS)8	Support different types of data tests	Should have
FR(PHAS)9	Produce and manage metrics for quality of analysis	Could have
FR(PHAS)10	Initiate data analysis session	Must have
FR(PHAS)11	Administrate (create, update, delete) analysis' outcomes	Must have
FR(PHAS)12	Notification of analysis completion	Must have
FR(PHAS)14	Suggest factors of analysis' outcome	Must have
FR(PHAS)15	Re-analysing a specific dataset with different factors	Must have
FR(PHAS)16	Data analysis, in a statistical way, between different data types	Should have
FR(PHAS)17	Support multiple types of analysis' criteria	Must have
FR(PHAS)18	Support of progressive notifications and storing of outcomes on data analysis	Should have
FR(PHAS)24	Identification of the resulting tense and generation of a potential policy model for implementation	Should have
FR(PHAS)26	Stop the relevant analytic activity	Must have
FR(CLIS)36	Correlation of HA usage with collected noise data	Should have
FR(CLIS)37	Manage HA usage with problems occurred, ratings provided and corresponding noise recorded	Could have
FR(CLIS)38	Different visualization modes of recorded data	Should have
FR(CLIS)42	Analyse END or BHD parameters and automatically respond to them by changing the fitting profile	Must have
FR(CLIS)48	Manage and visualize a detected event record	Should have
FR(CLIS)59	Visualize TTS/NIHL data recorded for a selected patient	Must have
FR(CLIS)60	Provide aggregated records of TTS episodes from patients from retrospective studies	Must have
FR(CLIS)61	Analyse data and suggest combination of factors affecting TTS and NIHL episodes	Must have
FR(CLIS)75	EVOTION platform deployed on the cloud	Should have
FR(CLIS)77	Visualize aggregated data sets	Must have
FR(CLIS)78	Visualize HA usage data with respect to various noise parameters	Must have

FR(CLIS)82	Analyse the responses to the auditory training tests	Must have
FR(CLIS)95	Analyse captured events from the patient's devices in relation to the patient responses in the questionnaires	Could have
FR(CLIS)100	Compare sensors' data and data collected from HAs	Must have
FR(CLIS)103	Analyse sensors' and HAs' data	Must have
FR(PSOS)139	Determine combination of factors (noise levels, duration of exposure, other physiological data) associated with TTS/NIHL episodes	Must have
FR(PSOS)144	Analyse issues, concerns and problems with hearing aid reported by HA users	Must have

In addition and related to the above generic scenarios-elicited requirements, in Table 2 we also define BDA specific requirements as follows.

Table 2: List of new functional requirements specifically defined for the BDA.

Functional req. ID	Description	Priority of accomplishment
FR(BDA)1	Batch processing capabilities	Must have
FR(BDA)2	Micro batch processing	Could have
FR(BDA)3	Provide implementation for Processing, Statistical and Data Mining tasks needed for the scenarios	Must have
FR(BDA)4	Direct connection to Data storage system (EVOTION Data Repository)	Must have
FR(BDA)5	Cataloque of available EDAW (obtained after the PHPDM Transformation)	Should have
FR(BDA)6	Workflow orchestrator	Should have
FR(BDA)7	Tracking of Workflow status, including intermidiate tasks.	Must have
FR(BDA)8	Catalogue of available tasks that can be used to compose a workflow	Must have
FR(BDA)9	Ability to execute scheduled workflows	Could have
FR(BDA)10	Predefined EDAWS for supporting scenarios for Clinical and Patients like the ones requiring FR(CLIS)36, FR(CLIS)42.	Must have
FR(BDA)11	Provide administrative Dashboard to deploy and set up tasks and workflows	Should have
FR(BDA)12	Provide interfaces to the rest of EVOTION architecture to support all the requirements in an integrated fashion	Must have

The above generic requirements as well as the ones elicited from the scenarios drive the definition of the EVOTION BDA.

1.2 Overall EVOTION Architecture: The role of the Big Data Analytic Engine

Let us consider a subsystem of the EVOTION Architecture defined in deliverable D2.2 and depicted in Figure 1. This subsystem contains the BDA and the components cooperating with the BDA for the definition and the execution of a given workflow. They are defined in deliverable D2.2 as follows:

- **BDA engine:** it mainly addresses the functionalities required for processing EDAWs and providing/storing execution results.
- **Data Repository**: it provides the storing facilities for EVOTION data, either retrospective data, or new data collected during the clinical trial. It directly interacts with the BDA Engine allowing the execution of analytic tasks.
- **Ontology Reasoner:** It is responsible for finding a semantic structure in the ontology representing a PHPDM model. It provides an interface to query the ontology and allows other components to update or create an ontology.
- **Decision Support System:** The purpose of the Decision Support System (DSS) is to provide data retrieval and summarization functionalities for text-mining related tasks, aimed mainly at PHPDM makers and clinicians to define and produce decision-related scenarios, based on information produced by the EVOTION platform and external sources as well.
- **PHPDM Transformation tool:** It takes a policy model instance defined by the PHPDM Specification tool and transforms it into a procedural set of commands that the BDA Engine should execute.
- **Mobile App:** It is used in the data gathering process of the EVOTION platform. It is the tool that collect information from the EVOTION HAs and biosensors, the environment and directly from patients. It is the main way of interaction with EVOTION patients enrolled in the clinical trial by getting input from them as well as serving them with relevant information.
- **Front-End Dashboard:** The purpose of this module is to provide a user-friendly visualization interface for policy makers and clinicians and allow them interacting with the EVOTION platform.
- **Security Manager:** It provides security features to the EVOTION platform like authentication/authorization, encryption to name but a few.

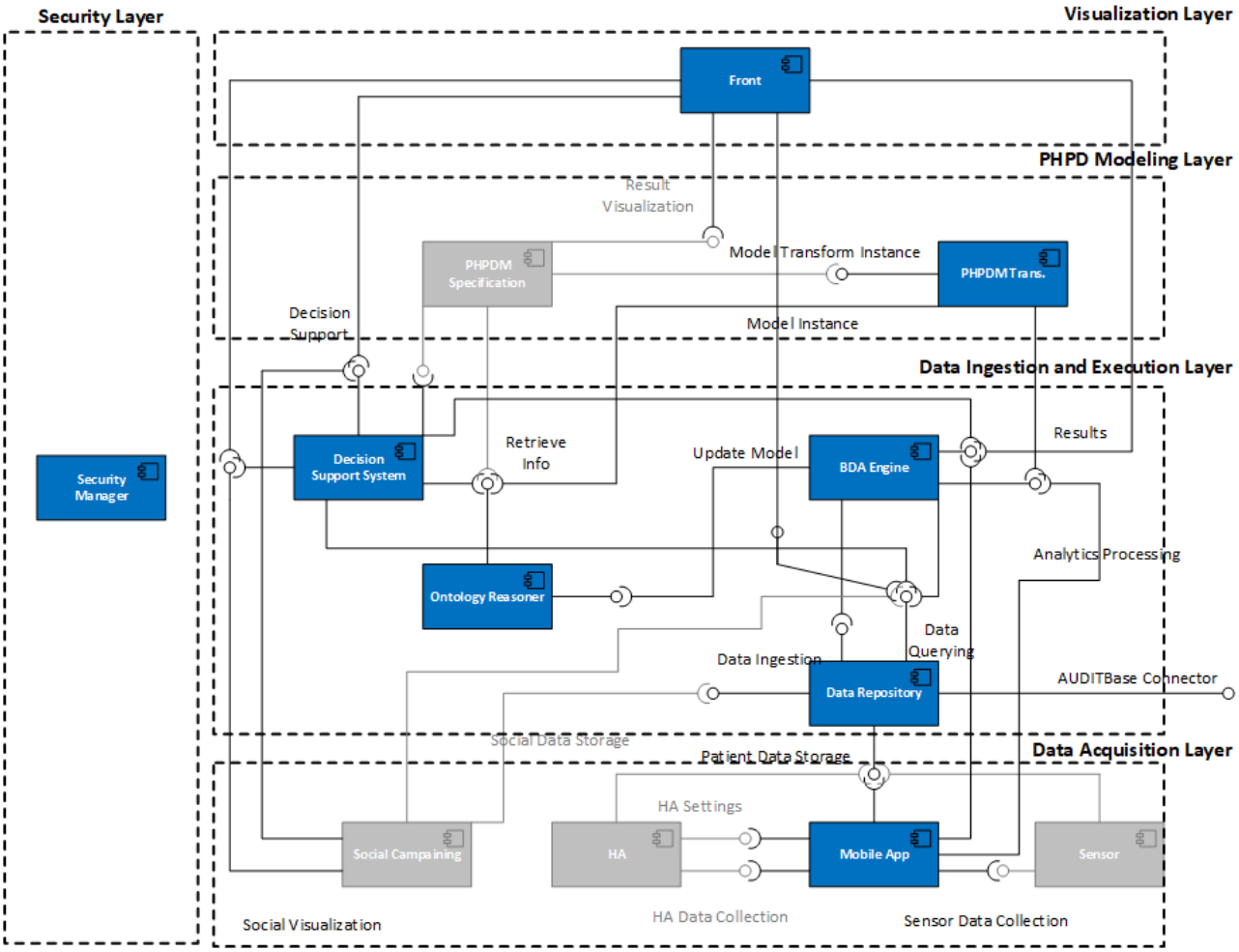


Figure 1 the architecture overview of the EVOTION platform: Principal components connected to the BDA

With regard to the technologies adopted for BDA development, they belong to the same Apache ecosystem used for the Data Repository. While in EVOTION the BDA and the Data Repository are considered two separate components, they are tightly integrated in the implementation and could exploit native features of the common technological framework.

More in general, as depicted in Figure 1, the BDA interact mainly with i) *Ontology Reasoner* to update ontological model instances, ii) *PHPDM Transformation* tool to receive an EDAW to be executed, iii) *Data Repository* to execute the analytics on stored data, iv) *Mobile App* to support HA user profile changes, iv) *Dashboard* to provide feedbacks about the status of a running EDWA and visualizing results, v) *DSS* to notify the completion of a EDWA, or to trigger one if needed.

In addition to the above architectural components, communication mechanisms have been developed focused on handling internal communication.

- **Enterprise Service Bus (ESB):** It provides the communication mechanisms between design components and implements data transformation between the EVOTION components.

- **Message Bus:** Provides a flexible mechanism to establish event driven communication channels between components. Each component can subscribe to a channel / topic and consume events that are sent over the Message Bus. Using Message Bus, a component can choose if it is necessary to react when a message arrives and, using the same channel, notify other components for this action.

1.3 Big Data Analytic Engine Component

The BDA Engine is based on a set of sub-components (see Figure 2) to fulfil the requirements specified in Section 1.1. Those subcomponents, that will be detailed in following Section 2, are:

- **BDA Infrastructure**: Big Data Processing infrastructure based on Apache Hadoop framework enhanced with additional components, like Apache Oozie, to support analytic workflow execution and libraries to supports statistical and data mining tasks, such as Mlib (requirements FR(BDA)1-4).
- **Task Catalogue**: list of Tasks (i.e., Processing, Statistical and Data Mining) for which an executable implementation is available in the BDA. For instance, the Task Catalogue will include Spark_ANOVA as an implementation of a task ANOVA defined in the PHPDM model instance (requirement FR(BDA)3, FR(BDA)8).
- **Workflow Catalogue**: list of EDAWs related to a corresponding PHPDM policy model instance. These workflows are composed of tasks appearing in the Task Catalogue and a coded logic driving the task execution. An EDAW can be scheduled according to PHPDM preference (requirements FR(BDA)5-7 and FR(BDA)9 and FR(BDA)10). It contains also a set of pre-defined EDAWs that are not derived from a PHPDM model instance, but are needed to cope with clinical specific scenarios.
- **Management/Catalogue Backend:** management backend for the BDA Infrastructure based on Ambari and for the Task and Workflow catalogues. It is used mainly for administrative management of the BDA Engine and Catalogues management (requirements FR(BDA)11).
- **API Module:** RESTFUL APIs for the EVOTION components interacting with the BDA (requirements FR(BDA)12).

These components are fundamental to provide the capability to process a DAW in the EVOTION framework. As a general remark, the idea is to keep the DAW in the PHPDM model instance as much generic as possible and deal with implementation details within the BDA Engine

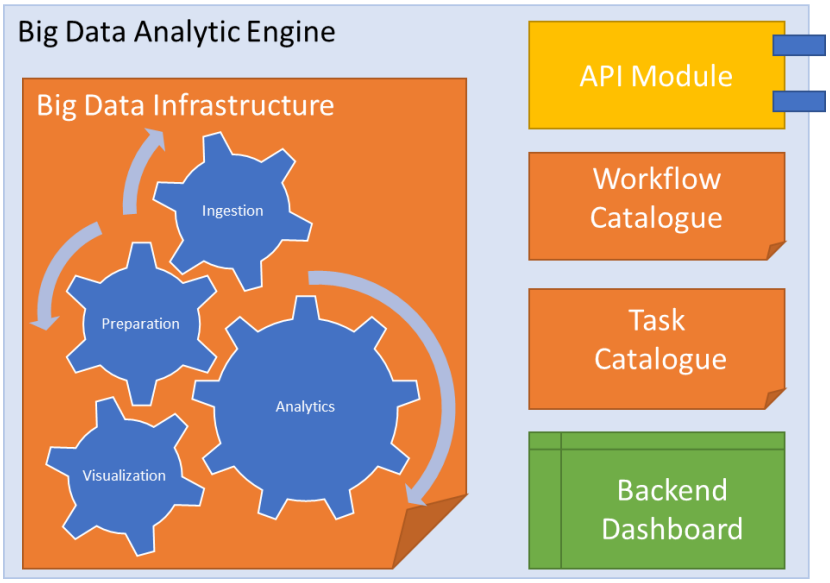


Figure 2: BDA Engine components

1.4 Definition and Execution of a DAW

Let us consider the process of defining and executing a DAW. A PHP policy model instance is defined through the PHPDM Specification tool by instantiating a PHPDM policy model. A PHPDM policy model instance is defined by means of the EVOTION declarative PHPDM language and of semantic definitions maintained by the Ontology Reasoner. It includes specification of tasks related to a DAW that needs to be transformed into an EDAW to be executed by the BDA engine.

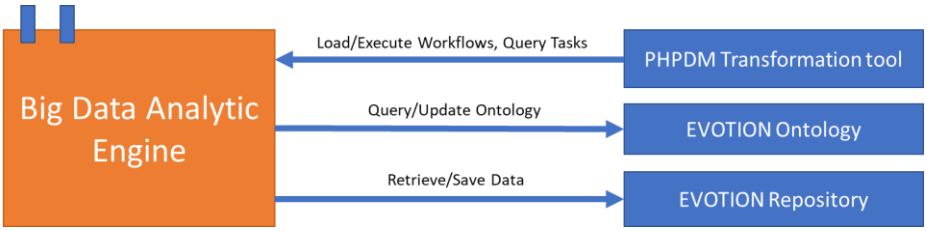


Figure 3: Architectural details of Big Data Analytic Engine interactions for DAW execution.

This operation is carried out by the PHPDM Transformation tool that takes a PHPDM policy model instance DAW, interacts with the BDA Task Catalogue to retrieve implementation details at task level, with the Ontology Reasoner to resolve semantic relationships, and then produce an EDAW to be stored into the Workflow Catalogue. The execution of the EDAW is then managed by a scheduler process and its evolution is notified by the BDA to the interested components, namely the Dashboard for feedbacks on the EDAW status and final results visualization, and the Ontology Manager to update the model of the given PHPDM model instance for instance locating the results of its execution.

We note that some EDAWs are not derived from a specific PHPDM policy, instead they are defined to support other scenarios like clinical and patient related scenarios. In that case, the EDAW is pre-defined and preloaded and can be triggered with parameters to tune their behaviour. The compatibility with TOREADOR mostly impacting on the capability of design these pre-defined EDAWs which are not model-driven by construction like the PHPDM ones, but that require modelling supports as well to provide fast prototyping.

We also note that tasks for PHPDM related EDAWs and pre-defined EDAWS will be continuously developed during the evolution of the project in the framework of WP4 Transformation Tool and WP3 Models.

2. Design

In this Section, we detail the design of each BDA sub-component defined in Section 1. We first revise the technologies that can be adopted for the BDA Engine and then we define how we structure our infrastructure. Finally, we present all the other sub-components constituting our BDA Engine.

2.1 EVOTION BDA Architecture

Given the BDA architecture presented in deliverable D2.1 and the requirements discussed in previous Section 1, we have designed the BDA infrastructure based on a tightly integrated set of tools developed by the Apache ecosystem.

The BDA Engine modular architecture integrates software tools and libraries selected from a rich open source, Big Data ecosystem having a large user-base and often employed in projects with characteristics and requirements similar to EVOTION.

More specifically, the BDA Engine will deal with a wide diversity of modern runtime environments. On top of such heterogeneous service architecture, an *Orchestration interface* (i.e., Task and Workflows Catalogue and the API module) has been deployed to maintain a runtime independent deployment logic. This way, procedures to deploy and manage the lifecycle of BDA's services can be delegated to runtime policies implementations.

In the following, we will be exploring the BDA Run Time Environment (RTE) architectural layouts, and what use cases will fit most with each one, along with an analysis of RTE tools.

In particular, the underlying technology composing the RTE should be flexible enough to support batch processing (mainly for policy making process) but also extendible to handle speed processing if needed (mainly to predict but also to update models given the availability of additional data). Therefore, given the two popular architectures Lambda and Kappa (initial review in D2.1), in the following we present the most suitable one for EVOTION, the Lambda Architecture.

2.1.1 Lambda Architecture

The Lambda Architecture (Grover et al 2015), consists of three layers: *Batch, Speed*, and *Serving* layer.

- *The Batch layer* merges incoming data to historical data and reiterates the procedural workflows on the combined data input to produce the results. The accuracy of batch views, comes at the cost of high latency, therefore Lambda Architecture must also be able to compute incremental updates via Speed layer to enhance the responsiveness.
- *The Speed layer* takes as input the new data in the form of either micro-batches (small chunks of data ingested on a regular basis) or single records, and the last update of batch data output. This implies that a requirement for algorithms to run in the Speed layer is to be able to process data incrementally.
- *The Serving layer* is composed by some ad-hoc low-latencies queries on indexed data resulting from the output of batch and real-time views to provide an aggregate view of both batch and speed layer data, resulting as a suitable interface for reporting and visualization tasks.

With both the Batch and the Speed layers, one of the problem to address is to synchronize them for preventing data redundancy or data loss. To address this problem, a solution has been proposed based on data tagging, tracking of delivery time, and selection of information that can be removed from speed data views, when batch processes produce results.

Selecting the correct architectural layout may depend on the application use case. For instance, recommender systems often make large use of machine learning techniques, like collaborative filtering via Matrix Factorization (MF) (Koren et al., 2009). In this case, to compute the MF matrix over all the historical data, some algorithms, like ALS, need to recursively process historical data, whenever new data chunks are delivered, while other algorithms are able to speed up the computation by incrementally update the MF models, at the cost of reduced precision, by providing an approximated result, with respect to the MF matrix calculated with ALS.

In this example, choosing a Lambda architecture layout is highly beneficial, since two different codebases could be used for Speed and Batch layers, as a trade-off between low latency (Speed layer) and high accuracy (Batch layer) in computing data views.

As a remark, EVOTION BDA is not handling Real Time/Near Real Time processing tasks only, like record level validation, window averages, counting and storing states, or even machine learning analytics having a native implementation like streaming linear regression in Spark MLlib. Other architectures focused on streaming layer (e.g., Kappa architecture), even if largely adopted and usable to obtain similar functionalities, are not perfectly tailored for the EVOTION BDA.

2.2 Big Data Runtime Environment

Figure 4 shows a Big Data Runtime Environment implementing Lambda architecture. In architectural view the Distribution Layer is added in order to deliver data to the processing part of the architecture.

2.2.1 Distribution Layer

Every time raw data is delivered to a BDA Distribution Layer, the process of ingesting data may involve several storage options depending on the size of data batched to be processed per time unit. One key factor to consider, when batches of raw data to be processed require large portions of storage and long-term historical data must be stored, is the optimization the data format for splittable compression. Other aspects to discuss are how data are updated when imported and how it is accessed. For instance, HDFS implements the "write once read many" paradigm, allowing no further update after the file has been created. Data is usually appended into "delta files" containing the incremental updates of last data batches imported. Having many small delta files inside the directories could require the execution of a specific process to merge files to reduce the overhead of read operations. This process is typically also able to produce an optimization by detecting records with the same key across the delta files and keep only the updated version records when merging delta files.

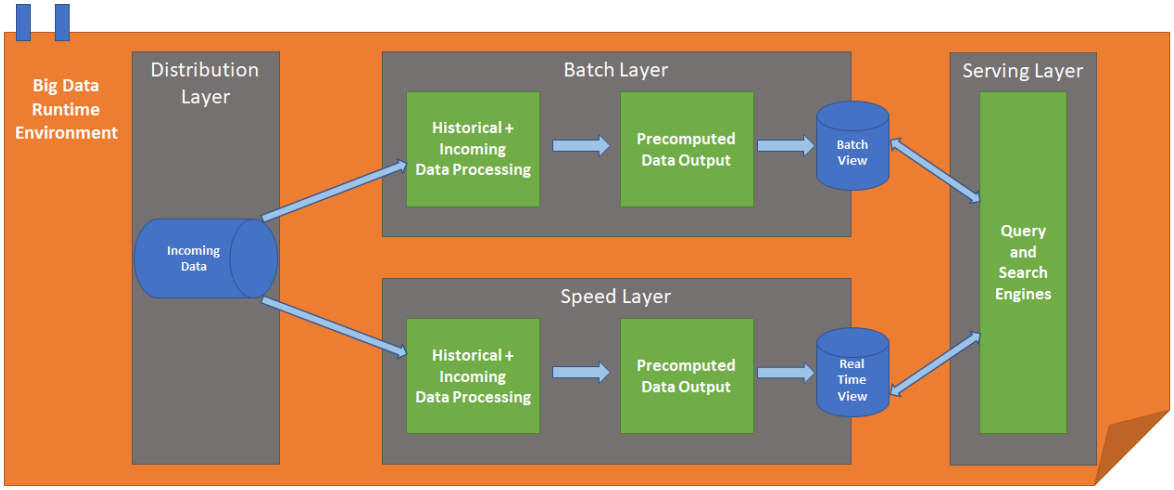


Figure 4: Big Data Runtime Environment

Considering the ingestion functionality of the distribution laver, different factors must be considered. One of the most relevant is data ingestion timeliness, namely the "time lag from when data is available for ingestion to when it's accessible to processing tools in the BDA ecosystem". A common classification for timeliness is:

1. *Macro batch*: Processes ingesting data from 15 minutes to hours.
2. *Micro batch*: Ingestion with data delivery time ranges of 2 to 15 minutes, often configured to continuously supply small amount of data to be asynchronously consumed and processed by short living processes.
3. *Near Real-time Decision support*: This is considered to be immediately actionable by the recipient of the information, with data delivered in less than 2 minutes but greater than 2 seconds.
4. *Near Real-time event-processing*: This usually refers to single events per time, processed in a time range going from 100ms to 2 seconds.
5. *Real Time*: It imports and processes single events per time and usually occurs in a period no longer than 100ms

Along with ingestion timeliness there are several factors to consider for choosing a suitable ingestion tool, we are referring to some of them in the following Table 3.

Table 3: Ingestion tools

Tool	Fault tolerance	Parallel ingestion	Transformation	Timeliness	Sources	Security
File Transfer	None (all or nothing)	Single-threaded	Only after landing in HDFS	Batch	External File System	SSL
Sqoop	Map tasks commit transactions periodically, resulting in a partial import/export up to the last commit	Using MapReduce Mappers to write a portion of the table to HDFS, loading multiple tables in parallel	Only after landing in HDFS	Batch	RDBMS	SSL

Kafka	Kafka replicates data to partitions, the level of fault tolerance is highly customizable, tuning in favour of availability or replicas consistency	Partitioned topics supply multiple consumer tasks in parallel	Only as data source for a streaming engine like Spark Streaming or Storm	Batch to RT	No native source nor sink implementation provided, but extreme pluggability thanks to pub/sub interfaces.	SSL and Kerberos
Flume	Supports splitting data on ingestion to feed a backup cluster	Flume supports multithreading and allows to tune up the fan-in with multi-agent ingestion	Supports low latency processing with interceptors	Near Real Time Event Processing/ Real Time	Spool Directory, HTTP, JMS, AVRO and many others	SSL

Table 3 shows some of the prominent ingestion tools and their peculiarities in terms of i) fault tolerance capability, ii) parallel ingestion method (single or multi thread), iii) data transformation modalities during the acquisition phase, iv) timelessness (batch/ real time /near real time process, v) acquisition sources supported and vi) security features. More details in the following.

- **File Transfer** is simply an ftp-based transfer protocol.
- **Sqoop** is made for efficiently transfer bulk data between HDFS and structured datastores such as relational databases. It supports incremental loads of a single table or a free form SQL query as well as saved jobs which can be run multiple times to import updates made to a database since the last import. Imports can also be used to populate tables.
- **Kafka** is the most customizable ingestion tool listed in Table 3, due to its simplicity. It is a distributed publish/subscribe message broker with implicit portability for real time and batch business applications.
  It works as a tool to manage streaming and operational data via in-memory analytical techniques for obtaining real-time decision-making. Kafka has persistent messaging, high-throughput, support for distributed processing, and support for parallel data load into Hadoop. Kafka combines off-line and on-line processing to provide real-time computation and produce ad hoc solution for these two kinds of data. Its streams API allows an application to act as a stream processor, consuming and producing input/output streams. It is mainly used in combination with other framework e.g. for building Lambda architecture.
- **Apache Flume** is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of streaming data into HDFS. It has a simple and flexible architecture based on streaming data flows; and is robust and fault tolerant with tunable reliability mechanisms for failover and recovery. YARN coordinates data ingest from Apache Flume and other services that deliver raw data into an Enterprise Hadoop cluster.

We note that the ingestion in this case is the ingestion operation internal to BDA, not external ingestion from other sources of data. External ingestion is carried out by the EVOTION Data Repository. We also note that current scenarios of EVOTION, as they are provided in D2.2, are focused mainly on macro and rarely on micro batch.

The ingestion of data can provide pre-processing capabilities offered by the above tools that includes:

- Clean selected data for better quality
- Treat missing values
- Identify or remove outliers
- Resolve redundancy caused by data integration
- Correct inconsistent data
- Flatten nested data formats to table formats
- Ordering

There is no silver bullet to accomplish these tasks without any ad-hoc transformation tailored to the data format and the nature of the discrepancies found. Discrepancies in a dataset may occur for many reasons, including failures during the ingestion process, data integration of different sources, and could result in anomalies not easy to find.

The adoption of data quality analysis tools allows to automatically detect many discrepancies in data. Apache Pig is the standard Hadoop script interpreter used for data quality in the pre-processing phase.

Anomaly detection and transformation processes are usually reiterated until no discrepancy is found in the dataset. Assuming that the process would take minutes to hours in a large data set, and would require some forms of human intervention (a common situation), a better approach called “Potter’s Wheel Framework” could be applied. Its principle is to reduce the amount of human interaction by incrementally applying discrepancy detection upon the records displayed to a user.

2.2.2 Analytics processing

As we have illustrated in Section 1, Lambda architecture is the preferred choice when we want to achieve a compromise between low latency and accurate batch calculations. Therefore, according to the Lambda architecture speed and batch processing should be considered.

This duality generates an important side effect for the Lambda architecture which is *code reusability*. This is an important feature to consider since it Lambda architecture requiring to maintain a double codebase, for streaming and batch operations. In EVOTION, batch and streaming workflows are in most of the cases disjoint, therefore this issue is less strategic. A more crucial aspect is the orchestration and synchronization of analytic tasks and workflows.

Orchestration of Analytics

While a big data processing is made of composition of different processing activities, an orchestrator is required to organize the processing workflow. The orchestrator manages the deployment of a generic BDA workflow to a target runtime environment, thus distributing computations and managing the resource consumption of the cloud infrastructure.

In general, the orchestrator is a sort of processing manager that may also take care of resource consumption while scheduling executions of workflows. It should take care of the following aspects:

- **Resource Management:** The amount of CPU, disk and memory reserved to every application process. This feature should be also able to monitor the health state of the system by measuring some specific resource metrics like:
    - **utilization** as the amount of time that the resource is busy, or the amount of resource capacity that is in use.
    - **saturation** as the measure of the amount of requested work that the resource cannot yet service, often queued.
    - **errors** representing internal errors related to a resource.
    - **availability** as the amount of time that the resource is available to respond to requests.
- **Fault tolerance:** Recover applications from downtime by maintaining the state.
- **Security:** Securing connections to end-points, and data encryption.

Lifecycle management is another relevant aspect, involving concepts like versioning, register deploy and removal, monitor active processes and managing their interruption. Two of the prominent tools working in the Apache ecosystem are the following.

- **Spring Cloud Dataflow** is a cloud-native Orchestration Service that aims to implement the key functionalities just described and to perform orchestration tasks sending commands via Service Provider Interface. It interacts with modern cloud infrastructure runtimes including: Apache YARN, Cloud Foundry, Apache Mesos, Kubernetes.
- **Oozie** is a largely used orchestration tool that covers the Apache Hadoop native ecosystem, and specifically designed for batch operations. It is not feasible to be used as a universal orchestration descriptive language, but still cover simple batch scheduling functionalities which are always required for handling orchestration of analytics.

Tasks composition patterns include sequential and parallel flows, failure handling, and decision nodes and they can be orchestrated by using Apache Oozie. Task execution design patterns include fan-out, point to point, capture or decide.

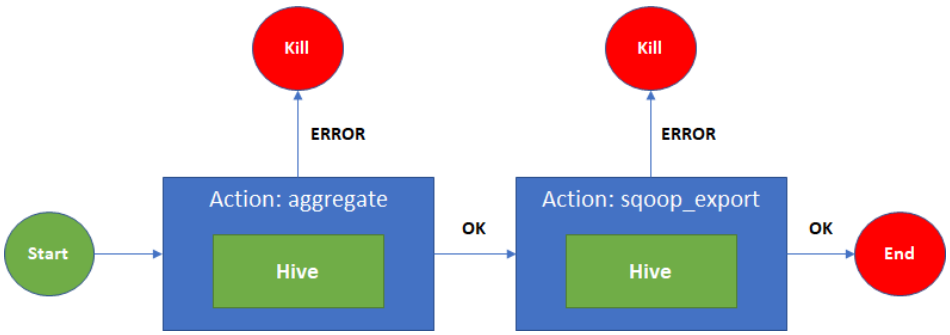


Figure 5: Point To Point workflow design pattern.

Point-to-point is common when some actions must be executed sequentially, e.g. Performing aggregation on data using Hive and if this process succeeds export it to RDBMS using Sqoop.

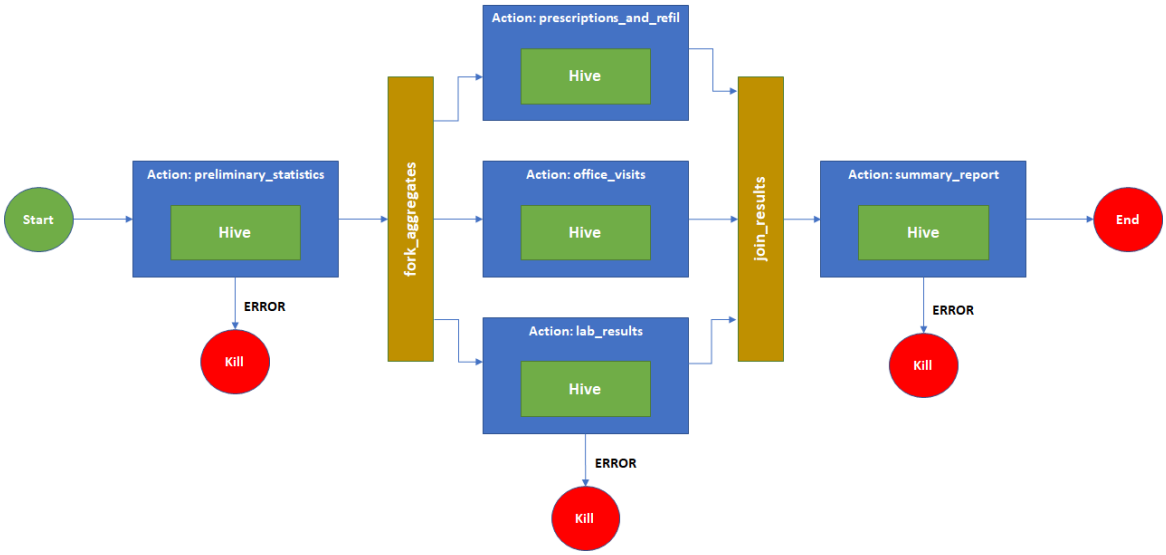


Figure 6: Fan-Out workflow design pattern.

Fan-out workflow pattern, also called fork-and-join pattern, runs some preliminary statistics, then executes several parallel tasks to fetch batches of result data from queries across different tables. It finally performs an aggregation task only if all the computations are performed with no errors. EVOTION platform is able to deal with the above patterns for handling its workflows.

2.3 BDA Analytic Ecosystem

BDA deals with High Performance Computation (HPC), which is a research field focusing on providing large processing capability by increasing parallelization and executing parallel tasks on top of a set of traditional computers communicating with each other. In the following, we introduce relevant tools for batch/micro batch data analytics.

2.3.1 Hadoop HDFS and MapReduce

The rise of Big Data research could be identified in the publication of two seminal papers from Google: one describing a distributed file system with fault tolerance capabilities (called Google File System) and another describing a distributed processing framework called MapReduce. The two systems were developed to tackle the problem of storing the index of the crawled webpages and the ability to recalculate the index in a timely fashion,as a batch process running during low traffic hours (Grover et al 2015-2).

The Hadoop[2] ecosystem was initially developed at Yahoo as an extension of Apache Nutch[3] (an open source web crawler) with the intention to implement the systems described in the two Google papers. In 2006, the Hadoop project became independent from Nutch and was released as an open source Apache incubator project. Initially the distributed file system and the processing framework were part of the Hadoop project, and later on, in 2009, were divided as different subprojects respectively named Hadoop Distributed File System (HDFS) and Hadoop MapReduce.

Since the release of the first version, a fast increasing number of companies adopted the framework for production systems. Nowadays, HDFS is the file system underlying most big data projects. It is a distributed

[2] http://hadoop.apache.org/
[3] http://nutch.apache.org/

file system developed in Java able to provide a reliable file system on commodity hardware. HDFS is scalable and fault-tolerant: in case more space is required, it is possible to add nodes to the HDFS clusters; when one or more nodes fail, the HDFS is able to still provide availability to the data stored on the isolated nodes by maintaining distributed replicas of data.

The architecture of a typical HDFS cluster is represented in Figure 7. There are two types of nodes: Data Nodes and Name Nodes. The first type is used to physically store data, while the latter is used to maintain a file table that is used to access the files.

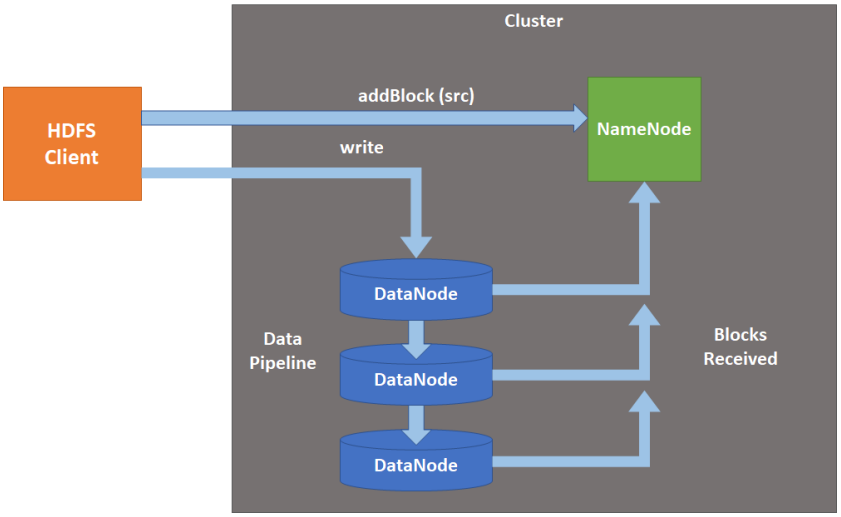


Figure 7: Architecture of an HDFS cluster

The developers of HDFS have improved the file system over time with the addition of several features described in the following Table 4.

Table 4: HDFS features.

Feature	Description
Rack awareness	Considers a node's physical location when allocating storage and scheduling tasks.
Minimal Data Motion	Hadoop moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides, which significantly reduces network I/O and provides very high aggregate bandwidth.
Health Utilities	Dynamically diagnose the health of the file system and rebalance the data on different nodes.
Rollback mechanism	Allow operators to bring back the previous version of HDFS after an upgrade, in case of human or systemic errors.
Minimal Intervention	HDFS requires minimal operator intervention, allowing a single operator to maintain a cluster of 1000s of nodes.
High Availability	Provides redundancy of the Name Node to supports high availability (HA).

The main problem with HDFS is that it does not perform well with small files, because it has been designed for storage of big sequential indexes and this limitation results from this initial assumption.

Hadoop MapReduce is the open source implementation of the MapReduce framework described in the Google paper. As the name suggests, it is composed by two phases: Map and Reduce.

The Map phase takes as input a set of objects and generates a set of Key-Values pairs. The criteria by which these Key-Values pairs are created depend on the implementation and on the problem to solve. As an example, we consider the typical word count problem. In the Map phase, each node takes as input a set of documents. Each document is tokenized into single words (Key) and for each word the Map phase counts the instances in the documents processed by the node (Value).

After the Map phase, there is an intermediate phase, which is called Shuffling, where the outputs of several Maps could be aggregated with same Keys and forwarded to a Reduce phase.

The Reduce phase takes as input a list of Key-Value pairs and creates a new list of Key-Values pairs. In the case of the word count problem, the Reduce phase takes the values associated with each Key and returns the sum.

The main problem with the Hadoop implementation of the MapReduce process is that the output of the Map phase is stored in the HDFS, possibly resulting in a bottleneck, because frequent and intensive disk operations can become very expensive in terms of latency, computational resources, and network bandwidth. These issues become more apparent in cases where it is necessary to update models with new data, which is the typical case in machine learning applications.

This limitation makes the Hadoop framework less performant with respect to more recent systems such as Spark, Flink and H_2O.

2.3.2 Mahout

On top of Hadoop MapReduce, a number of tools have been developed aiming at efficiently distribute machine learning tasks.

Among these tools, Apache Mahout[4] has been one of the most commonly used for managing machine learning techniques with Hadoop. However, a problem with Mahout over Hadoop is that, since April 2015, the Mahout development team is gradually removing support to algorithms running on native MapReduce environments, due to the inherent inefficiencies of frameworks like Hadoop. An alternative approach is to adopt a new environment for Mahout called Samsara, which includes statistical and algebraic operations. This change on the focus of Mahout has been motivated by the intention to provide a platform for developing distributed processing algorithms, rather than providing a set of ready-made ones.

Apache Mahout is both a machine learning library that can be integrated with different processing frameworks (e.g., Spark, H_2O and Flink) and can run either on a single machine or in a distributed environment.

Previously to its change of focus, Mahout provided a set of machine learning algorithms translated (where possible) into distributed processes and depending from the execution environment. The set of algorithms that were supported when Mahout was still fully supporting Hadoop were able to address classification, collaborative filtering, clustering, dimensionality reduction, topic modelling, Tf-Idf (Term frequency - Inverse document frequency), and others.

[4] https://mahout.apache.org/

Currently, the only types of algorithms supported are collaborative filtering, naive Bayes classification and dimensionality reduction. On the other hand, Mahout offers a very flexible solution to specific machine learning problems, allowing the composition of distributed algorithms from a set of basic operations. Some criticisms to Mahout were raised due to the difficulty with configuration, integration and development of new algorithms, however, older releases of Mahout were applied successfully in several production environments. At the moment of writing, due to the limited native support for machine learning algorithms, the suitability of Mahout for the EVOTION BDA is questionable, although it is still worth to be kept into consideration due to its flexibility in the creation of customized algorithms.

2.3.3 Spark

Spark[5] was developed at the University of California, Berkeley moving later on to become an Apache project. The framework is based on MapReduce. The main motivation behind the creation of Spark was to address the inefficiencies of Hadoop in situations of intensive disk usage. Spark supports iterative computation and improve speed and resource utilization by adopting an in-memory computation model. These features are particularly well-suited for machine learning tasks.

One of the main data abstraction used in Spark is called Resilient Distributed Dataset (RDD), which consists of a read-only distributed in-memory storage providing natively a fault tolerance mechanism without the need for physical replication of data on disk.

RDD can be the result of an input process, as for example importing data from HDFS file system, or the result of a transformation process from others RDDs to new RDDs. Spark allows another type of operation on RDD that is called *action*: this type of operation is used to produce final results from a RDD.

The architecture of Spark is composed by a Driver node and a set of Worker nodes as shown in Figure 8: a Driver node is where the program logic is executed, while Worker nodes store and perform operations on the RDD. Whenever is possible, there will be no data exchange between Workers nodes. For this reason transformations are divided into two groups: *narrow* and *wide* transformations, the first guarantees that no data between Worker nodes is exchanged (e,g., filter, map, sample), while in the second type of transformations data needs to be exchanged between Worker nodes (e.g., sort, group by, join,…).

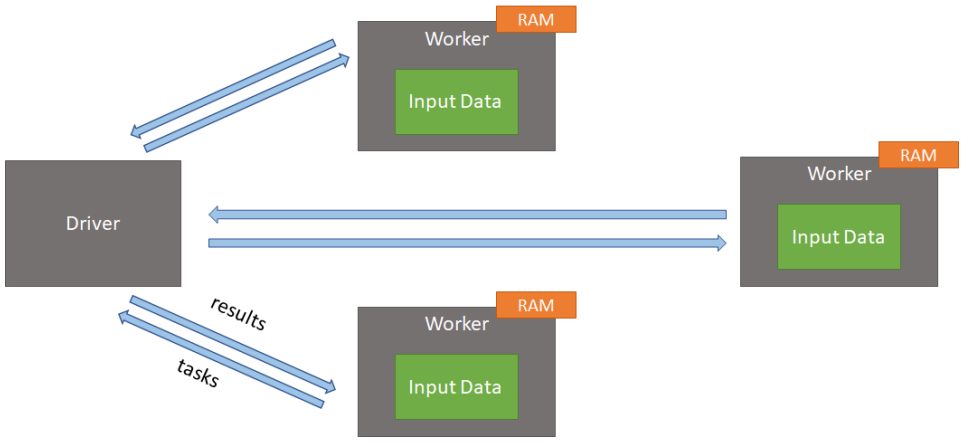


Figure 8- Example of Spark architecture

[5] https://spark.apache.org/

With Spark, intermediate results of the MapReduce computations, which were the reason for the performance problems with Hadoop, are stored in the distributed memory, significantly cutting down on the number of read and write operations on the file system and data exchange between nodes, as represented in Figure 9.

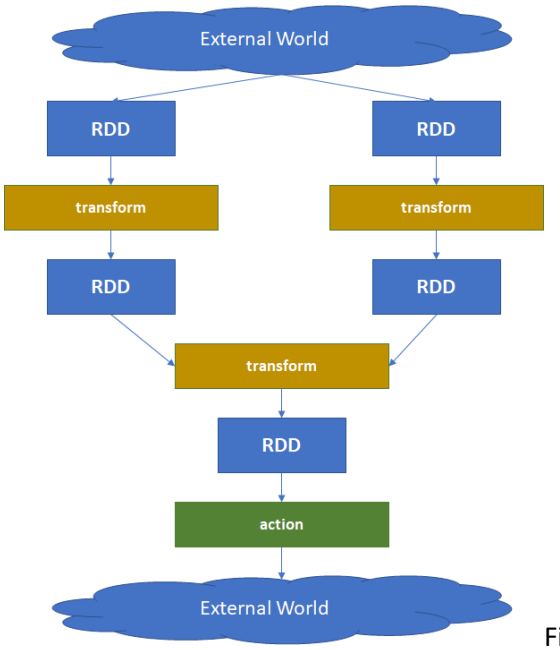


Figure 9- RDD processing pipeline

Spark also supports different programming languages (Java, Python, Scala and R), making it easy to develop and adapt to different environments.

However, even Spark has some limitations. Data transfers take place throughout the network and, because of the job isolation mechanism, only one Driver can serve requests to all of its RDDs, potentially leading to a bottleneck when there are multiple requests to multiple nodes.

Spark has also its own machine learning library called MLlib[6], which is similar to early versions of Mahout. It provides the same set of machine learning algorithms, topic modelling and frequent patterns mining. MLlib takes advantage of Spark features for iterative batch processing, stream (micro-batch) processing and in-memory caching of intermediate results and it is able to deliver better performances than Mahout. Other advantages of MLlib with respect to Mahout consist of its easy configuration and deployment, due to the fact that is tied to the RDD model and therefore does not require adaptation and integration efforts.

Some critics highlighted the fact that MLlib's behaviour in non-ideal situations (very large or very small vectors of data) have raised some issues. Other studies pointed out the fact that the tool has performance issues due to slow convergence of some algorithms.

More recently Mlib switch to DataFrame while RDD-Based APIs have entered in maintenance mode. A Dataset, introduced with Spark 1.6, is a distributed collection of data. Dataset is a new interface that provides the benefits of both RDDs and Spark SQL's optimized execution engine. A DataFrame is a Dataset organized into named columns that also provide a more user-friendly API than RDDs.

[6] http://spark.apache.org/mllib/

Additionally, Spark provides a concept called *ML pipeline*[7] built on top of Spark-SQL[8] library and allowing users to set up, build and execute machine learning processing pipelines. MLlib provides standardized APIs for machine learning algorithms which makes easier to combine MLlib algorithms into a processing pipeline. ML pipelines also allow creating non-linear processing pipelines, as long as they are defined as Directed Acyclic Graphs (DAG).

2.3.4 Flink

A relatively new approach supporting stream and batch processing is Apache Flink[9] originated from the Stratosphere research project of the University of Berlin. Flink combines database, stream processing and MapReduce technology to support user defined functions, complex data types, and the scalability of MapReduce systems. In addition, it offers the declarative, independence, and automatic optimization of database technology, allowing data analytics engineers to focus on the analytic problem instead of programming issues. Apache Flink also supports batch and stream processing, thus allowing the implementation of Lambda and Kappa processing architectures.

Similar to Spark, Flink is composed by a Job Manager and several Task Manager nodes. The Job Manager executes the program logic, while the Task Managers execute atomic operations on data. It supports generic Map and Reduce functions (allowing the execution of Flink code using traditional MapReduce approach without modification), but also some more specialized functions (e.g., join, group, iterate) and an optimization tool that automatically selects the best execution strategy for each process. A proprietary machine learning library is also available, called Flink-ML.

With respect to Mahout and MLlib, the list of supported algorithms is smaller: Support Vector Machines (SVM) and multiple linear regression are the only available supervised algorithm, while K-nearest neighbours is the only implemented unsupervised algorithm. An Alternating Least Squares (ALS) recommendation algorithm is also available as well as some data pre-processing methods and support utilities such as cross validation. Additional machine learning algorithms are expected to be available in future releases.

Still similar to Spark, the library provides the feature of building processing pipelines, in a similar fashion than the well-known scikit-learn[10] library developed for Python programming language.

Additionally, an adapter is available for the Apache SAMOA library (discussed in the following), which offers learning algorithms for stream processing.

The Flink team also published benchmark results on some machine learning tasks, such as Page Rank, reporting that Flink's execution was significantly faster than Spark.

2.3.5 H_2O

H_2O[11] is an open source project (with additional support for Enterprise editions) that, more than a distributed processing framework, can be considered as a complete analytical product on its own: it provides a distributed processing engine, data pre-processing, analytics, mathematics libraries, machine learning libraries and evaluation tools. As well as Spark, it offers support for Java, R, Python, and Scala languages and

[7] http://spark.apache.org/docs/latest/ml-pipeline.html
[8] http://spark.apache.org/docs/latest/sql-programming-guide.html
[9] http://flink.apache.org/
[10] http://scikit-learn.org/stable/
[11] http://www.h2o.ai/h2o/

it is able to execute Spark processes by integrating with Spark processing framework through its Sparking Water[12] library. H_2O is also able to execute its own processing models on top of Spark and Storm. Users with no programming skills could define processing pipelines via the Web interface.

Similar to Flink, H_2O processes data in-memory using multiple execution methods.

The generic approach to deploy a job in H_2O is called Distributed Fork/Join which is a divide and conquer technique adapting well to massively parallel tasks. This approach breaks down a processing job into smaller jobs, which are executed in parallel, resulting in dynamic fine-grained load balancing for MapReduce jobs as well as graphs and streams.

Regarding machine learning aspects, Mahout's library offers implementation of most of its machine learning algorithms for H_2O. However, H_2O platform is shipped with a ready-made machine learning module that in addition to traditional machine learning algorithms offers a set of tools for deep neural networks, which is nowadays a hot topic due to the important advances in several machine learning problems and the enormous hype that the media generated as a result of it.

At of the time of writing, the machine learning tools offered with H_2O are able to address a relatively wide set of tasks, including: classification, clustering, generalized linear models, statistical analysis, ensembles, optimization tools, data pre-processing options and deep neural networks. On the roadmap for future implementations there are additional algorithms and tools as well as recommendation and time-series analysis and prediction.

2.3.6 Streaming technology

Nevertheless, Hadoop is designed for batch processing, it shows multi-purpose features, but not the ones required for a real-time and high performance engine, due to the excessive throughout latency in its implementations.

Some of the batch frameworks previously described show a certain ability to deal with streams (e.g. Flink) but not in a real-time manner. Some native real-time Big Data platforms, such as Storm and Splunk, are specifically designed for real-time stream data analytics. It means that the ongoing data processing requires a very low latency of response, achieved by reducing the storing actions in the streaming pipeline (Zhang et al., 2016).

The streaming capabilities required for EVOTION are currently limited, therefore we have just revised the most commonly used solutions for streaming. Architecturally speaking, the EVOTION BDA is ready to extend its streaming capabilities easily if needed in the process of integration with the rest of the platform.

Storm

Storm[13] is specifically designed for distributed fault-tolerant real-time processing, initially conceived to overcome deficiencies of other processors in collecting and analyzing social media streams and currently released as open source.
To implement real-time computation on Storm, users need to create different topologies. A topology is a graph of computation representing the transformations of the stream, each node in the topology executes in parallel and the topology can be created and submitted using programming language. The Storm architecture consists of *spouts* and *bolts*. A spout represents one of the starting point of the graph denoting source of streams, while bolt processes input streams and outputs new streams. Each node in a topology

[12] http://www.h2o.ai/sparkling-water/
[13] http://storm.apache.org/

contains processing logic and links between nodes indicate how data should be processed between nodes. A Storm cluster consists of two kinds of working nodes, one Master node and several Worker nodes. The Master node (Nimbus) and Worker nodes (supervisor) implement two kinds of daemons responsible for distributing code across the Storm cluster, scheduling works assigning tasks to Worker nodes, and monitoring the whole system. The supervisor complies with tasks assigned by Nimbus, and starts or stops worker processes when necessary. The entire computational topology is partitioned and distributed to a number of worker processes implementing a part of the topology.
Storm was built as a stand-alone system independent from Hadoop. Recently some efforts have been devoted to integrate the two projects in the framework of the so-called Nathan Marz's "Lambda architecture".
Storm does not ship with a machine learning library but, for instance, SAMOA platform[14], for mining big data streams, has implementations for classification and clustering algorithms running on Storm, H2O has also offered a way to link the two projects, and Trident-ML offers a library of learning algorithms built on Storm, to name but a few.

Apache Samza

Apache Samza[15] is a distributed stream processing framework. It uses Apache Kafka for messaging, and Apache Hadoop YARN to provide fault tolerance (i.e. task migration), processor isolation, security, and resource management. Samza manages taking snapshots and restoration of a stream processor's state via consistent snapshot, and provides processor and resource isolation through Linux CGroups. It is part of the Hadoop ecosystem but also works with other messaging and executing environment thanks to the API.

Spark Streaming

Spark Streaming is an extension of Spark enabling scalable, high-throughput, fault-tolerant stream processing. Data ingestion can be originated from many sources like Kafka, Flume, TCP sockets, and can be processed using complex algorithms like map/reduce, as well as Spark's machine learning and graph processing algorithms. Processed data can be stored to filesystems, databases, and made useful in live dashboards. Spark Streaming provides a high-level abstraction called DStream (discretized stream) which represents a continuous stream of data as a sequence of RDDs. It divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

2.4 EVOTION BDA infrastructure layer

In this section, we illustrate the EVOTION architecture describing the main adopted tools. Figure 10 shows the main components of our Big Data infrastructure. We subdivided the platform components into four layers, as follows.

[14] https://samoa.incubator.apache.org
[15] http://samza.apache.org/

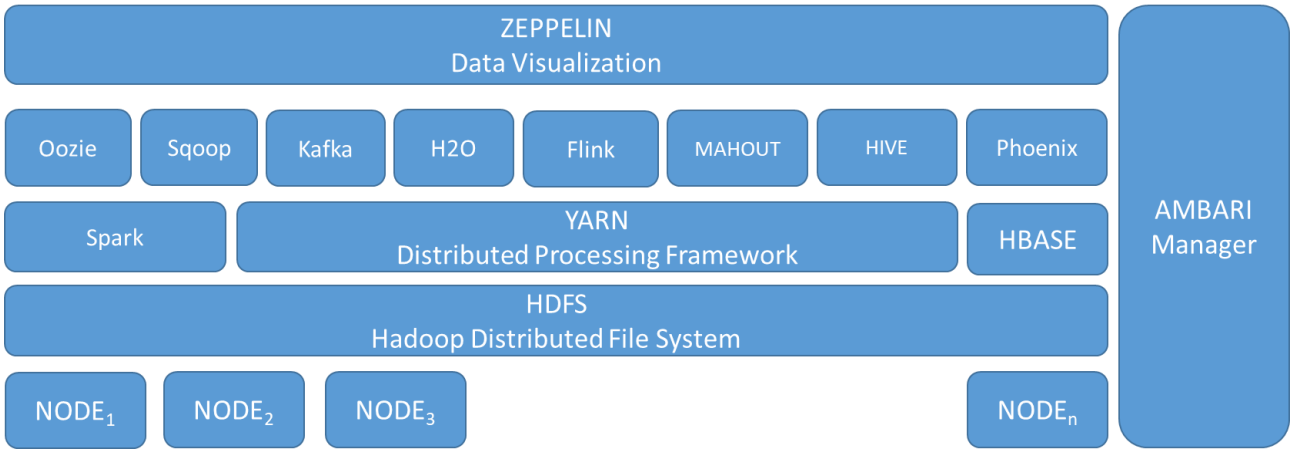


Figure 10 - EVOTION infrastructure overview

Data storing and management. The EVOTION BDA Engine adopts the Hadoop, a tool for data-intensive distributed applications, based on YARN programming model and a distributed file system called Hadoop Distributed Filesystem (HDFS). Hadoop allows writing applications that rapidly process large amounts of data on large clusters of compute nodes, which suits the EVOTION needs. A YARN tool permits to divide the input dataset into independent subsets that are processed in parallel. The data storage part of the BDA Engine is the EVOTION Repository that is designed based on the peculiarities of the EVOTION datasets. It adopts HBase, Hive and Phoenix (Kalakanti et al., 2015). Hbase as database engine optimized for real-time data access to large tables, Hive as the data warehousing infrastructure and Phoenix as data management suitable for massively parallel, relational database engine.

Data analysis and tasks. The EVOTION BDA provides a Task Catalogue based on a set of computation libraries to provide analytic and processing capabilities. Among the huge set of available libraries we selected: i) Spark, as general purpose cluster computing engine specialized at making data analysis faster supporting in-memory computing, ii) H_20 allowing users to fit thousands of potential models as part of discovering patterns in data, iii) Mahout as distributed linear algebra framework and mathematically expressive Scala DSL designed to let mathematicians, statisticians, and data scientists quickly implement their own algorithms, iv) Flink which is a dataflow programming model that provides event-at-a-time processing on both finite and infinite datasets.

Data flow management and workflow orchestration. The EVOTION BDA also offers Workflows Catalogue where executable workflows of analytics Tasks are stored. As orchestrator manager, we selected Oozie that permits to organize, manage and schedule workflows based on events and actions (Chuan-kai, 2014). Furthermore, EVOTION adopts Sqoop tool for efficiently transferring bulk data between Hadoop and structured datastores and vice versa. This can be useful in case of massive intermediate data to be moved to/from the Repository. EVOTION BDA Engine handles primarily batch and ready-to-orchestrate tasks, thus a flexible binding capability makes Kafka a convenient solution for EVOTION as well. Kafka will be primarily used to handle event driven processing of analytics.

Data visualization. The EVOTION BDA offers to the Dashboard a visualization tool to visualize results of analytics or directly ask for a simple aggregation or query. We adopt Zeppelin, a web-based and multi-purpose notebook that enables interactive data analytics. The notebook is a powerful yet limited tool, therefore we decided to use Zeppelin for simple analyses only, while we directly work with Spark for other complex analytics workflows.

Platform Management. The tool adopted by EVOTION BDA for platform management is Ambari, belonging to the Apache ecosystem. It offers features for provisioning, managing, monitoring and securing Apache Hadoop clusters. Ambari takes the guesswork out of operating Hadoop.

The architecture we adopted is capable to handle asynchronous update on the speed layer processing models and scheduled update on the batch layer handling scheduled analytics similarly to new models like bistro model (Savinov 2016, Savinov 2016-2 , Savinov 2017, Savinov 2017-2). In addition, we proposed a similar architecture focused on policy making in smart city scenario where the policies are simply defined with a deontic logic form (Anisetti at al., 2018). Fang et al. presents a survey on Big data application in health informatics. The survey undeline the BDA capabilities and how the current solutions are used for health and medical application (Fang et al., 2016). Rathore et al. proposes a Big data engine based on Hadoop but focused more on specific assisted healthcare (Rathore et al., 2017). EVOTION requires a much more flexible architecture than the one proposed in literature, allowing to programmatically customization and set up of analytics.

As it is structured The EVOTION BDA infrastructure is a TORADOR compatible landing platform meaning that it has orchestrations and execution capabilities that are in general required by a TORADOR big data infrastructure aimed to address analysis like the ones required for the EVOTION.

As an additional remark, the TOREADOR compatibility is provided by the development of the two catalogues, the Task and the Workflows catalogues that supports model-driven definition of analytics to be executed especially, but not only, the ones not driven by the PHPDM. In the following we will details these catalogues and their role in the creation and execution of analytics.

2.5 Task Catalogue

The Task Catalogue handles the available Implemented Analytic Tasks that can be composed into an EDAW by the PHPDM Transformation tool.

An Implemented Analytic Task is a DAW Task for which code had been developed to implement the relative Algorithm (e.g., cleaning, clustering) or simple Operation (e.g., data source selection), and it had been deployed into the EVOTION BDA in such a way that it can be executed while embedded into an EDAW.

An Implemented Analytic Task, as entry of the Task Catalogue includes the following attributes:

- unique ID;
- implemented Analytic Task name;
- the corresponding DAW Task ID in the EVOTION Ontology;
- the name and the version of required software libraries;
- the programming language used in the development;
- the path of source code/executable;
- details on dependencies (e.g., another task to be extended) if any;
- textual description about the algorithm;

In the following we provide an example of Task Catalogue Entry for *Spark_OneWayANOVA*, an implementation of *ANOVA* Task using Spark libraries.

Attribute	**Value**
ID:	123
Name:	Spark_OneWayANOVA
DAWTaskID:	StatisticalAnalysisAlgorithm:ANOVA
Lib:	Spark v 2
Language:	Scala
Path	HDFS://
Dependencies:	none
Description:	Scala Implementation of Simple OneWayANOVA using Spark

A number of algorithms implementing specific tasks are available in different libraries for Big Data computations (some examples are listed in Table 1).[16]

While libraries can be used directly to implement a task, they offer heterogeneous interfaces and express parameters and data formats in different ways. In order to have a uniform catalogue of Implemented Tasks, each Implemented Analytic Task implements an interface to wrap the adopted libraries (i.e., Task Wrapper interface) into a common standard interface. This Task Wrapper interface reduces dependencies on the libraries implementing specific algorithms and makes the orchestration of Tasks into EDAWs more straightforward.

More specifically, the PHPDM Model Instance defines, for each DAW's Task, some parameters in terms of expected input and output. The Task Wrapper maps them to the library ones to resolve library dependencies at task level.

[16] Our Implemented Analytic Task either will exploits these libraries when available or implements a specific algorithm from scratch.

Our Task Wrapper offers: i) *Init method* to maps expected parameters/input with the library ones, set up environmental settings, transform data types etc., ii) *Run method* to execute the core algorithm relative to the Implemented Data Analytic Task, iii) *postProcessing* method to reshape the output of the algorithm as expected by the PHPDM model instance (OutputDataSpecifications).

For instance, let us consider simple K-Means which is defined as an Unsupervised Data Mining Analytic Task in D4.1 (Prasinos at al., 2017). It can be implemented using the Mahout Hadoop library, and deployed in the BDA with all dependencies satisfied ready to be executed. The task implementation requires to wrap the Mahout-kmeans into our Task Wrapper to provide standardized interface. The Init method must be implemented to make the merge between the parameters expressed at PHPDM model instance level (e.g., seed, numKMeansRuns, maxIterations, to name but a few) and the ones relative to the specific Mahout-kmeans implementation. For instance, in case that the type of distance is not specified in the PHPDM Model Instance, the Init method must select one distance definition among the ones available in Mahout (e.g., ManhattanDistanceMeasure. class.getName()). The Run method executes the K-means calling the Mahout implementation (i.e., KMeansDriver.runJob()) and the postProcessing method maps the output (i.e., a csv file stored in the filesystem) to a table in the Repository as for instance requested by the PHPDM Model Instance.

In the following, we present the Task Wrapper Interface in Scala.

```
trait Task {
val params: Map[String,Any]
def init def run
def postprocessing
}
```

In Table 5, we presented some of the Data Mining algorithms that can be implemented and added to the Task Catalogue according to the relative library. According to D4.1, a number of them can be implemented to cover the scenarios of D2.1. We note that for some of them we have potential multiple implementations depending on the selected libraries.

The Task Catalogue will be incrementally populated with Task Implementations during the development of the PHPDM Transformation Tool in WP4.

Table 5: Some of the available Data Mining algorithms that could be specified in DAW Tasks and listed in the Task Catalogue

Task Name	Framework	Library	Algorithm
Hadoop-Mahout-LogisticRegression	Hadoop	Mahout	Logistic Regression using Stochastic Gradient Descent (SGD)
Hadoop-Mahout-NaiveBayes	Hadoop	Mahout	Standard Multinomial Naive Bayes Classifier, model and prediction
Hadoop-Mahout-Canopy	Hadoop	Mahout	Canopy clustering
Hadoop-Mahout-KMeansClustering	Hadoop	Mahout	K-means clustering, model and prediction
Hadoop-Mahout-FuzzyKMeans	Hadoop	Mahout	K-means with fuzzy sets
Hadoop-Mahout-Spectral	Hadoop	Mahout	Spectral clustering

Spark-Mllib-SVMModel	Spark	MLlib	Model for Support Vector Machines
Spark-Mllib-SVMPredict	Spark	Mllib	Prediction for a given SVM Model
Spark-Mllib-LogisticRegresionModel	Spark	Mllib	Model for logistic regression
Spark-Mllib-LogisticRegresionPredict	Spark	Mllib	Prediction for a given Logistic Regression Model
Spark-Mllib-RegressionModel	Spark	Mllib	Liner Regression Model (SGD)
Spark-Mllib-RegressionPredict	Spark	MLlib	Regression prediction given a Model
Spark-Mllib-DecisionTreeClassificationModel	Spark	MLlib	Model for Decision Tree
Spark-Mllib-DecisionTreeClassificationPredict	Spark	MLlib	Prediction for a given Model for Decision tree
Spark-Mllib-DecisionTreeRegressionModel	Spark	MLlib	Model for Decision Tree Regression
Spark-Mllib-DecisionTreeRegressionPredict	Spark	MLlib	Prediction for a given Model for Decision Tree Regression
Spark-Mllib-RandomForestClassificationModel	Spark	MLlib	Model for Random Forest (ensembles of decision trees)
Spark-Mllib-RandomForestClassificationPredict	Spark	Mllib	Prediction for a given Model for Random Forest
Spark-Mllib-RandomForestRegressionModel	Spark	Mllib	Model for Random Forest Regression
Spark-Mllib-RandomForestRegressionPredict	Spark	MLlib	Prediction for a given Model for Decision Tree Regression
Spark-Mllib-GradientBoostedTreeClassificationModel	Spark	MLlib	Model for Gradient-boosted tree Classification
Spark-Mllib-GradientBoostedTreeClassificationPredict	Spark	MLlib	Prediction for a given Model for Gradient-boosted tree Classification
Spark-Mllib-GradientBoostedTreeRegressionModel	Spark	MLlib	Model for Gradient-boosted tree Regression
Spark-Mllib-GradientBoostedTreeRegressionPredict	Spark	MLlib	Prediction for a given Model for Gradient-boosted tree Regression
Spark-Mllib-NaiveBayesModel	Spark	MLlib	Model for Naïve Bayes Classifier

Spark-Mllib-NaiveBayesPredict	Spark	Mllib	Prediction for a given Model for Naïve Bayes Classifier
Spark-Mllib-KMeansModel	Spark	Mllib	Model for K-means
Spark-Mllib-KMeansPredict	Spark	MLlib	Prediction for a given Model for K-means
Spark-Mllib-PicModel	Spark	MLlib	Model for Power Iteration Clustering
Spark-Mllib-PiicPredict	Spark	MLlib	Prediction for a given Model for Power Iteration Clustering
Spark-Mllib-LdaModel	Spark	MLlib	Model for Latent Dirichlet allocation (topics from text)
Spark-Mllib-LdaPredict	Spark	MLlib	Prediction for a given Model for Latent Dirichlet Allocation
Spark-Mllib-BisectingKmeansModel	Spark	MLlib	Model for Bisecting k-means (hierarchical clustering)
Spark-Mllib-BisectingKmeansPredict	Spark	Mllib	Prediction for a given Model for Bisecting k-means
Spark-Mllib-Pca	Spark	Mllib	Model form Principal component analysis (dimensionality reduction)
Spark-Mllib-AssociationRules	Spark	Mllib	Association Rules Learning (relation discovering between variables)
Flink-FlinkML-LinearRegression	Flink	FlinkML	Linear Regression
Flink-FlinkML-MultipleLinearRegression	Flink	FlinkML	Multiple Linear Regression Model and prediction
Flink-FlinkML-KMeans	Flink	FlinkML	K-means clustering
Flink-FlinkML-Knn	Flink	FlinkML	k-Nearest Neighbors Classification
Flink-FlinkML-Svm	Flink	FlinkML	Support Vector Machine
H20-GlmModel	H2O	H2O-ML	Generalized Linear Models Model
H20-GlmPredict	H2O	H2O-ML	Prediction for a given Generalized Linear Models Model
H2O-GbmModel	H2O	H2O-ML	Gradient Boosting Machine Model
H2O-GbmPredict	H2O	H2O-ML	Prediction for a given Gradient Boosting Machine Model

H2O-DeepLearningModel	H2O	H2O-ML	Deep Learning based on a multi-layer feedforward artificial neural network Model
H2O-DeepLearningPredict	H2O	H2O-ML	Prediction for a given Deep Learning Model
H2O-RandomForestModel	H2O	H2O-ML	Distributed Random Forest Model
H2O-RandomForestPredict	H2O	H2O-ML	Prediction for a given Distributed Random Forest Model
H2O-NaivebayesModel	H2O	H2O-ML	Model for Naïve Bayes Classifier
H2O-NaivebayesPredict	H2O	H2O-ML	Prediction for a given Naïve Bayes Classifier Model
H2O-StackEdensemblesModel	H2O	H2O-ML	Stacked Ensembles Machine Learning Model (multiple learning algorithms)
H2O-StackEdensemblesPredict	H2O	H2O-ML	Prediction for a given Stacked Ensembles Classifier Model
H2O-XgboostModel	H2O	H2O-ML	XGBoost boosted supervised learning Model
H2O-XgboostPredict	H2O	H2O-ML	Prediction for a given XGBoost Model
H2O-Kmeans	H2O	H2O-ML	K-Means
H2O-Pca	H2O	H2O-ML	Principal Component Analisys
Stream-Spark-Mllib-SVMPredict	Spark stream	MLlib	Batch model and Stream for SVM prediction
Stream-Spark-Mllib-LogisticRegression-Predict	Spark stream	MLlib	Streaming Logistic Regression
Stream-Spark-Mllib-Regression-Predict	Spark stream	MLlib	Streaming Regression
Stream-Spark-Mllib-StreamingLinearRegression	Spark stream	MLlib	Streaming Linear Regression
Stream-Spark-Mllib-DecisionTreeClassificationPredict	Spark stream	MLlib	Batch model and Stream for Decision Tree Classification prediction

Stream-Spark-Mllib-DecisionTreeRegressionPredict	Spark stream	MLlib	Batch model and Stream for Decision Tree Regression prediction
Stream-Spark-Mllib-RandomForestClassificationPredict	Spark stream	MLlib	Batch model and Stream for Random Forest prediction
Stream-Spark-Mllib-RandomForestRegressionPredict	Spark stream	MLlib	Batch model and Stream for Random Forest Regression prediction
Stream-Spark-Mllib-GradientBoostedTreeClassificationPredict	Spark stream	MLlib	Batch model and Stream for Gradient Boosted Tree Classification prediction
Stream-Spark-Mllib-GradientBoostedTreeRegressionPredict	Spark stream	MLlib	Batch model and Stream for Gradient Boosted Tree Regression prediction
Stream-Spark-NaiveBayesPredict	Spark stream	MLlib	Batch model and Stream for Naïve Bayes prediction
Stream-Spark-MLlib-Kmeans-Model	Spark stream	MLlib	Streaming Model for K means
Stream-Spark-MLlib-Kmeans-Predict	Spark stream	MLlib	Streaming for K-means prediction
Stream-Spark-MLlib-Pic-Predict	Spark stream	MLlib	Batch model and Stream for Power Iteration Clustering prediction
Stream-Spark-Mllib-Lda-Predict	Spark stream	MLlib	Batch model and Stream for Latent Dirichlet allocation prediction
Stream-Spark-Mllib-BisectingKMeans-Predict	Spark stream	MLlib	Batch model and Stream for Bisecting k-means prediction

We note that the above implementations are in general splitted into training/modelling Tasks and testing/predicting Tasks. We also note that in most cases, streaming processing adopts a model constructed via batch analysis.

In addition to data mining related tasks, the Task Catalogue includes Statistical Tasks like linear regression, ANOVA and F-Test to name but a few. In the following, Table 6 shows some of the Statistical algorithms that could be implemented as Task in the Catalogue.

Table 6: Some of the available statistical algorithms that could be specified in DAW Tasks and listed in the Task Catalogue.

Task Name	Framework	Library	Algorithm
Spark_LinearRegressionStats	Spark	Mlib	Linear Regression for Analysis of continuous dependent variable
Spark_OneWayANOVA	Spark	Mlib	OneWayANOVA
Spark_TwoWayANOVA	Spark	Mlib	TwoWayANOVA
Spark_MultiWayANOVA	Spark	Mlib	MutiWayANOVA
Spark_BPTest	Spark	Mlib	Breusch-Pagan Test
Spark_Ftest	Spark	Mlib	F-Test
Spark_Fisher	Spark	Mlib	Fisher's Exact Test

The catalogue also contains Processing Tasks involving pre-processing algorithm (e.g., feature selection and filtering) and operations (e.g., data cleaning, data merge, vertical and horizontal selection). Some of them are parametric but generic like merging, cleaning, selection. Other are more similar to Data Analytics Tasks but devoted to prepare the dataset for further analytics, like clustering to aggregate data set attributes and features selection. Concluding, tasks related to visualization are also part of the catalogue, like Zeppelin base task for visualization. They are optional and considered as part of the Processing Task in EVOTION.

2.5.1 Task Catalogue's APIs

The Task Catalogue offers the following functionalities for internal use only:

- **AddTask**: add an implementation of a given DAW Task. More implementations of the same DAW Task could coexist, for instance provided by different libraries. We allow the PHPDM Transformation tool to specify a particular available implementation of a Task.
- **ModifyTask**: modify any of the Task related aspects, like pointing to different implementation, modifying specific Task attributes like the reference library to name but a few.
- **RemoveTask**: remove a Task from the Catalogue with all its implementation, or to remove a specific implementation of a given Task. It is useful in case of update of libraries versions that makes some implementation no more executable, or deprecated.

It also offers a functionality for the interaction with the PHPDM Transformation tool:

- **FindTask**: search for an implementation given a DAW Task ID. It returns details needed to embed the Implemented Task into the EDAW (i.e., the entire Task Catalogue entry).

We note that when a Task is added to the Task Catalogue, the Ontology Manager is notified for updating the related Ontology Class with the corresponding Task and adding the new available implementation as subclass. Similarly, in case of removal of a given Implemented Analytic Task, the Ontology Manager is

notified. This will be of help to let EVOTION Stakeholder disambiguate between different implementations of the same DAW Task.

2.6 Workflow Catalogue

The Workflow Catalogue is responsible to handle EDAWs. To this aim, it is composed of three sub-components (see Figure 11): i) the Catalogue stores/removes EDAWs, ii) the Workflow Scheduler schedules the execution of EDAW according to the specification given (i.e., periodic, upon request, or driven by data size change), iii) the Workflow Manager responsible to keep track of the running EDAWs.

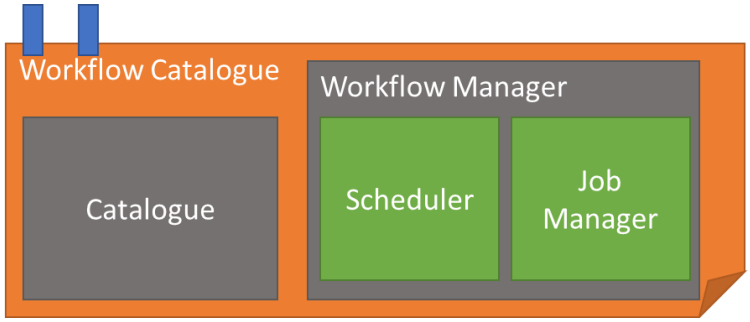


Figure 11: Internal structure of Workflow Catalogue

The EVOTION BDA Infrastructure is able to support orchestrated workflows, requiring the intervention of a dynamic orchestrator like Oozie, and statically generated workflows directly executable by the BDA Engine (EVOTION Native Workflows).

How the workflows are defined and structured will be detailed in the PHPDM Transformation Tool, of work package WP4, and the corresponding deliverable D4.3 to be released at M30. In this report, for sake of simplicity, we consider just statically generated workflows as running examples.

An EDAW, as an entry of the Workflow Catalogue, includes the following attributes:

- Unique ID;
- EDAW name;
- the corresponding DAW ID specified by the PHPDM Specification tool;
- the list of Implemented Analytic Tasks included in the EDAW;
- list of (formal) parameters for each Task of the EDAW, and the list of parameters for the workflow as a whole;
- the language used for the development (e.g., Scala, Java)/ the Orchestrator adopted for the orchestration (e.g., Oozie, Scala);;
- the path to the source/executable code or the path of the Orchestration meta description;
- the Workflow Execution Type (on request, schedules, data change driven) and the relative scheduling preference (e.g., every day, on request, when size increases about 20%);
- the Job ID, and identification of the running Job related to this EDAW as it is defined by the Workflow Manager
- a textual description of the Workflow purpose and characteristics.

In the following, we provide an example of a Workflow Catalogue Entry related to an EDAW (HA_Barriers_WF1) implementing a DAW (WF1) defined to cope with part of the D4.1 scenario “Addressing Barriers to HA usage”.

Attribute	Value

ID:	123
Name:	HA_Barriers_WF1
DAWID:	WF1
Tasks	<FeatureSelection, Spark_ANOVA>
Params:	List of Structured datatype like JSON one per Tasks (two JSON, plus one for the workflow-specific parameters)
Language:	Oozie
Path	HDFS://…
WorkflowExecutionType:	Monthly
JobID	1234
Description:	The implementation of WF1 relative to the evaluation if the Education level impacts the HA usage.

In Appendix A, we provide an initial modelling of some of the analytic workflows of interest for EVOTION.

2.6.1 Workflow Manager

Figure 11 shows the internal structure of the Workflow Manager which is responsible to handle the execution of the EDAW in the Catalogue when requested and coherently with the required scheduling preference (Scheduler module), and manage the running EDAW (Job Manager module).

More in details, the Workflow Manager is responsible to add the EDAW to the list of EDAWs to be executed (jobs list) according to scheduling preferences. In case of event driven scheduling (e.g., specific increment of the data size), the component periodically checks the scheduling preference and triggers the requested EDAWs execution. When the execution of an EDAW job is triggered, it is added to the list of running jobs until its completion. The running job list is handled by the *Job Manager* which can list all the running jobs of corresponding EDAWs and stop each of them releasing resources.

Scheduling capabilities are useful for recurrent analysis but also as a way to connect multiple workflow executions.

For instance, let's consider a scenario requiring Data Mining Analytics. In this case, two workflows should be triggered: i) the model learning workflow (whose execution is triggered by a given increment in the data size used for learning) aimed at generating a model based on the available data, ii) the evaluation workflow (whose execution is triggered by the availability of a new learning model) aimed at performing the evaluation using the most recent learning model.

During an EDAW execution, when a Task completes, a notification is produced to update the EDAW status at Dashboard level and to update the relative DAW in the PHPDM Model Instance stored in the Ontology Manager with intermediate results, if needed. These notifications are handled at Task level within the Task Wrapper.

When the execution of an EDAW is completed, the results are saved into the EVOTION Repository and a notification is released to notify EVOTION users, as well as the Ontology Manager in order to update, with the obtained results, the PHPDM Model Instance relative to the executed DAW.

The notification format always includes the EDAW ID and the location where results are available. The specific data structure of the results is part of a DAW specification.

Specific mechanisms to release notifications will be defined during the integration work package (WP6). Advanced approaches are currently under discussion, including Enterprise Service Bus and Notification Service.

The Workflow Manager therefore offers the following functionalities for internal use only:

- **AddEDAWJob**: add an EDAW to the list of scheduled jobs according to the scheduling preference.
- **RemoveEDAWJob**: remove an EDAW from the list of scheduled jobs. If in running state, it is firstly stopped (StopRunningEDAWJob) and then removed.
- **ListRunningEDAWJob**: list all the running jobs and the relative EDAWs. It can be also used as a way to monitor resource consumption.
- **StopRunningEDAWJob**: stop a specific job. For instance, some EDAWs may last for a long period of time depending on the complexity of the implemented Workflow; if an EVOTION Stakeholder decide to stop an analytic which is no more needed, she can interact with the BDA for stopping a running EDAW releasing resources for other computations. When a EDAW is stopped it still remains in the schedule list.

2.6.2 Workflow Catalogue's APIs

The Workflow Catalogue offers APIs to the rest of the EVOTION platform to handle and trigger execution of EDAWs. In the following, we present a revised version of the APIs initially provided in D2.2.

- **AddWorkflow**: allow to add a given EDAW into the Workflow Catalogue. When an EDAW is added, it provides as output the ID of the EDAW as it is saved in the Catalogue. Adding a EDAW is not automatically triggering its execution.
- **FindWorkflow:** allows to find a workflow by its ID or a list of Workflows implementing a specific DAW.
- **ModifyWorkflow**: allow to modify/update parameters for each Task of a loaded EDAW as well as EDAW attributes like scheduling preferences. For instance, it permits to modify the number of clusters *k* of an EDAW requiring k-means Task of to update the Workflow code. A modification of an EDAW that is in running state requires that it is stopped and removed from the scheduled job list.
- **RemoveWorkflow**: allow to remove an EDAW from the Catalogue. When an EDAW is removed from the Catalogue, it is also removed from the job schedule list (RemoveEDAWJob).
- **ExecuteWorkflow**: execute a given EDAW loaded in the catalogue. It can be also used to force re-execution of an EDAW coherently with the schedule preference. The use of this functionality automatically triggers the creation of a job in the scheduled job list (AddEDAWJob).
- **StopWorkflow**: stop the execution of an EDAW and perform the relative rollback activities if required (e.g., removing temporal stored data). The use of this functionality automatically triggers the stop of the relative job in the list of running jobs (StopRunningEDAWJob)

As a general comment, we note that different EVOTION Stakeholders may have different instances of scheduled EDAWs corresponding to the same source EDAW. The dashboard is the component responsible to maintain the references between EVOTION Stakeholders and the relative processes including PHP policies and EDAWs.

2.7 Infrastructure/Catalogues Management Backend Dashboard

We developed a BDA Backend Dashboard to allow direct interaction with the BDA Engine and to support execution of workflows before the final integration of the EVOTION architecture takes place. The usage of this dashboard is restricted to EVOTION administrators only and it will not be released to EVOTION Stakeholders. For instance, EVOTION data scientists that want to extend EVOTION Task Catalogue with additional analytic tasks will access the Task Catalogue through this dashboard and operate using the Web GUI.

Summarizing, the BDA offers a Management web interface for:

- Task Catalogue to search, list, add modify and remove Implemented Analytics Tasks.
- Workflow Catalogue to list all available EDAWs and to manually add/modify a given EDAW.
- Workflow Scheduler, to see scheduled EDAWs in a given timeframe.
- Workflow Manager, to see running EDAWs and manually stop them.

As an additional management Dashboard, we use the Ambari Dashboard to inspect the infrastructure status, updates tools or libraries, handles computation nodes, to name but a few.

2.8 API Module

To interact with the rest of the EVOTION components, the BDA Engine exposes RESTFUL APIs. These APIs permit to cover all the above functionalities/APIs exposed by the BDA Engine sub-components, following the EVOTION integration guideline. The API module exposes also internal component's functionalities that require specific authorization to be used. These internal functionalities are exposed to be consumed mainly by the BDA Dashboard for management. Authentication and Authorization JWT token is adopted for every interaction with the APIs to avoid unwanted/malicious interaction. Since it is used in all the APIs, we omitted it as parameters in the following API descriptions. The list of APIs with details are presented in the following, while the RESTFUL version is available in the implementation Section 3.

id AddWorkflow(EDAW, params)

AddWorkflow	It permits to add an EDAW into the Workflow Catalogue.	
Input parameters		
Name	**Type**	**Description**
EDAW	Structured type.	It is the Workflow Catalogue entry including executable/orchestrated workflow as provided by the PHPDM Transformation tool, scheduling preferences, formal parameters, etc.
Params	List of Structured types representing each Tasks' actual parameters.	It contails all the requested parameters for each Task in the EDAW. The EDAW is structured to take parameters at Task level.
Output parameters		
Name	**Type**	**Description**

Id	Numeric	It represents the id of the loaded EDAW as stored in the Workflow Catalogue

<list of EDAWs> FindWorkflowModel(DAWID)

FindWorkflowModel	Finde all the EDAW referred to a given PHPDM Model Instance Data Analytic Workflow (DAWID) from the Catalogue.	
Pre-conditions	DAWID must refers to an existing Model Instance in the Ontology Manager.	
Input parameters		
Name	**Type**	**Description**
DAWID	String	Unique identifier for a DAW Model Instance
Output parameters		
Name	**Type**	**Description**
list of EDAWs	List of Structured type	The list of EDAWS that refer to a specific DAWID in the Catalogue

EDAW FindWorkflowInstance(EDAWID)

FindWorkflowInstance	Find a given EDAW from the Catalogue.	
Pre-conditions	EDAWID must refers to a loaded EDAW.	
Input parameters		
Name	**Type**	**Description**
EDAWID	numeric	Unique identifier for a loaded EDAW
Output parameters		
Name	**Type**	**Description**
EDAW	Structured type	The EDAW as it is in the Catalogue

ret ModifyWorkflow (EDAW,params)

ModifyWorkflow	It permits to modify a given EDAW, like for instance updating the formal parameters, the schedule preference , the EWAD code, of a loaded EDAW. It allows to modify also the actual parameters.	
Pre-conditions	The EDAW must be already loaded into the Catalogue. If the EDAW is running, it is first stopped, modified and then re-executed.	
Post-conditions	Ret corresponds to a valid output message	
Input parameters		
Name	**Type**	**Description**
EDAW	Structured type.	It is the Workflow Catalogue entry including

		executable/orchestrated workflow as provided by the PHPDM Transformation tool, scheduling preferences, formal parameters, etc.
Params	List of Structured types representing each Tasks' actual parameters	It contails all the requested parameters for each Task in the EDAW. The EDAW is structured to take parameters at Task level.
Output parameters		
Name	**Type**	**Description**
Ret	boolean	Positive in case of success, negative in case of error (e.g., EDAW is not available in the Catalogue)

ret RemoveWorkflowModel(DAWID)

RemoveWorkflowModel	Remove all the EDAW referred to a given PHPDM Model Instance Data Analytic Workflow (DAWID) from the Catalogue.	
Pre-conditions	DAWID must refers to an existing Model Instance in the Ontology Manager.	
Post-conditions	Ret correspond to a valid output message.	
Input parameters		
Name	**Type**	**Description**
DAWID	String	Unique identifier for a DAW Model Instance
Output parameters		
Name	**Type**	**Description**
Ret	boolean	Negative if the given DAWID is not present in the Catalogue, positive otherwise. boolean

ret RemoveWorkflowInstance(EDAWID)

RemoveWorkflowInstance	Remove a given EDAW from the Catalogue.	
Pre-conditions	EDAWID must refers to a loaded EDAW.	
Post-conditions	Ret correspond to a valid output message.	
Input parameters		
Name	**Type**	**Description**
EDAWID	numeric	Unique identifier for a loaded EDAW
Output parameters		

Name	Type	Description
Ret	boolean	Negative if the given EDAWID is not present in the Catalogue, positive otherwise. boolean

ExecuteWorkflow(EDAWID)

ExecuteWorkflow	Execute a loaded EDAW given its EDAWID according to the scheduler preference. It can be also used to force re-execution of an EDAW coherently with the schedule preference.	
Pre-conditions	EDAWID must refers to a loaded EDAW	
Post-conditions	The Job id given by the Workflow Scheduler is saved into the Workflow Catalogue entry related to EDAWID to manage the running/scheduled EDAW.	
Input parameters		
Name	**Type**	**Description**
EDAWID	numeric	Unique identifier for a loaded EDAW

ret StopWorkflow(EDAWID)

StopWorkflow	Stop the execution of an EDAW and perform the relative rollback activities if required.	
Pre-conditions	EDAWID must refers to an EDAW under execution	
Post-conditions	Ref correspond to a valid output message	
Input parameters		
Name	**Type**	**Description**
EDAWID	numeric	Unique identifier for loaded EDAW
Output parameters		
Name	**Type**	**Description**
Ret	boolean	Positive if the EDAWID can be stopped, negative otherwise.

id AddTask (ImplDAWTask)

AddTask	It allows to add an implementation of a given DAW Task (ImplDAWTask) into the Task Catalogue.	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed.	
Input parameters		
Name	**Type**	**Description**
ImplDAWTask	A structured Type	It represents an entry of the Task Catalogue
Output parameters		

Name	Type	Description
Id	numeric	It represent the id of the Implemented Task as it is inserted into the Task Catalogue

Ret ModifyTask (ImplDAWTask)

ModifyTask	It allows to modify any of the Task related attributes, like pointing to different implementation, modifying the reference library to name but a few.	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed. ImplDAWTask must refers to an Implemented Task available in the Task Catalogue.	
Input parameters		
Name	**Type**	**Description**
ImplDAWTask	Structured Type	It represents the entry of the Task Catalogue for which we want to modify some of its attributes
Output parameters		
Name	**Type**	**Description**
Ret	boolean	Positive in case of successful modification, negative in case of error (e.g., EDAW not available in the Catalogue)

Ret RemoveTaskModel (DAWTaskID)

RemoveTaskModel	It allows to remove all the implementation of a given a DAW task model	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed. DAWTaskID must refer to a DAW Task present in the EVOTION Ontology Manager	
Input parameters		
Name	**Type**	**Description**
DAWTaskID	String representing a DAW Task ID in the Ontology Manager.	It is an unique identifier referring to a Class in the Ontology of the PHPDM language.
Output parameters		
Name	**Type**	**Description**
Ret	boolean	Negative if the given DAWTaskID is not present

		in the Catalogue, positive otherwise

RemoveTaskInstance (ImplDAWTaskID)

RemoveTaskInstance	It allows to a specific implementation of a given Task.	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed. ImplDAWTaskID must refer to an Implemented Task available in the Task Catalogue.	
Input parameters		
Name	**Type**	**Description**
ImplDAWTaskID	Numeric. It represents an Implemented DAW Task. It is optional.	It represents the entry of the Task Catalogue. If present the deletion refers just to this ID otherwise to all the implemented Task related to the DAWTaskID
Output parameters		
Name	**Type**	**Description**
Ret	boolean	Negative if the given ImplDAWTaskID is not present in the Catalogue, positive otherwise

<ImplementedTasks> FindTask(DAWTaskID)

FindTask	It permits to search for an implementation of a given DAW Task ID. It returns the details needed to generate an EDAW involving the retrieved Implemented Task.	
Input parameters		
Name	**Type**	**Description**
DAWTaskID	A String representing a DAW Task ID in the Ontology Manager.	It is an unique identifier referring to a Class in the Ontology of the PHPDM language.
Output parameters		
Name	**Type**	**Description**
ImplementedTasks	List of Implemented Tasks as they are specified in the Task Catalogue (i.e., the list of corresponding Catalogue Entries)	It contains the Implemented Analytic Tasks' details relative to a given DAW Task ID.

JobID AddEDAWJob (EDAWID)

AddEDAWJob	It adds an EDAW to the schedule according to the preference.	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed. EDAWID must refer to an EDAW available in the Workflow Catalogue.	
Input parameters		
Name	**Type**	**Description**
EDAWID	Numeric Type representing an EDAW	It represents the entry of the Workflow Catalogue for which we want set up a schedule.
Output parameters		
Name	**Type**	**Description**
JobID	Job identifier	It represents the identifier related to the EDAW in execution as it is represented by the Execution Manager/Scheduler

RemoveEDAWJob (EDAWID)

RemoveEDAWJob	It removes an EDAW from the scheduler. If the EDAW is in running, it stops the EDAW job before the removal.	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed. EDAWID must refer to an EDAW available in the Workflow Catalogue.	
Input parameters		
Name	**Type**	**Description**
EDAWID	Numeric Type representing an EDAW in the Workflow Catalogue	It represents the entry of the Workflow Catalogue for which we want to modify a schedule.

<EDAWID> ListRunningEDAWJob

ListRunningEDAW	It lists all the EDAW that are running.	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed.	
Output parameters		
Name	**Type**	**Description**
<EDAWID>	List of EDAWID	It contains the list of all the EDAWID that are running

StopRunningEDAWJob(EDAWID)

ListRunningEDAW	It stops specific EDAWID.	
Pre-conditions	This API is for internal use only and it requires BDA Administration authorizations to be executed. EDAWID must refer to a running EDAW.	
Input parameters		
Name	**Type**	**Description**
EDAWID	Numeric Type representing an EDAW in the Workflow Catalogue	It represents the entry of the Workflow Catalogue wich is under execution and for which we want to stop the execution.

3. Security and privacy

ENISA Threat Landscape[17] provides a complete overview of the security concerns of a BDA architecture. Two aspects need to be carefully evaluated in terms of security and privacy: the BDA architecture and the analytic processing environment. Concerning the architecture, by design, the EVOTION BDA is a single-tenant system[18], therefore we don't have to deal with security problems introduced by multi-tenancy, and it fully adopts the mechanisms defied for the overall EVOTION platform network protection, authentication and authorization.

Considering confidentiality applied to BDA data analytic processes, it is still an open challenge. This is because although there is significant ongoing research on techniques supporting searching and reporting on encrypted data, like functional encryption and homomorphic encryption, such techniques cannot support practical big data analytics. However, such considerations are mainly focused on Big Data offered as a service. EVOTION is a vertical application of Big Data where the BDA is a subsystem mostly interacting via controlled API. Data are modelled and anonymized before entering the BDA, reducing confidentiality requirements after data ingestion. In addition, the BDA Engine is not in charge of security and privacy requirements of the workflows and the analytics, which are demanded to other EVOTION subsystems (e.g., PHPDM Specification and Transformation tools). With regards to executable workflows and analytics, the role of the BDA Engine is limited to grant correctness, also in terms of non-functional property, of the implementation.

Considering integrity, it needs to be ensured across all processes run by the BDA Engine. Integrity of executable Tasks and Workflows may be enhanced through security-tagging. Every executable task and workflow arriving from a specified data source could be tagged with a tamper proof security tag identifying its producer and the time of production. However, we note that this is a process involving the Infrastructure as a whole and not just the BDA itself.

Given the above general considerations, in this section, we present the threat model to be considered for the EVOTION BDA Engine and the corresponding objectives to be met.

3.1 Threat Model

The Big Data Architecture component could be subject to a number of threats coming, in general, from the access, network or physical, to its features.

T.MALICIOUS_APPS - Malicious or Flawed Application

In general, malicious or flawed applications pose a threat because of unauthorized features, embedded or resulting after an exploitation. Strictly speaking, the EVOTION BDA does not include applications nor is a general-purpose execution environment where rogue applications could be installed. However, the BDA makes use of libraries of functions, providing the building blocks for big data analytics, and computes workflows, as sequence of operations representing the building blocks of public health policies defined by EVOTION users, policy makers, in particular. Malicious or flawed libraries could replace legitimate ones and provide undocumented features for accessing or exfiltrating data, or for exploiting software and hardware resources of the BDA platform. For instance, a ransomware-like attack towards the BDA platform could

[17] https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/enisa-threat-landscape
[18] The BDA is not shared for different purposes but EVOTION.

potentially be carried out through a malicious library. Malicious workflows or tasks, instead, could be workflows/tasks with unauthorized sequences of operations, possibly compromising the security of EVOTION data or performing operations violating the intellectual property of the project, the purpose of the clinical trial, or deemed unethical.

The twos, libraries and workflows, represent software artefacts that should be protected from tampering, illegal modification, and unauthorized replacement. Therefore, for the specific case of the BDA, the generic T.MALICIOUS_APPS threat class should be interpreted as possibly malicious or flawed library or workflow/task.

T.NETWORK_ATTACK - Network Attack

The EVOTION BDA has no direct network connection outside the EVOTION architecture, so it is not directly exposed to network-based threats coming from third parties or users. It has a number of network connections with EVOTION components through APIs. From a network perspective, then, threats for the BDA may originate if an attacker is able to gain access to internal, component-to-component, communication or is able to impersonate a fake EVOTION component.

T.PHYSICAL Physical Access

Threats from an unauthorized physical access to the BDA platform may include the disruption of hardware assets, tampering with the execution environment (operating system and file system), the upload or modification of critical BDA software components. The T.PHYSICAL threat class could realize the conditions required by T.MALICIOUS_APPS threats.

T.PERSISTENT - Persistent Presence

The persistent presence of an attacker on the BDA platform would have the consequence that the whole EVOTION functionality is possibly compromised. If an attacker has full control of the BDA, then results of analytics and workflows cannot be no longer trusted. A possible motivation for such a scenario could be that of an attacker wishing to influence decisions by policy makers by manipulating the results of the clinical trial and the hints that the EVOTION platform would provide to policy makers for evaluating a public policy. Another scenario could be of an attacker wishing to exfiltrate all results of the activity of policy makers, compromising confidentiality. The motivation in this case could be to anticipate possible decisions of policy makers and gaining illegal advantages from this information. A third scenario, could be that of an attacker wishing to exfiltrate data from the repository and illegally trade them. This would be possible with the persistent control of the BDA not just to exfiltrate them in bulk, but mainly to leak data slowly, in order not to trigger possible security monitoring mechanisms or behavioural controls on the activity of the BDA upon the data (Anisetti at al. 2017).

3.2 Security Objectives and Functionalities

With respect to the threat classes just presented, the BDA development has met a number of corresponding objectives.

O.DATA_PROTECTION_TRANSIT - Protected Communications

Communication between the BDA and other EVOTION components carried over a private network are secured with TLS/SSL security protocols. The same security solution at network level is applied for communication between nodes of the distributed BDA architecture for analytics execution and data

processing. This addresses the threat of network attacks (i.e., [T. NETWORK_ATTACK]). To achieve this objective, we enabled SSL on each BDA services.

Furthermore, communications between EVOTION components and the BDA - i.e., with the Data Repository, the Ontology Reasoner, the PHPDM Transformation tool, the Frontend, and the DSS - is supported by mutual authentication of components, so to prevent MITM attacks or impersonation. This aspect will be covered during the Integration of the entire EVOTION Platform, currently all the components are ready to support this feature.

O.STORAGE - Protected Storage

The BDA does not provide typical storage features, because data processing is executed on the Data Repository and all the required security features is defined in the relative deliverable D5.2. However, the BDA has a local storage for complementary functionality. For instance, it keeps a copy of executable workflows organized in a catalogue and a task catalogue listing executable functions. These two local catalogues are needed for authorizing the execution of model instances. This local storage must be protected with respect to confidentiality, integrity and availability threats. Standard cryptographic features are adopted for the security of the local storage. In particular, we configure our MariaDB used for the Catalogues following the Center for Internet Security Benchmark[19]enabling at rest encryption. In addition we treat integrity of Workflows and Tasks at application level (see O.INTEGRITY - Component Integrity). Availability is guaranteed by the distributed file system for the Workflows and Tasks. We also note that the same protection is required for the learning models generated the BDA Engine.

O.AUTH - Authorization and Authentication

EVOTION Users do not have direct access to the BDA. The APIs exposed by the BDA in the EVOTION service architecture can only be accessed by authenticated and authorized components (Frontend, DSS, PHPDM Transformation tool). The same applies for components that the BDA accesses (Data Repository, Ontology Reasoner). Furthermore, entries of the Workflow and the Task Catalogues maintained locally by the BDA require authentication and authorization to prevent threats of the T.MALICIOUS_APPS class.

O.ACCOUNTABILITY - System Reporting

The BDA has features to generate logs of activity and executions, which may produce reports of activity. Anomalous behaviors can be identified enabling timely corrective actions or reconfiguration. Threats of the T.PERSISTENT class may be addressed through audits and anomaly detection actions. Activity logs produced by the BDA are protected against confidentiality, integrity, and availability attacks. Log collection at API level will be enabled during the integration thanks to specific log collection service. Internal BDA related logs are available as well but at component level to be used for auditing.

[19] https://www.cisecurity.org/cis-benchmarks/

O.APPLY_POLICY - Component Configuration

Similar to all EVOTION components, also the BDA is configured to accept security policies defined at project level in order to ensure protection of enterprise or personal data, stored and processed. Security policies cannot be unenrolled and updates could be accepted only if signed by the EVOTION central management facility.

O.INTEGRITY - Component Integrity

Integrity requirements for the BDA concentrate mainly on the Workflow and Task catalogues, which should be protected from possible tampering, manipulation, or unauthorized updates. Cryptographic mechanisms and hashing techniques are deployed for assuring that only legitimate workflows, as maintained by the Ontology Reasoner, are available, and only legitimate tasks (e.g., functions, libraries) can be executed during an analytic execution. Workflows and Tasks integrity is guaranteed at application level using hashing. In addition, we consider of primary importance to keep monitored the vulnerabilities of the libraries used to implement Tasks

O.PRIVACY - Component Privacy

Privacy requirements for the BDA do not involve data from the clinical trial, because data processing is carried out completely on the Data Repository. Privacy requirements are instead related to the operations of the BDA and the policy model for which it could be used. Log and audit trails must be kept confidential, as well as the Workflow and the Task Catalogues. Log and audit trails, if disclosed, may provide information about the policy models defined by policy makers and the simulations run through the BDA, which are sensible information with respect to the activity of public agencies and the possible social or economic relevance represented by having access to preliminary work carried out by public policy makers. The Workflow and the Task Catalogue, instead, may reveal intellectual properties of the EVOTION project. Learning models in addition may provide the room for privacy leakages being able to exercise them appropriately.

Task and Catalogue, as well as learning models are kept in a confidential location within the HDFS file system as well as logs that resides on an encrypted file system. Authentication and authorization is provided at API level.

As a general consideration, a number of security services can be configured within the Apache ecosystem, like for instance Kerberos for authentication and authorization, Ranger, Knox, to name but a few. However, as emerged by the previous threat-objective analysis, they are not strictly required for the EVOTION BDA Engine, which is an internal API wrapped component. We will investigate the utility to have them included in the BDA during the integration process, as a more holistic view on security and privacy.

3.3 BDA Engine Privacy requirements

Table 7 lists BDA specific relevant requirements as they are presented in D9.3. plus details on how our BDA addresses them.

Table 7: List of new functional requirements specifically defined for the BDA.

Generic	GDPR requirement	Action	BDA
G-REQ11	10: Security	EVOTION platform MUST treat structured data confidentially	The Catalogues are kept confidential as well as Workflows and Tasks using encryption at rest
G-REQ12	3: Aggregation of data 4: Local de-identification and anonymization	EVOTION platform MUST keep the privacy and security of structured data	The data are pseudo anonymized a priori. The data produced by the BDA is stored in the repository exploiting the Repository privacy and security features. Tasks and Workflows are protected via Confidentiality and integrity measures
G-REQ13	5: Authentication 6: Authorization	EVOTION platform MUST allow access to structured data only to authorized person	BDA API requires Authentication/Authorization token provided by the Security Module as the rest of EVOTION Components
G-REQ14	10: Security	EVOTION platform SHOULD not to keep data saved locally in a permanent buffer	While processing we avoid the use of permanent buffers outside the repository.
G-REQ15	2: Separation of data 5: Authentication 6: Authorization	EVOTION platform MAY provide different level of data obfuscation depending on the level of authorization	BDA supports obfuscations via pre-processing of data.
G-REQ16	10: Security 5: Authentication 6: Authorization 8: Transparency	EVOTION platform MAY provide a “brake the glass scenario” for emergency situation.	Emergency situation can be handled using the backend Dashboard requiring administrative access.
G-REQ17	10: Security 5: Authentication 6: Authorization 11: Accountability	EVOTION platform SHOULD provide CRUD interface and API preserving privacy and security	The API are implemented to preserve privacy and security. They use authentication/authorization token and encrypted channel. They are defined to fulfil privacy by design, for instance, not offering inspections that are not permitted. While integrated the security features will be enhanced.
G-REQ26	10: Security	EVOTION BDA MUST process all the date confidentially	The processing of the data is kept confidential, still internal to the BDA Engine, no external services are used and no data are saved locally. The only available information are notifications of percentage of completeness and the intermediate and final results available and protected at Repository level.

G-REQ27	1: Minimize personal data 8: Transparency	EVOTION BDA MUST provide adequate level of privacy	The BDA is a mere executor, the model are designed to be privacy preserving and the transformation into an executable workflow will maintain this property. In addition no local data are stored outside the Repository as well as no external untrusted libraries is used.
G-REQ28	10: Security 1: Minimize personal data 3: Aggregation of data	EVOTION BDA SHOLD not keep local copies of the data unprotected or permanent	No permanent copies of data or semi processed data are saved locally.
G-REQ29	8: Transparency	EVOTION BDA SHOULD follow the transparency regulation while computing analytics	The analytics execution can be monitored thanks to the logs collected at API level developed in the integration framework
G-REQ30	10: Security	Integrity of ALL the data type both at rest or in transit SHOULD be guarantee	At BDA level the integrity of Tasks and Workflows are guaranteed by hash integrity mechanisms. We enabled the security features available for the ”wire Encryption”
G-REQ31	1: Minimize personal data 3: Aggregation of data 4: Local de-identification and anonymization	ALL components when delete sensible data SHOULD perform the deletion securely avoiding data scavenging.	All the deletion of Tasks and Workflows will be treated using secure deletion.
G-REQ32	11: Accountability 2: Separation of data	Data retention policy MUST be satisfied by all the components storing data	Workflows can be associated to data retention policy while tasks are functionalities that cannot be removed
G-REQ33	7: Monitoring data access	EVOTION as a whole MAY adopt continuous compliance monitoring approach supporting auditability	We adopted Moon Cloud and Ranger as a way to monitor che compliance to standards and regulations such as GDPR
G-REQ34	3: Aggregation of data 4: Local de-identification and anonymization 8: Transparency	EVOTION analytical models SHOULD be design to preserve privacy during analysis	BDA engine is the executor. With this regards it will not introduce any privacy leaks while executing an analytics. This requirements mainly effects PHPDM models and PHPMD Transformation Tool

4. Implementation

In the following we describe implementation details regarding the BDA Engine, focusing on the technology adopted and the development decisions taken. Additional implementation details at workflows and task level are available in Appendix A.

4.1 BDA Infrastructure

The BDA has been deployed on top of a virtualization layer enabling the cluster of processing and storing nodes. The deployment is based on Ambari version 2.6.1.5 and the list and versions of the deployed services are listed as follows.

- HDFS 2.7.3
- YARN 2.7.3
- MapReduce2 2.7.3
- Hive 1.2.1000
- Hive 1.1.2
- HBase 0.16.0
- Pig 1.4.6
- Sqoop 4.2.0
- Oozie 4.2.0
- Spark 1.6.3
- Spark2 2.2.0
- Zeppelin 0.7.3
- Kafka 0.10.1

We decided not to have the latest versions but the most stable one providing the functionality required.

One specific virtual machine is dedicated to the Workflow and Task Catalogues, API module and Backend Dashboard.

4.2 Task Catalogue

Task Catalogue had been developed in Java version 1.8, using MVC design patterns where, model and controller are dedicated to the data validation, task generation and insertion.

We implement the Catalogue storage using a local BDMS (MariaDB version 5.5.56). We store the Implemented Analytic Task in the HDFS filesystem.

In the following an example of K-Means Task implemented in Java using Spark Mlib.

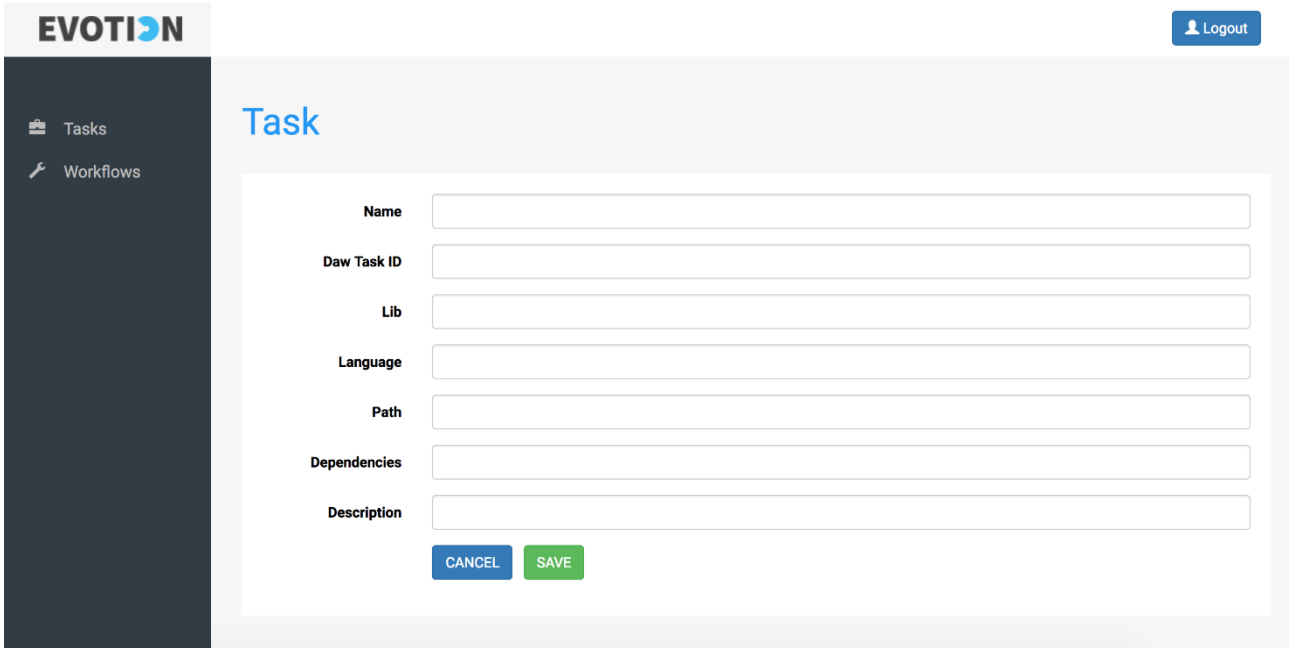
```
package it.unimi.evotion.tasks;

import java.io.File;
import java.io.IOException;
import java.io.Serializable;

import org.apache.commons.io.FileUtils;
…
…

public class Kmeans implements Task, Serializable {
            private static final long serialVersionUID = 1L;
            private JavaRDD<Vector> parsedData;
```

```
private KMeansModel clusters;
private RDD<Vector> parsedRdd;
public static boolean isVector(String s) {
            String[] sarray = s.split(",");
            try {
            Double.parseDouble(sarray[0]);
            return true;
            } catch (Exception e) {
            return false;
            }
}
public static Vector asVector(String s) {
            String[] sarray = s.split(",");
            double[] values = new double[sarray.length];
            for (int i = 0; i < sarray.length; i++) {
            values[i] = Double.parseDouble(sarray[i]);
            }
            return Vectors.dense(values);
}
static void deleteDir(String dir) {
try {
FileUtils.deleteDirectory(new File(dir));
} catch (IOException e) {

}
}
static double[] concat(double[] u, double[] v) {
            double[] r = new double[u.length + v.length];
            int i = 0;

            for (double a : u)
            r[i++] = a;
            for (double a : v)
            r[i++] = a;

            return r;
}
private static String toString(double[] a) {
            if (a == null || a.length == 0)
            return "";
            StringBuffer sb = new StringBuffer();
            sb.append(a[0]);
            for (int i = 1; i < a.length; ++i)
            sb.append(",").append(a[i]);
            return sb.toString();
}
public void init(Object... params) {
            SparkContext sc = ((SparkSession) params[0]).sparkContext();

            JavaRDD<String> data = sc.textFile((String) params[1],
1).toJavaRDD();
            data = data.filter(s -> isVector(s));
            parsedData = data.map(s -> asVector(s));
}
public void run(Object... params) {

            int numClusters = (Integer) params[0];
```

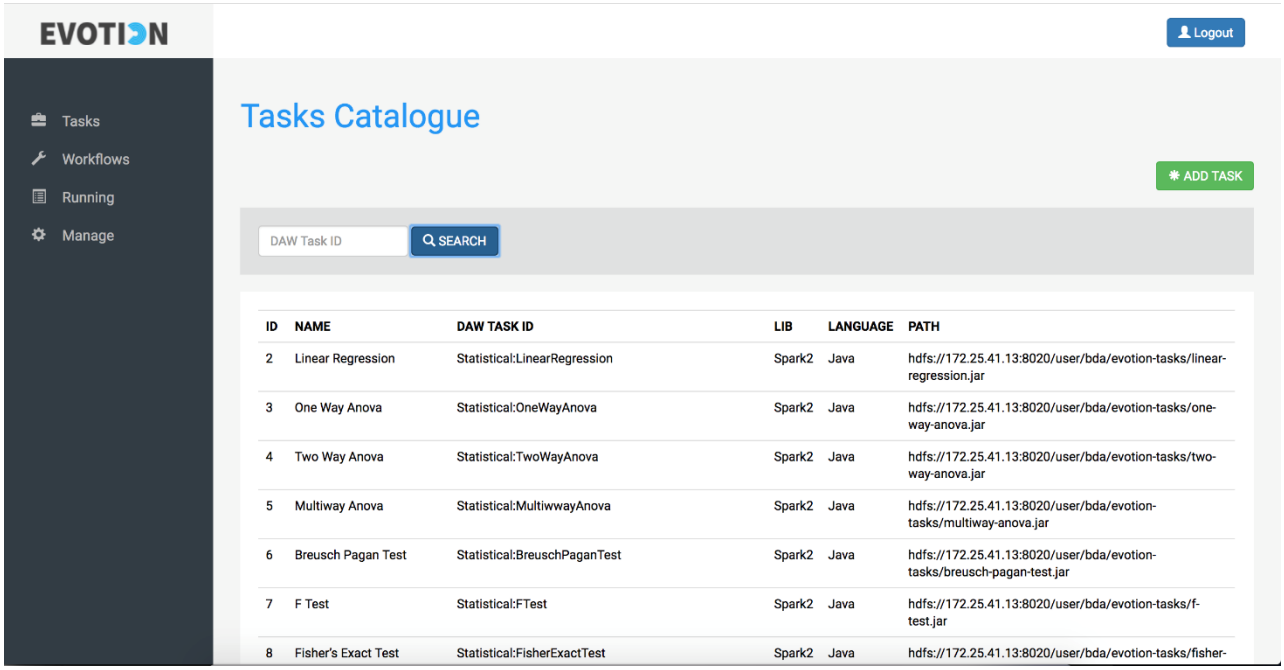
```
                int numIterations = (Integer) params[1];
                parsedRdd = parsedData.rdd();
                clusters = KMeans.train(parsedRdd, numClusters, numIterations);
        }
        public void postProcessing(Object... params) {
                JavaRDD<Object> objCluster = clusters.predict(parsedRdd).toJava
        RDD();
                JavaRDD<Vector> vctCluster = objCluster.map(o -> {
                return Vectors.dense(((Integer) o).doubleValue());
                });

                JavaRDD<Vector> inCluster = parsedData.zip(vctCluster).map(t ->
                {
                return Vectors.dense(concat(t._1().toArray(), t._2().toArray())
                );
                });
                JavaRDD<String> csvout = inCluster.map(v -> {
                double[] a = v.toArray();
                return toString(a);
                });
                csvout.saveAsTextFile((String) params[1]);
                ((SparkSession) params[0]).close();
        }

public static void main(String[] args) {
        SparkSession spark =
        SparkSession.builder().appName("Kmeans").getOrCreate();
        deleteDir(args[1]);
        Kmeans kmeans = new Kmeans();
        kmeans.init(spark, args[0]);
        kmeans.run(Integer.parseInt(args[2]), Integer.parseInt(args[3]));
        kmeans.postProcessing(spark, args[1]);
        }
}
```

The provided example of implemented Task follows the Task Wrapper interface and ingest data from a csv file from the file system in the Init method, execute the K-means and format the output again as csv file in file system.

In the following, we provide a screenshot of the Management Dashboard showing the form to add a Task in the Catalogue.



In the following screenshot, the list of available Implemented Tasks.



4.3 Workflow catalogue

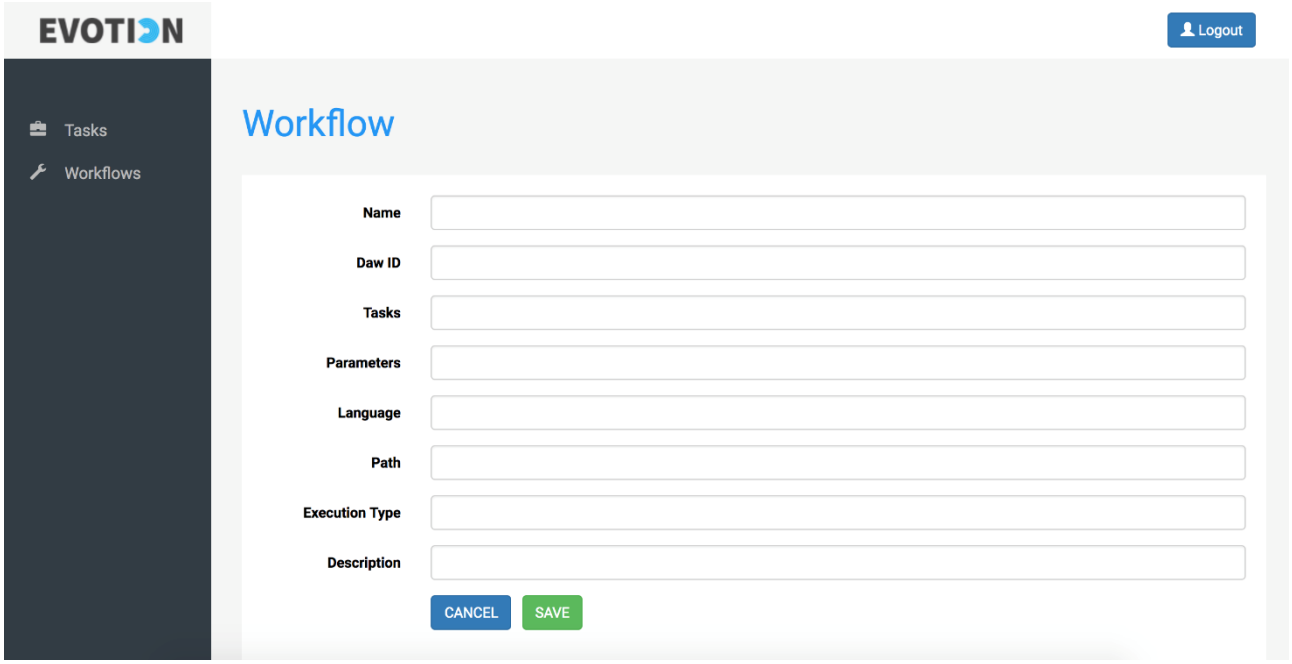
Similar to the Task Catalogue, the Workflow Catalogue has been developed in Java version 1.8, using MVC design patterns where, model and controller are dedicated to the data validation, Workflow generation and insertion. The catalogue storage is implemented again using a local BDMS (MariaDB version 5.5.56). We store the EDAW in the HDFS filesystem so that they can be executed when needed.

In the following, an example of Oozie EDAW for the execution of PCA on a dataset followed by a K-Means on the selected features. This EDAW is scheduled monthly started from the 30th or April 2018 until 30th of June 2018.

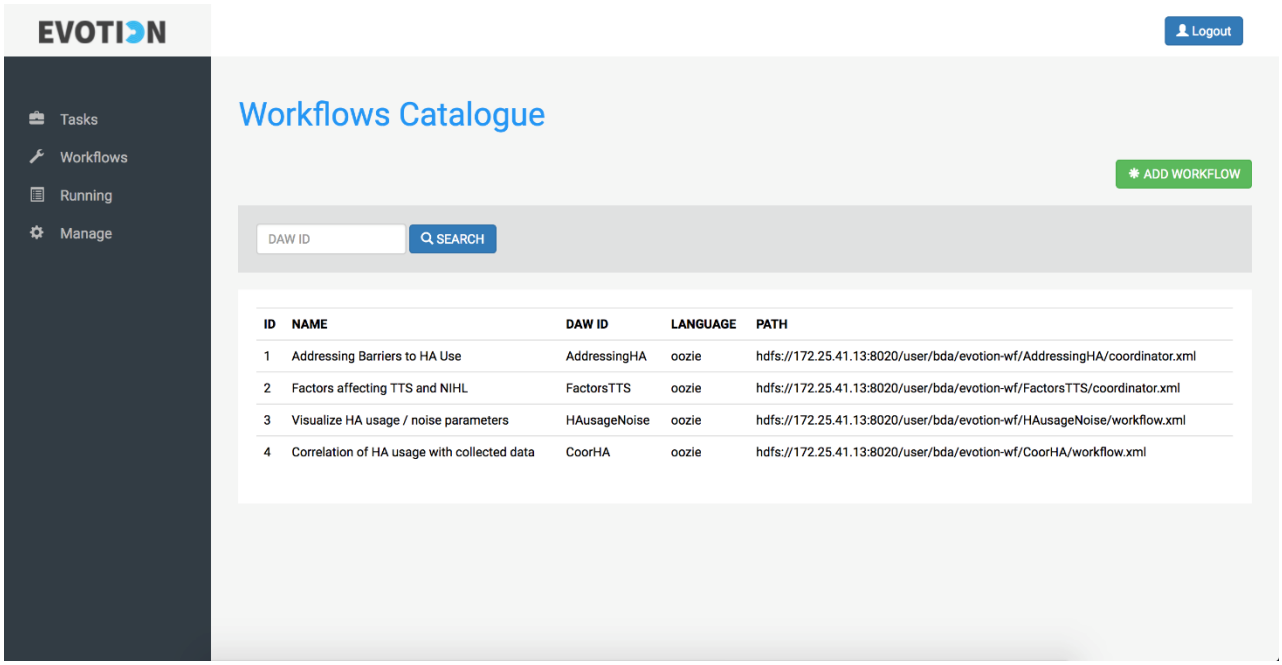
```
<coordinator-app name="Periodic" frequency="${coord:months(1)}" start="2018-04-30T00:03Z" end="2018-06
-30T00:04Z" timezone="UTC"
xmlns="uri:oozie:coordinator:0.5">
<action>
<workflow-app name="Cluster_ENVI"
  xmlns="uri:oozie:workflow:0.5">
  <start to="PCA"/>
  <action name="PCA">
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>Spark_PCA</name>
      <jar>/user/bda/evotion-Task/PCATask.jar</jar>
      <arg>${Num}</arg>
      <arg>${ENVI}</arg>
    </spark>
    <ok to="K-Means"/>
    <error to="kill"/>
  </action>
  <action name="K-Means">
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>Spark_Kmeans</name>
      <jar>/user/bda/evotion-Task/KMeansTask.jar</jar>
      <arg>${K}</arg>
      <arg>${maxiter}</arg>
    </spark>
    <ok to="end"/>
    <error to="kill"/>
  </action>
  <kill name="kill">
    <message>${wf:errorMessage(wf:lastErrorNode())}</message>
  </kill>
  <end name="end"/>
</workflow-app>
</action>
</coordinator-app>
```

Oozie workflows will be produces by the PHPDM Transformation tool in order to satisfy stakeholders needs. Oozie is also used for pre-defined Workflows, in that case they can be defined using the Management Dashboard workflows definition tool and then added to the BDA using the Workflow Catalogue.

In the following a screenshot of the Dashboard related to adding a new Workflow. We note that for Policy Maker workflows the PHPDM Transformation Tool interact directly using the API while the form is just to insert pre-defined workflows manually.



In the following the list of the available workflows in the Catalogue.



4.4 Workflow Manager

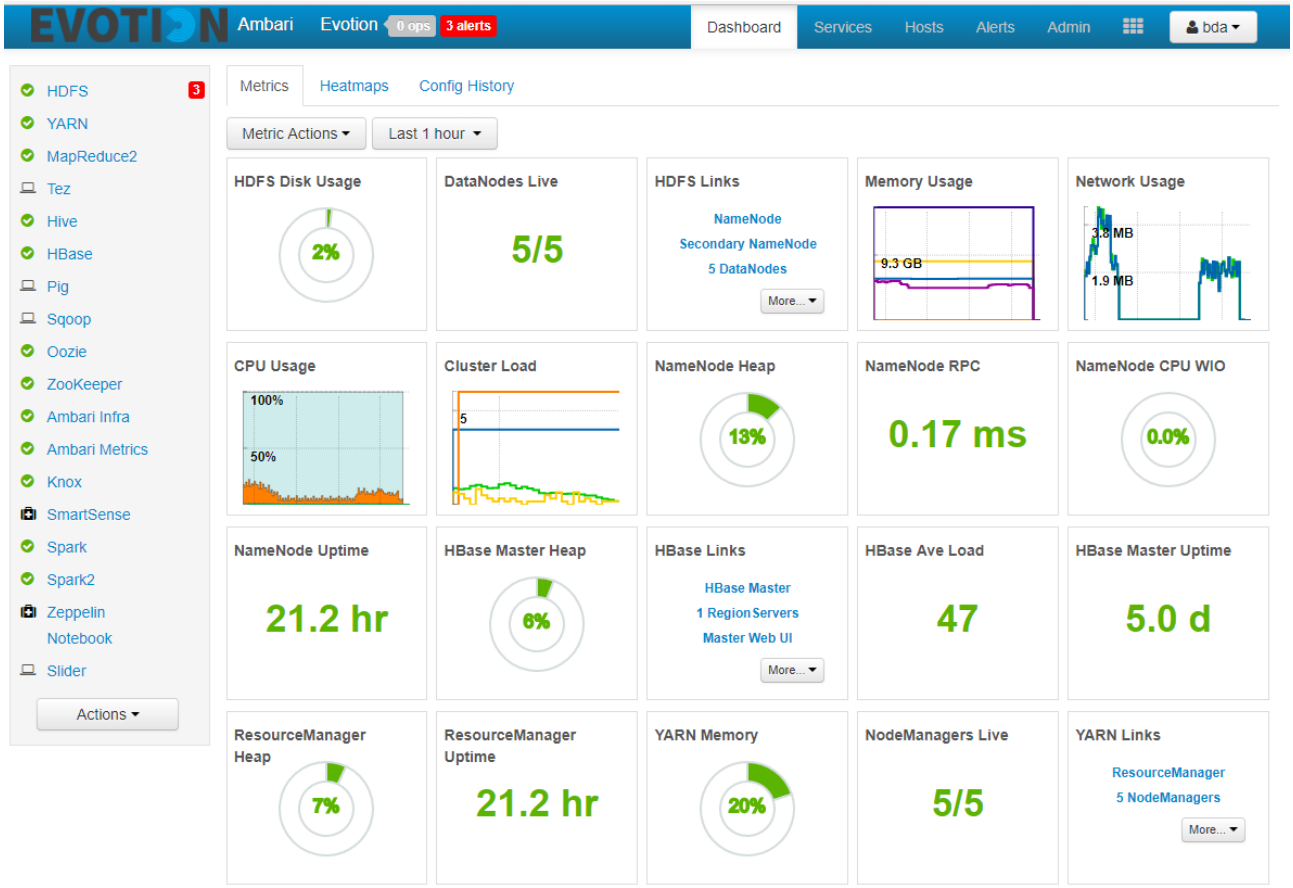
Workflow Manager is implemented based on Oozie API (v4.2)[20], which permits to schedule Workflows and monitor their execution. When completed, the results of a given Workflow are saved into the EVOTION Repository with a table name including the DAW, EDAW, and Oozie Job ID. The Workflow Manager handle all the possible state of an Oozie Job (i.e., a workflow in execution), and the relative commands like submit to add a workflow to the schedule without starting it, start to start it, kill to terminate and remove from the schedule etc.

[20] https://oozie.apache.org/docs/4.2.0/WebServicesAPI.html

4.5 Backend Dashboard

The Backend Dashboard is based on Java JSP web application with JQuery and AJAX for API calls to EVOTION components like the security manager for authentication/authorization and BDA Engine to access the Catalogues.

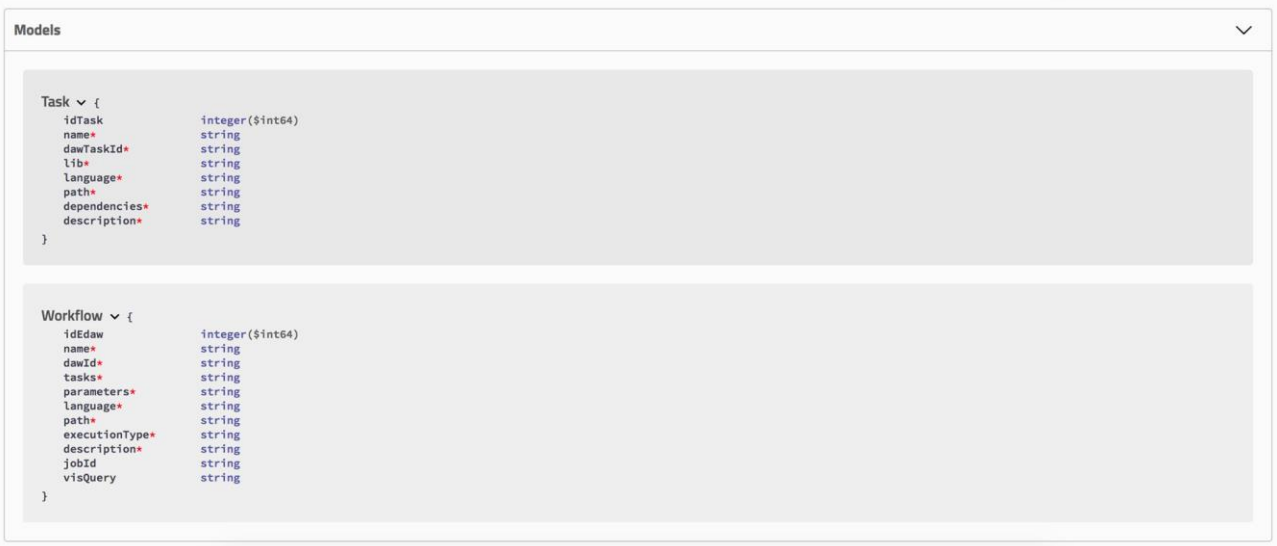
It includes a link to the Infrastructure Administrative Dashboard which is based on Ambari (screenshot in the following)



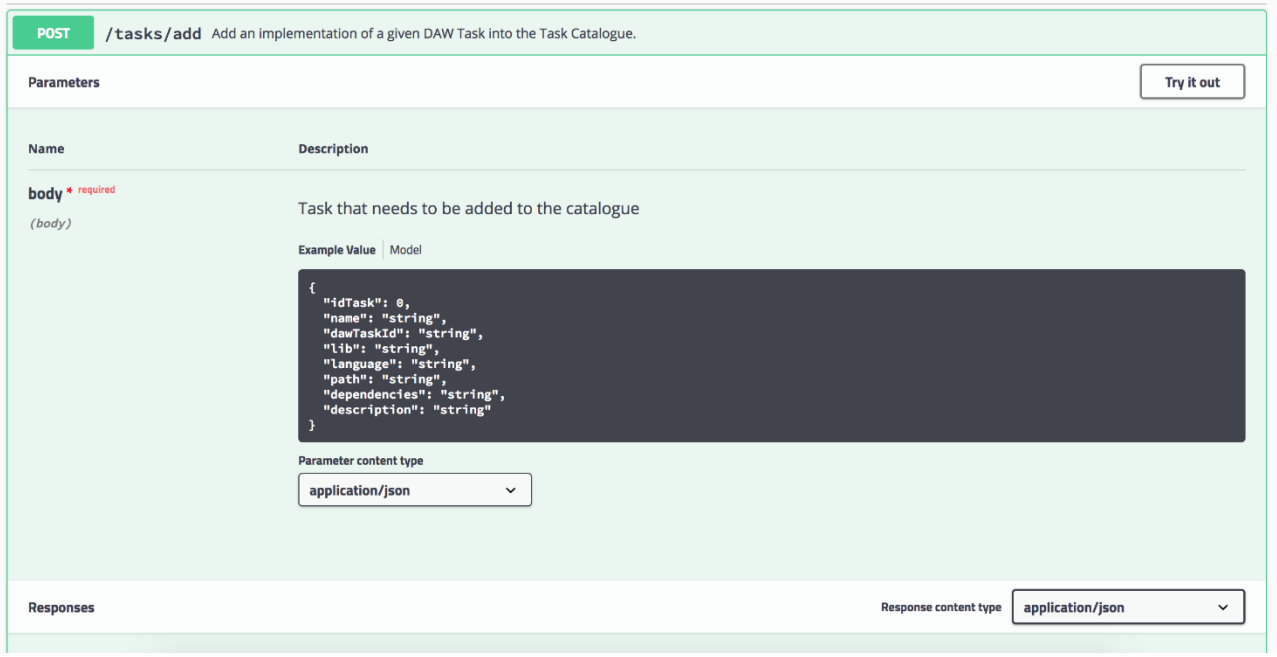
4.6 API Module

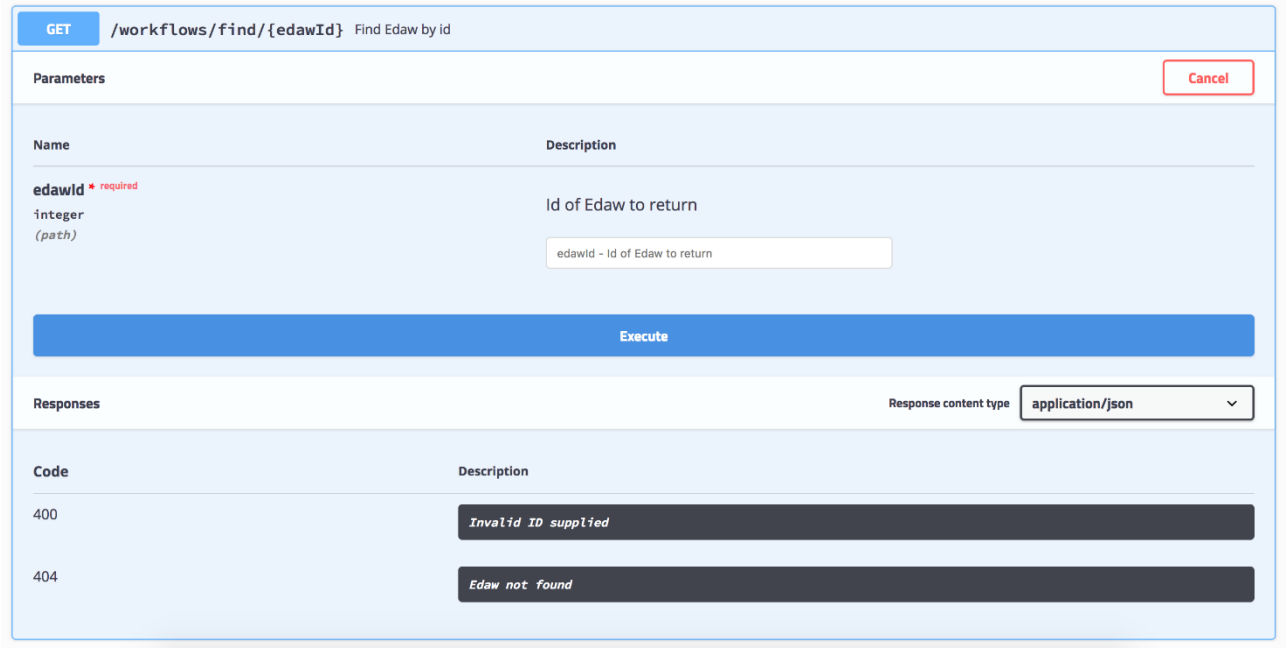
The API Module is developed using Java Jersey (JAX-RS 2.1/Jersey 2.26) input and output parameters are in JSON format extension for Jersey. At each call, input and output parameters are validated, authentication and authorization are enforced. In the following we provided some screenshots of our REST API implementation using swagger. More specifically the set of APIs for Tasks and the set of APIs for Workflow.

In the following we also provide the models as they are in swagger.



In the following for completeness we also provide screenshot showing the API related to add an implementation of a given task that in REST is a post, and find a specific workflow in the catalogue, that is a get according to REST paradigm.





Swagger API provided documentation with is of a great help for integration and as a way to keep API interface under control.

5. Use cases

For EVOTION Stakeholders, interacting with the BDA is possible through the Dashboard, which might be used to trigger computations. In the following, we present some relevant use cases with interactions with related EVOTION components.

5.1 Policy making EDAW Use cases

The overall purpose of the following use cases is to support Policy Makers in triggering Big Data Analytics Workflows during the policy making process.

USE CASE NAME: Execute a given EDAW
ACTOR: Policy Maker via Dashboard
PRECONDITIONS: The Policy Maker interacts with the Dashboard and the PHPDM Specification tool to define a PHPDM model instance. This model instance contains a DAW expressing a Workflow to be executed.
POSTCONDITIONS: The results of an EDAW are written in the Repository and could be either accessed by a Policy Maker or further elaborated by the DSS. They can also be the input for subsequent EDAWs
FLOW OF EVENTS
1. The DAW is transformed by the PHPDM Transformation tool into EDAW (a) Interact with the BDA Engine searching for available implementation for all requested tasks (FindTask()). (b) In case of positive transformation an EDAW is generated. 2. The PHPDM Transformation tool ask the BDA to load the new EDAW (AddWorkflow), together with the scheduling preferences and parameters for each of the involved Tasks. 3. It receives back the ID of the EDAW identifying it into the Workflow Catalogue. 4. The BDA Engine adds the EDAW to the schedule, according to the preference and using the EDAW ID (AddEDAWJob). 5. During the execution, the EDAW notifyies the Dashboard about each step completion and contacts the Ontology Manager to update the model with references to intermediate/final results. (a) The BDA Engine stores intermediate and final results into the Repository and communicates the link where the results are available.

Notes:

(a) This use case takes place during the process of producing a PHP.
(b) This step takes place through the Dashboard.
(c) At each schedule, the EDAW execution starts from step 5.
(d) Storage format or the EDAW output is dictated by the PHPDM Model Instance

USE CASE NAME: Stop a running EDAW
ACTOR: Policy Maker via Dashboard
PRECONDITIONS: The EDAW to be stopped must be running, otherwise no effect is produced
POSTCONDITIONS: When stopped, an EDWA executes a rollback and will be re-executed according to the scheduling
FLOW OF EVENTS
1. Via Dashboard, a Policy Maker can stop a running Analytic Workflow that she had previously lunched. 2. The Dashboard interacts with the BDA API to stop the execution of the specific EDAW (StopWorkflow). 3. Stopping an EDAW will not remove it from the scheduler, which will re-execute it according with scheduling preferences. (a) The API calls the Workflow Manager to stop the execution (StopRunningEDAWJob). 4. Rollback functions are executed to remove intermediate results 5. At the next scheduled execution, the EDAW will start as a new process

Notes:

(a) This step takes place through the Dashboard.

(b) Stopping complex and computational intensive Workflows is useful to free resources or to terminate unwanted execution

USE CASE NAME: Remove an EDAW from the Workflow Catalogue
ACTOR: Policy Maker via Dashboard
PRECONDITIONS: The EDAW to be removed must be in the Workflow Catalogue, otherwise the action has no effect
POSTCONDITIONS: The removal of an EDAW removes also all associated scheduled processes
FLOW OF EVENTS
1. Via Dashboard, a Policy Maker can remove a given EDAW which is no more needed. (a) EDAW is associated to a PHPDM model instance. 2. The Dashboard interacts with the API asking the deletion of the given EDAW from the Catalogue (RemoveWorkflowInstance). 3. If the EDAW is running, it is stopped first and then removed from the Catalogue and from the scheduler.

Notes:

(a) This step takes place through the Dashboard.

(b) Useful when some of the triggered analytics are no more useful for a specific PHPDM evaluation

USE CASE NAME: Remove all EDAWs relative to a specific DAW from the Catalogue
ACTOR: Policy Maker via Dashboard
PRECONDITIONS: The EDAWs to be removed must be in the Catalogue otherwise the removal has no effect
POSTCONDITIONS: The removal of the EDAWs removes also all scheduled processes associated with them
FLOW OF EVENTS
1. Via Dashboard, a Policy Maker asks to delete a certain DAW; 2. The Dashboard, interacting with BDA APIs, requires to remove all EDAWs from the Catalogue (RemoveWorkflowModel) corresponding to the DAW to be deleted (a) All EDAWs selected for removal should be stopped, if running, and then deleted from the Catalogue and the scheduler.

Notes:

(c) This step takes place through the Dashboard.

(d) Useful when all of the triggered analytics are no more useful for a specific PHPDM evaluation

(e) It implicitly stops all EDAWs relative to a given DAW and removes them from the Catalogue.

USE CASE NAME: Re-execute the EDAW with different parameters
ACTOR: Policy Maker via Dashboard
PRECONDITIONS: The EDAW must be listed in the Catalogue, otherwise the action produces no effect
POSTCONDITIONS: The modified EDAW substitutes the previous one in the Catalogue
FLOW OF EVENTS

1. Via Dashboard, a Policy Maker selects a previously executed DAW and triggers a new execution by specifying new parameters.
2. The Dashboard interacts with the API asking the modification of the given EDAW (ModifyWorkflow), for instance, by incrementing the maximum number of iterations relative to a k-means, or the number of clusters K
3. When the BDA receives the request, the corresponding EDAW is stopped if running, then the BDA:
    (a) Removes the EDAW from the schedule (RemoveEDAWJob)
    (b) Modifies the EDAW in the Catalogue as requested
    (c) Triggers re-execution of the modified EDAW (ExecuteWorkflow) according to the scheduled parameters (AddEDAWJob)

Notes:

(a) This step takes place through the Dashboard, but can be used by the PHPDM Transformation Tool to change the implementation of a given EDAW if needed
(b) It is used until the evaluation process converges to the final policy

USE CASE NAME: Add new Implemented Analytic Task
ACTOR: EVOTION BDA Administrator - Data Scientist via Backend Dashboard
PRECONDITIONS: The software library adopted in the task must be first installed in the BDA Infrastructure
POSTCONDITIONS: none
FLOW OF EVENTS
1. Let us consider the situation where, for supporting a given scenario, a specific Analytic Task is required but it is not yet listed in the BDA Task Catalogue. 2. The PHPDM Transformation tool interacts with the BDA Engine searching for an implementation of the specific Task (FindTask) 3. No Implementation is available (a) The DAW cannot be transformed into an EDAW until an implementation of the missing Task is provided. 4. EVOTION Data scientist, via Backend Dashboard, add an implementation of the Analytic Task (addTask) wrapped into Task Wrapper as requested for compatibility with EVOTION BDA. (a) The PHPDM Transformation Tool is now able to transform the PHPDM model into the relative EDAW

Notes:

(a) Administration activities to ensure the presence of all implemented tasks needed

5.2 Other use cases

For clinical and patient use cases, the interaction with the BDA is minimal because Analytics are pre-defined and only their parameters can be modified. Only the EVOTION Administrator can modify a pre-defined EDAW using the Backend dashboard. We remark that these use cases are expected to evolve during the project as the EVOTION platform integration proceeds.

USE CASE NAME: Visualize data
ACTOR: Clinicians via Dashboard
PRECONDITIONS: All available visualization capabilities must have a corresponding Zeppelin notebook
POSTCONDITIONS: The integration of Zeppelin in the EVOTION Dashboard
FLOW OF EVENTS
1. A Clinician accesses the Dashboard page where visualization is available: (a) Dashboard interacts with the BDA for retrieving the corresponding Zeppelin Notebook. (b) The Zeppelin notebook contains the features to plot the requested results.

Notes:

(a) This step takes place through the Dashboard.

(b) Zeppelin-based analytics are pre-loaded EDAW based on Zeppelin framework

USE CASE NAME: Visualize correlation of HA usage with collected data
ACTOR: Clinicians via Dashboard
PRECONDITIONS: The relative EDAW must be scheduled
POSTCONDITIONS: none
FLOW OF EVENTS
1. A Clinician interacts with the Dashboard page for this use case: (a) Dashboard interacts with the EVOTION Repository where the last evaluation results are stored. (b) BDA recomputes the results according to the scheduling preference. 2. If available through the Dashboard, the Clinician can ask for an immediate refresh: (a) This triggers re-execution of the relative EDAW (ExecuteWorkflow)

Notes:

(a) This step takes place through the Dashboard.

6. Demonstrator

This report is part of the D5.5 deliverable which also contains the source code of all the components developed that constitute the BDA Engine. A video showing the functionality of the administrator Backend Dashboard is also released. We remark that the dashboard will not be offered to the final user but just to the EVOTION super Administrator in order to handle Workflows and Tasks and to inspect and manage BDA jobs and infrastructure.

The video is available here: http://h2020evotion.eu/?ddownload=692

The source code is available here: http://h2020evotion.eu/?ddownload=699

The BDA Engine developed in this deliverable will be integrated at the CITY premises in order to provide access at consortium level as agreed in the consortium in accordance with the relative security and privacy considerations.

7. Conclusions

This report is part of the EVOTION deliverable D5.5 and describes the design and implementation of the EVOTION Big Data Architecture and the components interacting with it.

The BDA documented in this report represents a complete, pre-final deployed version of the EVOTION BDA engine. This version includes all architectural components for supporting the execution of workflows containing analytic tasks and directives derived from a PHP model instance specified as a DAW through the PHP Specification tool and translated into an EDAW by the PHPDM Transformation tool.

The features of the implemented EVOTION BDA includes also a tight integration with the EVOTION data repository and the APIs needed by other components in order to interact with the BDA. Mechanisms to support notification of ongoing analytic executions and the production of final results have been implemented, as well as notifications and visualization support for the EVOTION Dashboard to let policy makers and clinicians visualize the results and require new executions.

Further functionalities are to be included and extended in future developments as the project progresses towards a full integration. Extensions that have been planned are related to i) tasks to improve the set of available algorithms, ii) pre-defined workflows to support additional functionalities that may emerge after or during the integration, iii) security services in case they are needed while the BDA is integrated with the rest of the EVOTION platform.

This document is complemented by a video demonstrating some scenarios of execution and the backend to support the definition and deployment of Tasks and Workflows and an access link to the source code repository. These artefacts and this report constitute the deliverable D5.5.

References

Damiani, E., Ardagna, C., Ceravolo, P., and Scarabottolo, N. (2017, September). Toward Model-Based Big Data-as-a-Service: The TOREADOR Approach. In Advances in Databases and Information Systems (pp. 3-9). Springer, Cham.

Nikos Dimakopoulos, George Giotis, Panagiotis Kokkinakis, George Spanoudakis, Dimitris Kikidis, Thanos Bibas, Doris-Eva Bamiou, Giorgos Dritsakis, Ariane Laplante-Lévesque, Niels Pontoppidan, Luybov Trenkova Panagiotis Katrakazas, Manolis Stefanakis, Paschalis Papagrigoriou, Apostolos Economou, Dario Brdarić, Josip Milas, Ernesto Damiani, Francesco Zavatarelli, Fulvio Frati, Louisa Murdin, Nina Koloutsou, Mariola Sliwinska Kowalska, 2017. EVOTION stakeholders, scenarios and requirements, Deliverable D2.1 to the EVOTION-727521 Project funded by the European Union. ATC, Athens Greece

Ye, B., Spanoudakis, G., Prasinos, M., Dimakopoulos, N., Kokkinakis, P., Giotis, G., Papas, I., Papagrigoriou, P., Smyrlis, M., Stefanakis, M., Katrakazas, P., Koutsouris, D., Brdaric, D., Pandzic, Z., Salavarda, D., Urban, M., Milas, J., Pontoppidan, N.H., Trenkova, L., Kaloyanova, G., Tsokova, N., Anisetti, M., Bellandi, V., Cremonini, M., Damiani, E., 2017. EVOTION Architecture and Detailed Design, Deliverable D2.2 to the EVOTION-727521 Project funded by the European Union. CITY University, London, United Kingdom.

A. Savinov, Concept-oriented model: the Functional View, Eprint: arXiv:1606.02237 [cs.DB], 2016.

A. Savinov, Joins vs. Links or Relational Join Considered Harmful. Proc. IoTBD 2016, 362-368.

A. Savinov, From Group-By to Accumulation: Data Aggregation Revisited. Proc. IoTBDS 2017, 370-379.

A. Savinov, DataCommandr: Column-Oriented Data Integration, Transformation and Analysis. Proc. IoTBD 2016, 339-347.

M. Anisetti, C.A. Ardagna, V. Bellandi, M. Cremonini, E. Damiani, ``Privacy-aware Big Data Analytics as a Service for Public Health Policies in Smart Cities'', Sustainable Cities and Society, 2018

Marios Prasinos, George Spanoudakis, Ye Bin, Ioannis Basdekis and Andrew Smith, Panagiotis Katrakazas, Thelma Androutsou and Penelope Ioannidou, Marco Anisetti and Marco Cremonini, PHPDM Model Specification Language, Deliverable D4.1 to the EVOTION-727521 Project funded by the European Union. CITY University, London, United Kingdom.

M. Rathore, P. Anand, A. Ahmad, M. Anisetti, G. Jeon, ``Hadoop-based Intelligent Care System (HICS): Analytical Approach for Big Data in IoT'' ACM TOIT 2017

Fang, R., Pouyanfar, S., Yang, Y., Chen, S. C., & Iyengar, S. S. (2016). Computational health informatics in the big data age: A survey. ACM Computing Surveys (CSUR), 49(1), 12.

M. Anisetti, C.A. Ardagna, E. Damiani, F. Gaudenzi, N. El Ioini, ``Modeling Time, Probability, and Configuration Constraints for Continuous Cloud Service Certification'', COSE 2017

Bisgaard, Nikolai, Marcel SMG Vlaming, and Martin Dahlquist. "Standard audiograms for the IEC 60118-15 measurement procedure." Trends in amplification 14.2 (2010): 113-120.

Bhatt, Ishan S., and O'neil Guthrie. "Analysis of audiometric notch as a noise-induced hearing loss phenotype in US youth: data from the National Health And Nutrition Examination Survey, 2005–2010." International journal of audiology 56.6 (2017): 392-399.

Lee, Cheng-Yung, et al. "Using cluster analysis to classify audiogram shapes." International journal of audiology 49.9 (2010): 628-633.

Majumder, J., and L. K. Sharma. "Application of Data Mining Techniques to Audiometric Data among Professionals in India'." Journal of Scientific Research & Reports 3.23 (2014): 2860-2971.

Yoder, Jordan, and Carey E. Priebe. "Semi-supervised k-means++." Journal of Statistical Computation and Simulation 87.13 (2017): 2597-2608.

Grover, M. Malaska, T., Seidman, J.: "Hadoop Application Architectures: Designing Real-World Big Data Applications", "O'Reilly Media, Inc.", 30 giu 2015

Grover, M. Malaska, T., Seidman, J.: "Pattern: Hadoop Application Architectures: Designing Real-World Big Data Applications" (chapter Time Series Modification, Use HBase with a row key of Record Key and StartTime)."O'Reilly Media, Inc.", 30 giu 2015

Chuan-kai, L., Black, A. P. : "DirectFlow: a Domain-Specific Language for Information-Flow Systems", Department of Computer Science Portland State University 2014

Koren, Y., Bell, R., Volinsky, C. "Matrix Factorization Techniques for Recommender Systems," in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009. doi: 10.1109/MC.2009.263

Zhang,C., Chen, X., Feng, X., Ge, B.:"Storing and Querying Semi-structured Spatio-Temporal Data in HBase" WAIM Workshops 2016: 303-314

Kalakanti, A. K. , Sudhakaran, V., Raveendran, V. Menon, N.: "A comprehensive evaluation of NoSQL datastores in the context of historians and sensor data analysis," 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, 2015, pp. 1797-1806.

Appendix A: Preliminary Workflows Definition

In the following, we provide some of the initial modelling of EVOTION analytic workflows. We divide the workflows in Policy making workflows and Clinical and Patients-related workflows.

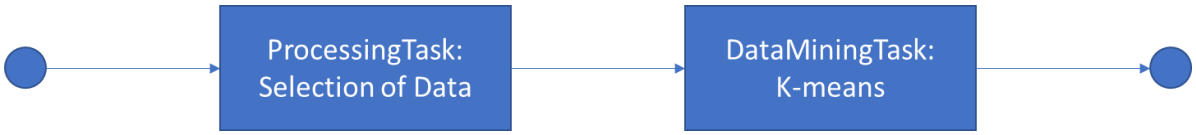
When needed for the following scenarios, we used synthetic datasets for testing purposes in accordance to the recommendation of D9.3 related to ethics and privacy. In some other cases, we use real datasets anonymized related to HA data only to have more realistic scenarios.

We note that during the evolution of the project we will tune them and restructured them for finding the most suitable approach for each of specific EVOTION domains, therefore the clinical/statistical validity of the following solutions, even if extracted from literature in some cases, are not currently verified and out of the scope of this deliverable.

Workflows for Policy Making Scenarios

These types of workflows are modelled by a policy maker using the EVOTION PHPDM Specification tool and the EVOTION PHP language. The portion of the obtained PHPDM model instance related to BDA workflow, i.e. the DAW, is transformed by the PHPDM Transformation tool into EDAW an added to the Workflow Catalogue to be executed.

Let us consider for example the scenario related to "Addressing Barriers to HA Use" modelled in D4.1. The scope is to explore whether the (i) Occupation, (ii) Education level and (iii) Age of HA users affects their daily usage. Following the PHPDM model in D4.1 the EDAW generated after a conversion can be modelled as follows



It is a sequence of two Tasks, one is a processing Task selecting the source of data and the second one is a clustering task. There can be more than one Processing Task in order for instance to clean the data from outliers or to select a specific subset. Similarly, there can be more than one Analytics Tasks like statistical analysis to evaluate correlations of factors. In general, a policy making process is made by a number of different consecutive analyses to let policy makers infer something from the data to model the policy.

These analyses are transformed to workflows via PHPDM Transformation tool.

Workflows for Clinical and Patient-related Scenarios

Our BDA Engine will also support a number of different scenarios where the EDAWs are not dynamically generated by the Transformation tool, but instead pre-loaded and scheduled for supporting recurrent analytics like in case of Clinical and Patients scenarios, or for managing complementary tasks of the BDA.

Some of these pre-loaded EDAWs refer to aggregation-oriented workflows for visualization purposes

Requirement	Description
FR(CLIS)37	Manage HA usage with problems occurred, ratings provided and corresponding noise recorded
FR(CLIS)48	Manage and visualize a detected event record

FR(CLIS)77	Visualize aggregated data sets
FR(CLIS)78	Visualize HA usage data with respect to various noise parameters

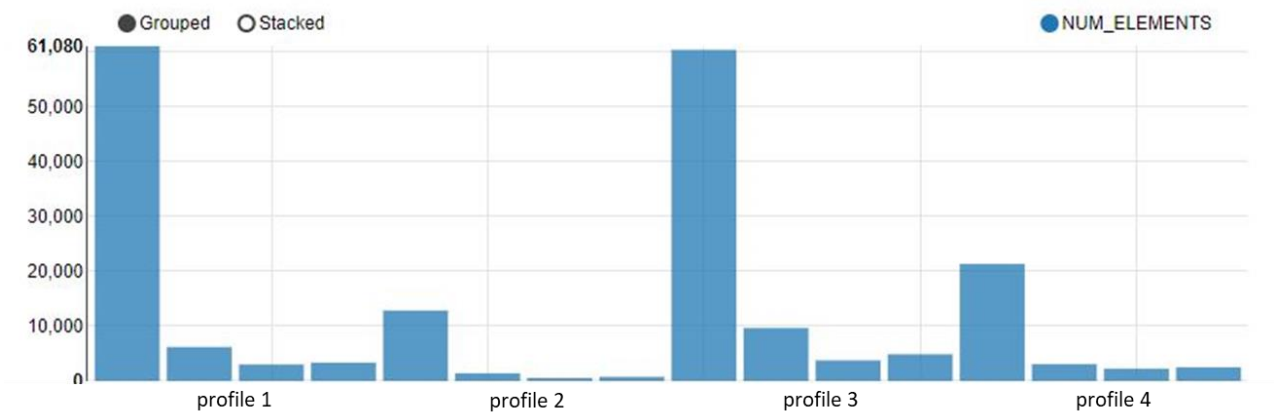
These aggregation/visualization workflows can be provided with processing Task and exploiting Zeppelin-based EDAW at EVOTION Dashboard level. Normally, the principal scope of these simple visualization oriented EDAWs is the prototyping of analytics.

FR(CLIS)77: Visualize aggregated data sets [aggregation and simple pre-processing]

For the shake of simplicity let us consider as dataset, the real but pseudo-anonymized data on HA_ENVIRONMENT_DATA and TIME_PERIOD tables in the EVOTION Repository. Aggregation and visualization EDAWs have almost a similar structure made with one or mode Processing Tasks for aggregation and then a visualization of the final result.

Let us consider the relation between the HA profile and the Environment (i.e., quiet, noise, speech, speech in noise). An EDAW can be structured as follows:

Given this EDAW different aggregations and types of visualization can be provided. In the following an histogram showing the aggregation per Environment and per Profile.

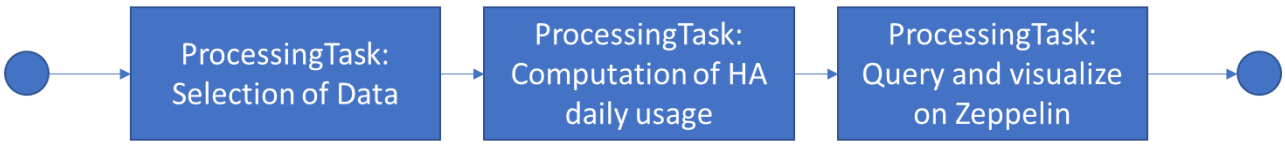


Form this plot is quite clear that the quite environment is the mostly frequent for all the profile

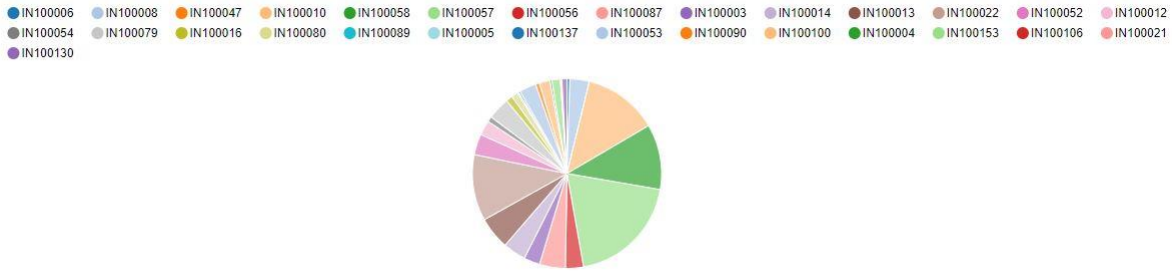
FR(CLIS)78: Visualize HA usage data with respect to various noise parameters [aggregation and simple pre-processing]

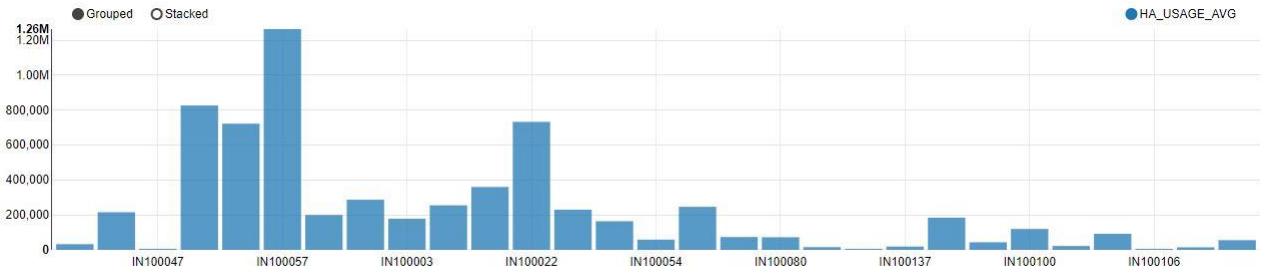
All the HA daily usage related EDAWs share the HA usage computation processing Task, that compute the daily usage given the data in the HA TIME PERIOD table.

More specifically let us consider a workflow for HA usage per patient. To make it effective in terms of visualization, a subset of patients must be selected. Let us consider a simple random selection of a subset of patients. The workflows can be structured as follows.

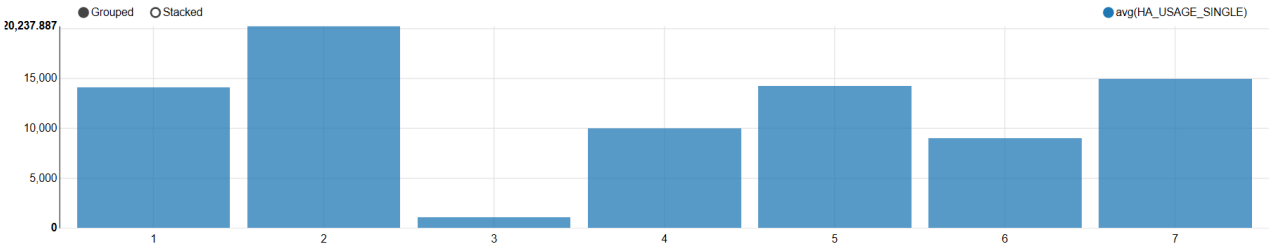


Given this EDAW the distribution of HA daily usage on 27 randomly selected patients can be visualized as follows using both the two different view





Let's now consider to include audiograms as a way to cluster patients. We aggregate patients with similar audiograms and provide the same aforementioned analysis but at audiogram clustering level. Let us assume that audiogram classification has been executed a priori (see below for details). In the following we aggregate audiograms in 7 similarity clustering and then compute the average HA usage for all the patients belonging to each cluster.



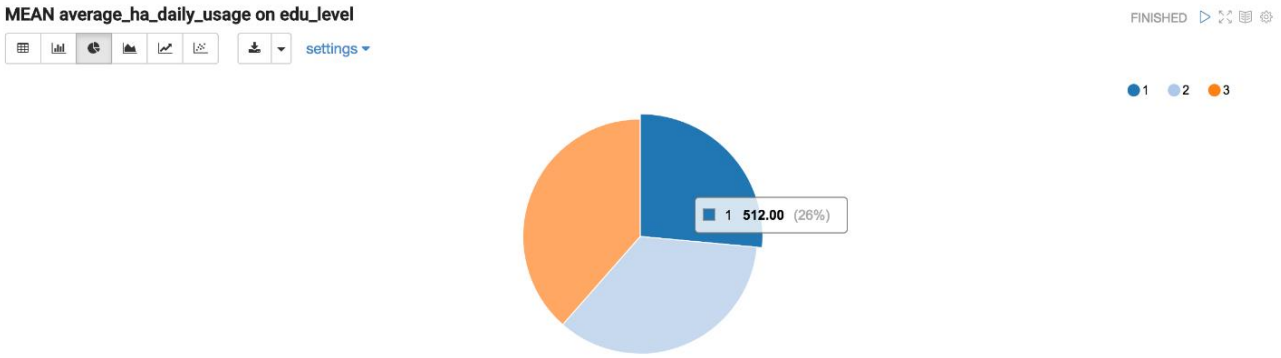
HA usage can be also used in relation with other more personal characteristics like educational level, age etc.

In the following an example of average plot of HA Delay usage, grouped by the three educational levels labelled as 1, 2 and 3 directly written using Zeppelin.

MEAN average_ha_daily_usage on edu_level

```
val dataText = sc.textFile(z.input("file"))

case class Data(patient_id: Integer, ghabp_q1: Integer,  ghabp_q2: Integer, ghabp_q3:
Integer, ghabp_q4: Integer, ghabp_q5: Integer,
 ghabp_q6: Integer, ghabp_q7: Integer,moca_total: Integer, speech_in_noise: Integer,
average_ha_daily_usage: Integer, edu_level: Integer,
 engagement_in_at: Double, periodic_follow_up: Integer)

val data = dataText.map(s => s.split(",")).filter(s => s(0) != "PATIENT_ID").map(
    s => Data(s(0).toInt, s(1).toInt, s(2).toInt, s(3).toInt, s(4).toInt, s(5).toInt,
s(6).toInt, s(7).toInt,
        s(8).toInt, s(9).toInt, s(10).toInt, s(11).toInt, s(12).toDouble, s(13).toInt
        )
).toDF()
data.registerTempTable("data")

var groupby="";
var selgroupby="";
if(!z.input("groupby").equals("")){
    groupby=" GROUP BY "+z.input("groupby")+" ORDER BY "+z.input("groupby");
    selgroupby=z.input("groupby")+",";
}

val df = sqlContext.sql("SELECT "+selgroupby+z.input("stat")+"("+z.input("field")+") FROM
data"+groupby)
df.coalesce(1).write.option("header", "true")
  .csv(""+z.input("location"))
```

The following figure shows the results of the execution, where the dependencies between educational level and the average HA usage is not evident.



Other pre-loaded EDAWs refer to specific analysis on patient's data:

Requirement	Description
FR(CLIS)36	Correlation of HA usage with collected data
FR(CLIS)42	Analyse END or BHD parameters and automatically respond to them by changing the fitting profile

FR(CLIS)61	Analyse data and suggest combination of factors affecting TTS and NIHL episodes
FR(CLIS)82	Analyze the responses to the auditory training tests
FR(CLIS)95	Analyse captured events from the patient's devices in relation to the patient responses in the questionnaires
FR(CLIS)103	Analyze sensors' and HAs' data
FR(PSOS)139	Determine combination of factors (noise levels, duration of exposure, other physiological data) associated with TTS/NIHL episodes
FR(PSOS)144	Analyse issues, concerns and problems with hearing aid reported by HA users

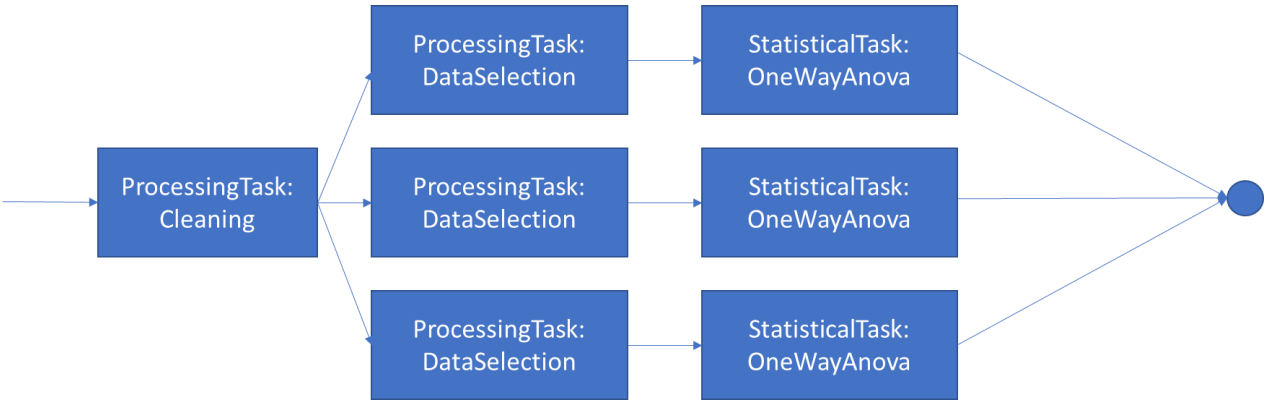
This workflows for patients/clinicians analysis require specific analytics to be triggered periodically. These permanent EDAWs can be tuned by changing operational and scheduling parameters to fit the specific needs.

FR(CLIS)36: Correlation of HA usage with collected data - [Statistical Analysis]

The scope is to analyse the correlations of HA usage with noise exposure, environment (END 1,2,3) and activities as recorded by the sensors (working, driving, watching TV-BHD 1,2,4,5) of a group or a given patient.

This is the typical analysis requiring Statistical Analytics. More specifically we consider One-Way Analysis of Variance (ANOVA), that is a technique for studying the relationship between a quantitative dependent variable and a single qualitative independent variable. In this case, the quantitative variable is the HA usage, while the qualitative variables are respectively noise exposure, environment and activities.

The workflow is composed of 3 parallel execution of One-Way ANOVA for which all the detailed statistics are provided. The correlation is than evidenced by the P-values computed.



In Oozie parallel jobs can be obtained using fork join construct. The relative Oozie workflow extract for the above EDAW is the following.

```
<workflow-app name="HA_Usage_ANOVA"
  xmlns="uri:oozie:workflow:0.5">
  <start to="Cleaning"/>
  <fork name="fork0_1">
    <path start="SelectBHD"></path>
    <path start="SelectNoise"></path>
    <path start="SelectEnv"></path>
  </fork>
  <join name="join1" to="end"></join>
  <action name="Cleaning">
```

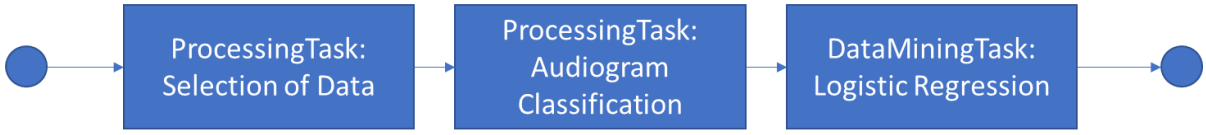
```
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>ProcessingTask_Cleaning</name>
      <jar>/user/bda/evotion-task/CleaningTask.jar</jar>
      <arg>...</arg>
    </spark>
    <ok to="fork0_1"/>
    <error to="kill"/>
  </action>
  <action name="SelectBHD">
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>SelectBHD</name>
      <jar>/user/bda/evotion-task/SelectionTask.jar</jar>
      <arg>...</arg>
    </spark>
    <ok to="ANOVABHD"/>
    <error to="kill"/>
  </action>
  <action name="ANOVABHD">
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>OneWayANOVA3</name>
      <jar>/user/bda/evotion-task/OneWayANOVA.jar</jar>
      <arg>...</arg>
    </spark>
    <ok to="join1"/>
    <error to="kill"/>
  </action>
  <action name="SelectNoise">
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>ProcessingTask_Dataselection</name>
      <jar>/user/bda/evotion-task/SelectionTask.jar</jar>
    </spark>
    <ok to="ANOVANoise"/>
    <error to="kill"/>
  </action>
  <action name="ANOVANoise">
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>OneWayANOVA2</name>
```

```
      <jar>/user/bda/evotion-task/OneWayANOVA.jar</jar>
      <arg>...</arg>
    </spark>
    <ok to="join1"/>
    <error to="kill"/>
  </action>
  <action name="SelectEnv">
    <spark
      xmlns="uri:oozie:spark-action:0.2">
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <master>yarn-cluster</master>
      <name>SelectEnv</name>
      <jar>/user/bda/evotion-task/SelectionTask.jar</jar>
      <arg>...</arg>
    </spark>
    <ok to="ANOVAENV"/>
    <error to="kill"/>
  </action>
...
  <kill name="kill">
    <message>${wf:errorMessage(wf:lastErrorNode())}</message>
  </kill>
  <end name="end"/>
</workflow-app>
```

This is clearly a simple way of statistically evaluate correlation that can be enhanced with other investigations, for instance using a single workflow implementing a Three-Way ANOVA enabling also the two-way interaction evaluations to have a complete view on the phenomenon.

FR(CLIS)61 and FR(PSOS)139: suggest combination of factors affecting TTS and NIHL - [Data Mining Analysis]

The scope of FR(CLIS) 61 is to analyse data and suggest combination of factors affecting TTS and NIHL episodes (noise levels, duration of exposure, other physiological data).



In this scenario, audiograms need to be classified as pre-processing before starting the Analytic task. Bisgaard et al. (Bisgaard et al., 2010) defined a number of standard audiograms, which cover the entire range of audiograms met in clinical practice. Being able to map a given audiogram to one of the list of standard ones is of a paramount importance for many scenarios including this one. To implement this task a number of solution is available in literature like semi-supervised clustering approaches (Yoder et al., 2017). The scenario is much more simple due to the fact that we just have one representative per audiogram, therefore distance based approach is suitable. Therefore, the audiogram classification is based on the minimal distance computations between audiogram vectors.

We then use a Logistic regression to inspect the combination of factors using an approach similar to the one in Bhatt et al. (Bhatt et al., 2017)

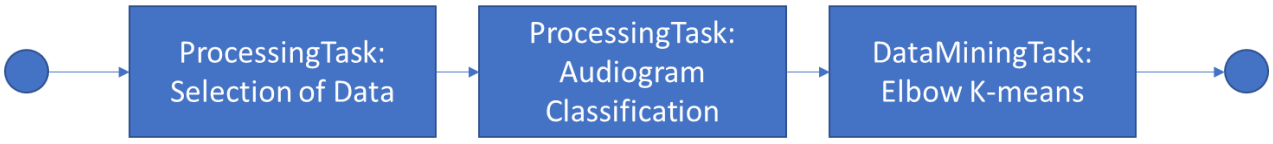
The aforementioned approach can be used also for FR(PSOS)139, but as a patient scenario it can be modified in a more personalized way, for instance by adding prediction capabilities to notify dangerous trend increasing the hearing loss. This model for predicting hearing loss situations can be constructed also with the future goal of helping clinicians in the diagnosis of hearing loss (Lee et al., 2010). First, the audiogram shape as well as other data can be clustered into groups. Then using Silhouette, the clusters can be labeled and used as training set for a classification purposes based on Naïve Bayes (NB) classification similarly to the approach in Majumder et al. (Majumder el al., 2014) .

FR(CLIS)42: changing the fitting profile – [Streaming/Batch Data Mining]

It is focused on Analyse END or BHD parameters and automatically respond to them by changing the fitting profile. This case points to a streaming scenario. Let us consider a more complex scenario based on a set of workflows aimed at providing fitting profile changes suggestions.

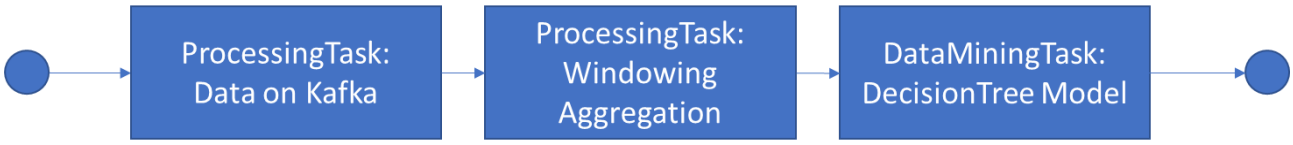
The idea is: i) clusterize the population of patients into classes of risk using personal behavioural data and audiograms, ii) for each class, develop a specific classification algorithm taking as input the stream of sensors data and produce as output suggestions to change the fitting profile. This is based on the idea that the changing of fitting profiles should depend not only on the sensors data but also on the risk class of a patient. This approach requires batch learning for class of risk clustering (low frequency) and micro-batch learning for fitting profile (high frequency), one per class of risk. It also requires a stream classification workflow that, given the data from an input Kafka queue, is able to identify the risk cluster (prediction) and redirect the data stream and the classification feedback to the correct learning classifier (i.e., the one for that risk class) for update.

The risk clustering workflow to build the model for clustering patients based on personal data like age, education level, occupation, but also last available audiometry (or a set of audiometry), can be structured as follows: i) preliminary pre-processing for selection of data, ii) classification of the audiometry iii) k-means enhanced with elbow method to find the best number k of clusters.



This workflow is scheduled with low frequency since it works on high latency data like personal data. The output of this workflow is a clustering model that will be then used to classify each patient with the relative cluster.

The fitting profile learning workflow is aimed at learning a model to suggest fitting profile for a given class of patients. It is trained mainly with the sensor streams and patient (using mobile device) or clinician (during the follow-up visit) feedbacks. The learning model evolves while additional sensors data will be acquired as well as feedbacks collected. This workflow can be structured as follows: i) pre-processing on the queue that already contains data for a specific cluster of patient only, ii) windowing or aggregation of data to make them coherent in the given time frame, iii) a decision tree that will be trained with the scope of deriving the most suitable profile given the data received.



This workflow is scheduled with a higher frequency compared with the previous clustering one. Each of these workflows read data on a specific queue and accumulate them to be used to update the training model.

We remark that just the data belonging to cluster k is used to train the fitting *profile model of class k*.

The complete workflow for profile suggestion is made of predictors and receives as input the last available clustering model, and the data from the HA sensors and mobile device. It first classifies patients to the correct cluster k then it passes all the data to the fitting prediction that uses the relative decision tree for that specific k. The decision tree generates a decision which could be a change of the patient profile to another one.

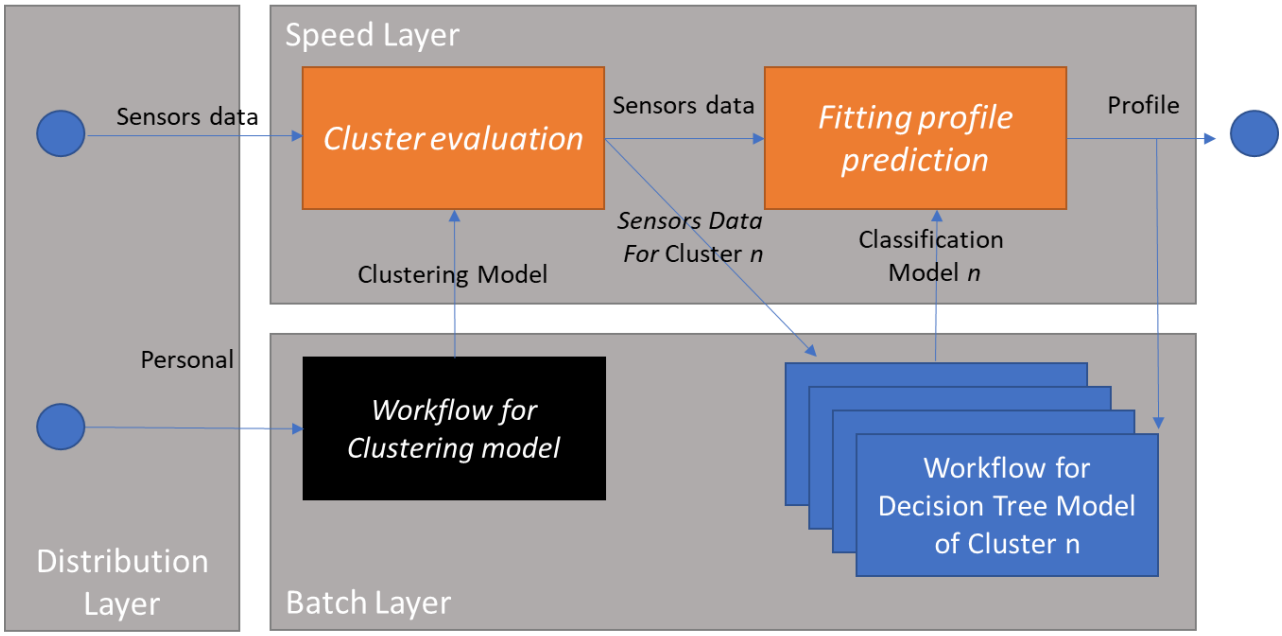


Figure 12: Schema for the workflow for profile suggestion and a mapping to the Lambda Architecture. In orange the streaming workflows, in black the classification using high latency Clustering model and in blue the high frequency decision tree classification for the given cluster n.

These prediction workflows are normally triggered when new amount of data is available in a queue. They are structured as consumer of the queue. In EVOTION we use Kafka queue. Each pre-defined EDAW doing stream processing on the Kafka queue are similar to the normal EDAW in terms of structure with the difference that they are always running processes fetching data from the queue. All learning models produced can be saved in the repository or in the HDFS any time an update is required.

The architecture just described is a typical approach following the Lambda architecture way of producing analytics where the model is batch and the evaluation/prediction is a stream.

This approach can be useful also for other scenarios like CLIS.3 “Ask the expert” hearing aid fitting or PSOS.3 “Self-testing of hearing and self-adjustment of hearing aids”.

We note that the architecture is ready to provide additional streaming features. We also note that the streaming capabilities provided can be emulated also using micro-batch and frequent scheduling to cope with the actual scenarios.