Project Number 723634

D2.5 Final Specification of Predictive Analytics Platform

Version 1.0
12 November 2018
Final

EC Distribution

IKERLAN

Project Partners: **ATB, Electrolux, IKERLAN, OAS, ONA, The Open Group, University of York**

PROJECT PARTNER CONTACT INFORMATION

ATB Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany Tel: +49 421 22092 0 E-mail: scholze@atb-bremen.de	**Electrolux Italia** Claudio Cenedese Corso Lino Zanussi 30 33080 Porcia Italy Tel: +39 0434 394907 E-mail: claudio.cenedese@electrolux.it
IKERLAN Trujillo Salvador P Jose Maria Arizmendiarrieta 20500 Mondragon Spain Tel: +34 943 712 400 E-mail: strujillo@ikerlan.es	**OAS** Karl Krone Caroline Herschel Strasse 1 28359 Bremen Germany Tel: +49 421 2206 0 E-mail: kkrone@oas.de
ONA Electroerosión Jose M. Ramos Eguzkitza, 1. Apdo 64 48200 Durango Spain Tel: +34 94 620 08 00 jramos@onaedm.com	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of York Leandro Soares Indrusiak Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325 570 E-mail: leandro.indrusiak@york.ac.uk	

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial Contents from D2.2 Incremental Update	11 September 2018
0.2	Additional Contents for final specs added	2 October 2018
0.9	Document Final Draft	24 October 2018
1.0	Proofreading and typos	12 November 2018

TABLE OF CONTENTS

1. Introduction **7**
1.1 Document purpose *7*
1.2 Progress beyond D2.2 Early Specification of Predictive Analytics Platform *7*
1.3 Approach Applied *8*
2. Background of Existing Technologies **9**
2.1 Big Data Frameworks *9*
2.2 Analytical Techniques Background *10*
2.2.1 Regression Techniques 11
2.2.2 Machine Learning Techniques 13
2.3 Machine Learning Algorithms in Apache Spark *14*
2.3.1 Core Concepts 14
2.3.2 MLib - Conceptual main classes 17
2.3.3 Algorithms in Spark's Machine Learning Library (MLlib) 19
2.3.4 Hyperparameter tuning 23
2.3.5 Spark, TensorFlow, Keras and Deeplearning4j 25
2.4 Other Predictive Analytics Software *26*
2.4.1 Open-Source Software 26
2.4.2 Commercial Software 27
2.4.3 Interoperability of Predictive Analytics Software 28
2.5 Related EU Projects *30*
3. Innovation **31**
3.1 Dataflow architecture *32*
3.2 Predictive analytics elasticity *32*
3.3 Predictive analytics service *33*
4. Predictive Data Analytics Platform Module Specifications **34**
4.1 Specifications to handle General Requirements *34*
4.1.1 Requirements 34
4.1.2 Specifications 35
4.2 Specifications to handle Cross-Components Requirements *36*
4.2.1 Requirements 36
4.2.2 Specifications 36
4.3 Specifications to handle Industrial Business Case Requirements *36*
4.3.1 Requirements 37
4.3.2 Specifications 39
4.4 Specification of Generic Functionalities *39*
4.4.1 Data Collection/Storage Functionality Specification 40
4.4.2 Data Query Functionality Specification 44
4.4.3 Predictive Modelling Functionality Specification 44
4.4.4 Data Quality Assurance Specification 57
4.4.5 GDPR Compliance Specification 61
4.5 Specification of High-Level Architectural Design *62*
5. Technology Specification of Software Tools **64**
6. Full Prototypes feature set **65**
7. Requirements coverage **66**

7.1 Data Mining and Analytics Requirements from Industrial Business Cases *66*
7.2 Performance Requirements from Industrial Business Cases *67*
7.3 Interface Requirements from Industrial Business Cases *67*
8. Conclusions **68**
9. References **69**

EXECUTIVE SUMMARY

The document presents the final specifications of the Predictive Analytics Platform and, as stated in the technical annex of the project, is an incremental update of the deliverable *D2.2 Early Specification of Predictive Analytics Platform*.

The deliverable contains (a) a review of current big data and predictive analytics techniques, (b) a summary of innovations to be developed in the project, (c) specifications for the component are defined, tracing them back to the requirements stated in previous deliverables, (d) a classification of the specifications according to the requirements they handle (general requirements, cross-component requirements and industrial business case requirements) and finally (e) a list of software tools to be used in the development of the software components.

1. INTRODUCTION

1.1 DOCUMENT PURPOSE

The current document presents the deliverable *D2.5 – Final Specification of Predictive Analytics Platform*, of the project *SAFIRE - Cloud-based Situational Analysis for Factories providing Real-time Reconfiguration Services*. This document was compiled by enhancing *D22 Early Specifications of Predictive Analytics Platform* with additional contents to produce a *self-contained* final deliverable *D2.5 - Specifications of Predictive Analytics Platform*

The work described here is part of the *T2.2 Specification of Predictive Analytics Platform* for the *WP2 – Predictive Analytics Platform*. The objective of task T2.2 is to address specific problems and requirements collected within WP1, through the specification of the architectural design decisions for the SAFIRE platform for real-time big data analytics.

1.2 PROGRESS BEYOND D2.2 EARLY SPECIFICATION OF PREDICTIVE ANALYTICS PLATFORM

This final specification document is a follow-up to "D2.2 Early Specification of Predictive Analytics Platform". This section outlines the main additions to that specification:

- Specification of the Predictive Analytics Platform Module regarding the dimensions of data quality.
- Specification of the Predictive Analytics Platform Module regarding the compliance of big data technologies to the European General Data Protection Regulation (GDPR).
- Enhanced specification of available machine learning algorithms and core concepts of Spark available for Predictive Analytics Platform.
- Update the requirements coverage table in Section 7.
- Specification of the Predictive Analytics Prediction REST Web Service.
- Description of Source Code templates for Predictive Analytics Prediction REST Web Service Clients development to be developed in the full prototype.
- Specification of testing to be done in business cases with the full prototype.

1.3 APPROACH APPLIED

The Final Specification of Predictive analytics presented here is the result of a process already started in SAFIRE Concept and the following previous deliverables:

- *D1.1 Application Scenarios Requirements Analysis.*
- *D1.2 Optimisation Metrics and Benchmarking.*
- *D1.3 Business Cases Infrastructure Specification.*
- *D1.4 SAFIRE Concept.*
- *D2.2 Early Specifications of Predictive Analytics Platform.*

In short, the steps followed to compile this deliverable were as follows:

- First, predictive analytics requirements specified by industrial business cases, cross components requirements and general requirements for a predictive analytics platform have been analysed.
- Next, a review of existing technologies in big data and predictive analytics has been conducted, trying to focus the research in the requirements described above.
- Finally, the early specifications deliverable has been enhanced with additional specifications to finally produce this deliverable.

The structure of the document is organized as detailed below:

- **Section 1, Introduction** - Includes a concise overview of the overall content of the document, mentioning; document purpose, progress beyond D2.2, approach to produce this document and structure of the document.
- **Section 2, Background of Existing Technologies -** Provides an introduction to the state-of-the-art in big data, predictive analytics techniques and some related EU projects.
- **Section 3, Innovations -** Provides a summary of innovations developed in Predictive Analytics Platform module.
- **Section 4, Predictive Analytics Platform Module Specifications -** Provides a description of the specifications, classified according to the requirements they are handling: general requirements, cross-components requirements and industrial business cases requirements. And presents a high-level architectural design of the full prototype.
- **Section 5, Technology Specification of Software Tools -** Provides a list of software tools that can be used in the component.
- **Section 6, Full Prototypes feature set -** Provides a list of features available on the Full Prototype.
- **Section 7, Requirements coverage -** Provides a detailed coverage of each BC requirement.

- **Section 8, Conclusions -** Finally, this section provides a brief final summary of the document.
- **Section 9, References -** References are depicted in this section.

2. BACKGROUND OF EXISTING TECHNOLOGIES

Predictive Analytics is a broad set of techniques ranging from statistics to machine learning with the aim of analyzing data (historical data, real-time data, data traces, etc...) in order to, for example, predict future values of the data, find relationships or discover behavioural patterns.

In this context, big data analytics is the process of applying predictive analytics to very large datasets to uncover hidden patterns, unknown correlations, customer preferences and other useful business information. Big data analytics has been successfully applied to real world industrial use cases, bringing in a number of benefits related to advanced manufacturing. Nowadays, the main focus is to obtain predictions using real-time techniques. As a case in point, maintenance predictions derived from real-time data analytics of worldwide operating products and operating systems is a reality on many industrial domains.

Sections below analyse background of existing technologies along four main axes:

- *Big Data Frameworks.*
- *Analytical Techniques background.*
- *Predictive Analytics Software.*
- *Related EU Projects.*

2.1 BIG DATA FRAMEWORKS

Big data frameworks enable organizations to store, manage and manipulate vast amounts of disparate data. The Hadoop File System [1] (HDFS) is the *de facto* standard framework that allows massive data storage in its native form to speed up analysis and insight. The Hadoop framework implements its own approach to programming distributed computing, called Map Reduce [1]. There are many pure Hadoop providers such as Cloudera, Hortonworks, MapR, Pivotal or TeraData that integrate Hadoop with other frameworks for a complete solution. On the other hand, the biggest cloud providers (Amazon AWS [2], Google Cloud [3], and Microsoft Azure [4]) offer complete elastic solutions with a pay-per-use business model. Moreover, companies such as IBM [5], SAP [6], and GE [7] also offer global solutions for industry on a single platform that combines these technologies with tools for predictive analytics.

Apart from HDFS, the organizations that require real-time access to their big data warehouse use NoSQL databases such as MongoDB [8] and Cassandra [9] which are compatible with the Hadoop ecosystem and being horizontally scalable too. NoSQL databases are present nowadays as an alternative to HDFS on big data systems where the data is frequently accessed in real-time [10][11]. Nevertheless, this kind of databases do not have the query capabilities of the SQL ones leading to the development of NewSQL [12] databases that have both the scalability of NoSQL and the query capabilities of SQL databases.

Moreover, organizations require Analytics to gain insights in their data. Predictive Analytics use data mining analytics, as well as predictive modelling to anticipate what will likely happen in the future based on insights gained through descriptive and diagnostic analytics. The ability to predict what is likely to happen next, is essential to improve the overall performance of manufacturing systems, especially product operations such as maintenance and utilisation. Using machine learning techniques, patterns can be found in historical operational data and real-time data to signal what is ahead. Lee et al. [13] describe recent advances and trends in predictive manufacturing systems in big data and cloud environment manufacturing.

Beyond the Hadoop approach for batch analytics, real-time capable processing frameworks such as Flink [14] or Spark [15] have been adopted quickly as an alternative to Map Reduce. These new approaches claim to be 100 times faster than Hadoop because of their in-memory processing capabilities. Spark and Flink provide a unique engine for batch and real-time big data analytics, simplifying the operation and maintenance of the system. Furthermore, they are interoperable with the wider Hadoop ecosystem providing specific libraries for machine learning.

These kinds of platforms are usually deployed following the so-called Lambda Architecture [16]. However, a Lambda Architecture is inherently complex as batch data and real-time data are processed on different paths. For this reason, the Kappa Architecture [17] was born as an effort to simplify the architectures of real-time big data platforms. More recently, the NoLambda [18] Architecture was designed combining streaming, machine learning and batch analytics in a simpler way.

Finally, operating a big data analytics platform usually involves dealing with a lot of computing resources; therefore, using some kind of resource manager greatly improves the performance. Historically, the Yarn [19] resource manager has been employed for Hadoop workloads. Nevertheless, new resource managers, such as Mesos [20], have been employed in production by many companies (e.g. Twitter, Apple, Netflix, Paypal), as they are capable of dealing with Hadoop and other kinds of workload. The maturity of Mesos has led to the concept of Data Centre Operating System [21].

2.2 ANALYTICAL TECHNIQUES BACKGROUND

Predictive Analytics [22] is based in a variety of techniques that basically, can be classified into two groups:

- ***Regression Techniques*** - These techniques try to find a mathematical relationship between the input variables and the output variables. Main techniques in this group include, among others:
    - Linear Regression.
    - Logistic Regression.
    - Multinomial Logistic Regression.
    - Time Series Models.
    - Decision Trees / Random Forest Trees.
    - Multivariate Adaptive Regression Splines.
- ***Machine Learning Techniques*** – These techniques, developed in the artificial intelligence community, include, among others:
    - Deep Learning algorithms / Neural Networks.
    - Support Vector Machines (SVM).
    - Naive Bayes (Bayesian Algorithms).
    - Clustering Algorithms.

In the following sections, each of these techniques will be described very briefly so that the reader can have a minimal background and a link to a resource with more information.

However, an in-depth description of each technique is out of the scope of this document, as it is not intended to be a state-of-the-art presentation of regression and machine learning techniques which, by themselves, are huge scientific research areas.

2.2.1 Regression Techniques

The wide variety of regression techniques available is the consequence of none of them offering optimal results for all problems. In fact, for a given type of problem, furthermore, a given instance of a problem, some techniques give better results than others. Often, it is necessary a data scientist to study the problem, select and apply the most appropriate technique. Main regression techniques include, among others, the following:

- *Linear Regression* – In this technique the relationship between input variables and output variables is expressed as a linear equation. The goal of the algorithm is to find the parameters of the equation that minimize a loss function that measures the difference between the values by the equation and the actual values. Loss function is typically the sum of squared residuals.

Though in some cases linear regression fits very well, relationship between variables is often non linear. And, what is more important, in vast amounts of input and output data it is difficult to judge "a priori" if a multiple dimensional linear regression fits the "shape" of the data. In addition to this, noise in the data can also damage linear regression.

- *Logistic Regression* [24] – Known also as Logit Regressions, or Logit Model, is a special case of regression in which the output variable is a "categorical variable" with a discrete set of values, being the most typical example a binary variable with only two values 0 or 1 (although it can be applied to multiple categories). In the case of binary categories, for each set of input variable values, the logistic regression tries to assign a probability to classify them as belonging to category 0 or category 1.

  The logistic regression uses the mechanism developed in linear regression by modelling the probability with a *logistic function* [24] applied to a linear equation of the input variables. The goal of the algorithms is to find the optimal parameters of the equation so that a given loss function is minimized. There are a variety of loss function types that will not be explained here, such as mean squared error, mean squared logarithmic error, cross-entropy error (frequently used for binary classification), etc...

  Logistic regression is very useful to model the influence of a set of independent variables in a categorical output variable (classification problem). However, success depends on selecting the right set of input variables (those who have influence) and ensuring the data collected from these variables are actually independent. In addition to this, unbalanced sets of input examples may lead to apparently good "predictive accuracy" even when the algorithm is not predicting anything. For example, if 95% of samples belong to category 1 and the algorithm classifies, blindly, all samples as 1, the accuracy would be 95%. Obviously, there are techniques to correct this problem but a skilled data scientist is needed.

- *Multinomial Logistic Regression* [25] – is basically a generalization of Logistic Regression to multiple categories.

- *Time Series models* – These models are used for predicting future values of variables in which a previous set of values and the temporal order in which they happened, are relevant. In this case, previously mentioned regression techniques cannot be applied. These techniques include mainly *autoregressive* models and *moving-average* models.

- *Decision Trees* [26] – are tree like DAG graphs (Directed Acyclic Graph) in which nodes represent decisions or chances. Traversing the graph helps taking a decision given an input set of input variables.

  This kind of tree is simple to understand, may be defined with little data at the beginning and extended with more data later, allowing even the representation

of new scenarios. However, defining a decision tree is complex and time consuming, often requires expert knowledge in the domain, and trees usually grow very large and therefore become difficult to understand/visualize, especially when working with big data.

- *Multivariate Adaptive Regression Splines (MARS)* [27] – It is a non-parametric simple and easy to understand regression technique with the ability to automatically model non-linearity. It is a kind of an extension of linear models but able to handle both categorical and continuous variables.

  MARS is a powerful technique that can be applied to large data sets and can compete and, in some cases, outperform Neural Networks [28]. This technique has the added benefit of not being a black box, and therefore, allowing engineers to understand and visualize discovered relationships better than NN.

2.2.2 Machine Learning Techniques

In many situations relationships between input variables can be very complex and no mathematical approaches such as regression are appropriate. In these cases, machine learning algorithms can emulate human reasoning and can learn, once they are given a set of training samples (input and corresponding outputs samples), how the variables are related.

Main machine learning techniques [23] include, among others, the following listed in the subsections below.

- *Deep Learning Algorithms / Neural Networks* [29] – Deep Learning algorithms, (basically deep neural networks) consist of a set of algorithms that are capable of training a structure of layers of nodes (neurons) with connections among neurons from previous and next layers. A single neuron consists of a processing element which has a number of inputs, each with an associated weight, a transfer function which determines the output given the weighted sum of the inputs, and the output itself. During the training process the network is fed with a set of samples (inputs and outputs) and the network learns to produce the right output for a given input by updating the weights. Nowadays, neural networks can be trained with huge amounts of data (hence they are very suitable for big data) and, once trained, can respond very quickly to new samples.

  In contrast to linear or polynomial regression no assumption is needed about the underlying relationship between input/output variables as the network can discover complex non-linear relationships by itself. It is also not necessary to identify with precision the relevant input variables as the network will learn to ignore non-relevant inputs. Neural networks are particularly robust to noise in the data.

  In addition to this, Recurrent Neural Networks as LSTM (Long-Short-Term-Memory) and Convolutional LSTM are special architectures to deal with Time Series data and are very appropriate for prediction and forecasting problems

where temporal order is important and where causal relation with events happened in the past is relevant to identify current events.

Neural Networks have also been applied to image recognition, speech recognition and text translation problems. In recent years, neural networks have witnessed a very notorious success in these fields.

- *Support Vector Machines (SVM)* – It is a supervised learning algorithm that constructs a set of multidimensional hyperplanes (there is a linear and non-linear version) typically used for classification and regression. They are used for text and hypertext classification, image classification, character recognition and also widely used in biological sciences for example to classify proteins.

- *Naive Bayes (Bayesian Algorithms)* – Naive Bayes classifiers, are a family of simple probabilistic classifiers based on the application of the Bayes Theorem with the assumption of independence between every pair of features. They have been shown to be very useful for text categorization, this is, categorizing the topic of a given document (for example, sports, politics or spam) using the frequencies of words, and even in medical diagnosis based on symptoms, physical examination and medical record of a patient. They may compete with SVM algorithms.

- *Clustering Algorithms* - These algorithms, such as k-nearest neighbours algorithm (k-NN), are used for classification and regression. Given a set of samples, the algorithm tries to classify the samples in groups by finding "centre points" in each group so that the distance of the members of a given group to its centre is minimized. Then, to classify a new sample, the distances of the new sample to the centres of the groups are computed, assigning the new sample to the closest group.

2.3 MACHINE LEARNING ALGORITHMS IN APACHE SPARK

As Apache Spark is the core of SAFIRE framework, this section specifies which are the core concepts used in SAFIRE and the main machine learning algorithms (called estimators in Spark).

2.3.1 Core Concepts

Spark is an open-source framework oriented to *real-time* big data ingestion and processing. The key concepts here are real-time and machine learning. The main features of Spark are:

- It is specially designed to tackle *real-time* data *ingestion* and *processing*.

- It is designed to take advantage of *parallelization* as data processing can be executed in multiple computing *clusters* in parallel.

- It uses an *in-memory* data processing architecture which results in much faster accessing and processing of the data (this is one of the key reasons for its adoption in SAFIRE, devoted to real-time).

- Is especially suited to support *machine learning algorithms* due to the fact that iterative or cyclic processes take advantage of the in-memory architecture benefitting of a much faster load and query data in the memory of a cluster. This is one of its main differences with Hadoop that is not memory-oriented, resulting in a much slower support, in fact inappropriate, for machine learning algorithms.

- There are three data abstraction types, *unstructured data RDD* (Resilient Distributed Datasets), more *structured DataFrame* and *DataSets*. These types represent and evolution from Spark 1.0 to Spark 2.0 and later: RDD is the oldest, Data Frames are more user-friendly and more efficient, and finally Datasets increase the ease of use and efficiency. (RDDs will be deprecated in the future for direct usage of developers).

- Data (Data Frames and Datasets included) is internally always organized as RDDs and can be distributed in different clusters. There are two types of operations on RDDS:

    - *Transformations* – Define how to create an RDD or how to transform one RDD out of other RDDs. Transformations, which can be chained in pipelines of subsequent operations, return new RDDs.

    - *Actions* – Ask for processing and returns a result (not an RDD) to the driver program (i.e collect, count, take, etc).

- Provides an API in *Python*, *Java*, *Scala* and *R*, and allows the definition of in-line functions with lambda architecture.

- A program in Spark is organised as a *Driver Program* that defines a graph of operations to be applied to RDDs, Data Frames and Datasets. These operations are executed by a set of *Worker Nodes* in clusters. Spark applies a *lazy evaluation* to the graph of operations so that execution is really performed when the whole graph is defined and the execution is finally asked. Before execution Spark may optimize the graph resulting into a simpler and faster graph. This process is depicted in figure below.

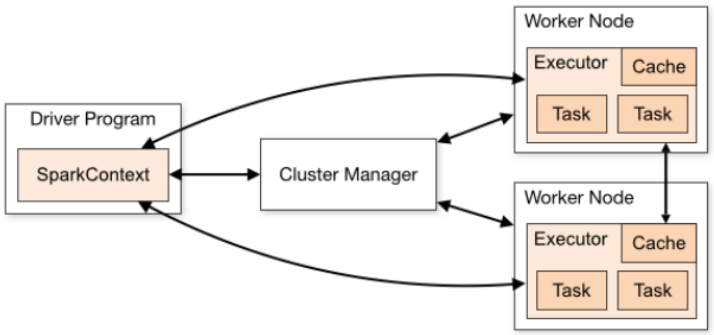


Figure 2-1: Sparks Driver Program-Worker Node architecture in cluster mode (figure taken from Apache Spark's web page)

Key components of Spark can be summarised as follows:

- ***Spark*** Core and ***Resilient Distributed Datasets (RDD), Data Frames and Datasets***. These components provide:
    - Input/output operations (i.e. import data from external files).
    - Distributed task dispatching and scheduling.
    - Data transformations (reduce, join, filter, normalize, tokenize, map, etc).
    - Etc.
- ***Spark SQL.*** Supports semi-structured and structured data and supports SQL with ODBC/JDBC servers.
- ***Spark Streaming***. Leverages the fast scheduling of Spark Core, allowing very quickly ingesting small data batches (needed for streaming), and applying transformations to these data batches in real-time.
- ***Machine Learning Library (MLib).***
    - It is a set of state-of-art machine learning algorithms.
    - Following Spark's philosophy, these algorithms can also be executed in a distributed environment (clusters) taking advantage of parallelization.
    - For a given problem, *it is very easy to switch out the learning algorithm*, so it is very easy to explore and play with different algorithms to discover which one works better for the problem to solve. This is a very important feature of Spark.

- ***GraphX***. It is a distributed graph processing framework. Provides an API for the computation of large-scale graphs of operations (transformators, estimators, etc) in an optimized way.

Sparks offers a library of Machine Learning algorithms with the goal of being real-time, scalable and easy to use (also key reasons for Spark adoption in SAFIRE). In the following subsections, main features and algorithms of Spark's library are explained.

2.3.2 MLib - Conceptual main classes

Conceptually there are four main abstract classes in the Spark MLib API (see https://www.youtube.com/watch?v=6tgvHDYT_AM for an excellent introduction to Spark's MLib) :

- ***Transformers*** *(data -> transformed data)*
    - Are used to pre-process data before applying a machine learning algorithm to the data.
    - Transform Data Frames into other Data Frames. A typical example of transformation is to *Normalize* the data (i.e. achieve 0.0 *mean*, and 1.0 *standard deviation*).
    - Example: *Transformer.transform(**data**: DataFrame)-> **data**: DataFrame.*
- ***Estimators*** *(data -> model)*
    - Train Data Frames and create a model. These are the machine learning algorithms and will be explained in some detail later in this document. There are estimators of the following main types.
        - *Classifiers* (Logistic Regressions, Decision Trees, Multi Layer Perceptrons, Random Forest, bayes, etc).
        - *Regressors* (Linear, Decision Trees, Random Forest, etc).
        - *Clusterers* (K-means, Latent Dirichlet Allocation, Gaussian Mixture).
    - Example: *Estimator.fit(**data**: DataFrame)-> trainedModel: **Model***
- ***Model*** *(data -> predicted data)*
    - Represents a trained engine that can be used to make predictions according to the training.

- It is, in fact, a kind of transformer, as it transform a Data Frame in another Data Frame with an additional column with the predictions.
- Example: *Model.transform(**testData**: DataFrame)-> **predictions**: DataFrame*

- ***Pipelines** (data -> data -> data -> .... -> data-> model)*
  - Allow chaining an arbitrary number of transformations followed by one estimator.
  - The pipeline itself is an Estimator, while the resulting model obtained by calling the fit() function is a transformed,
  - Example: *Pipeline.fit(**data**: DataFrame)-> **trainedModel**: PipelineModel*

Figure 2-2, borrowed from Apache Spark's MLlib main Guide in Spark's web page, explains these key concepts.

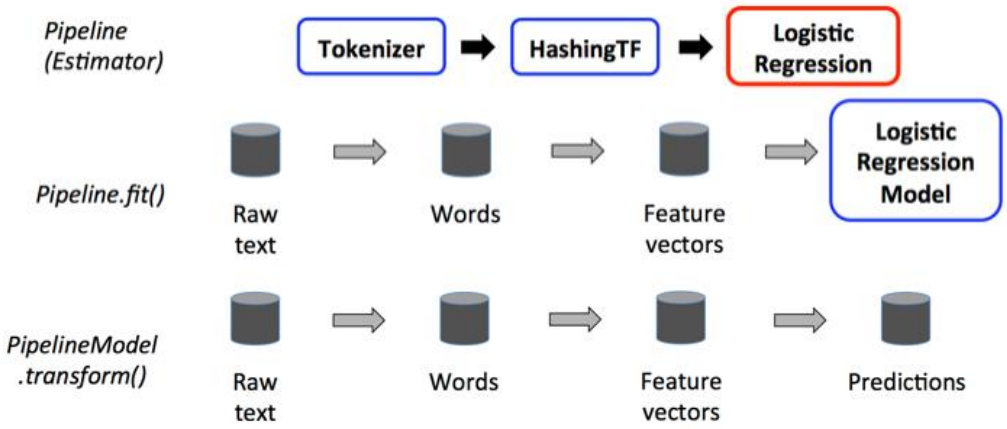


Figure 2-2: Pipeline is an Estimator that produces a Model (figure composed by images from Apache Spark's web page).

In Figure 2-2, the upper row shows the definition of an *estimator* by chaining two transformators (Tokenizer, HashingTF) and an estimator (LogisticRegression). By calling the *fit*() function, the pipeline (middle row) produces a model (Logistic Regression Model) that is a *transformator* that can produce predictions (lower row).

In summary, by using these elements (*Transformators, Estimators, Models, Pipelines*), the *Driver Program* defines a data processing that will be assigned to *Worker Nodes* in clusters and executed upon asking.

2.3.3 Algorithms in Spark's Machine Learning Library (MLlib)

This section gives an overview of machine learning capabilities of Spark (a short explanation will be given for some of the most interesting). Most of these algorithms have been introduced in Section 2.2.

Before getting into the description of the algorithms, it is worth introducing the list of functions to *extract*, *transform* and *select* the *features* that will be processed by the machine learning algorithms.

2.3.3.1 Extracting, Transforming and Selecting features

These functions help prepare the data for being processed with an estimator (a machine learning algorithm). The list of feature processing functions is huge, and it is out of the scope of this document to explain each one, as this document only wants to give an idea of the power of Spark's machine learning algorithms. See http://spark.apache.org/docs/latest/ml-features.html for a complete explanation of each function.

- Feature ***Extractors***
    - *TF-IDF* – feature vectorization method widely used in text mining to reflect the importance of a term in a document.
    - *WordToVec* – maps words of documents to fixed-size vectors.
    - *CountVectorizer* – convert a collection of text documents to vectors of token counts.
    - *FeatureHasher* – projects a set of categorical or numerical features into a feature vector of a specified dimension.
- Feature ***Transformation***
    - *Tokenizer* – converts text into individual words.
    - *StopWordsRemover* – excludes some words from a given text.
    - *n-gram* – converts a sequence of tokens (i.e. string) in sequences of n tokens (n-grams).
    - *Binarizer* – converts numerical values to 0 or 1 according to a threshold value.
    - *PCA* – performs an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables (principal components).
    - *PolynomialExpansion* – expands a set of features into a polynomial space (that is a n-degree combination of the input features).

- *Discrete Cosine Transform (DCT)* – applies a discrete fourier transform but only with real numbers.
- *StringIndexer* – encode list of string into numbers (assigned numbers are ordered by frequency).
- *IndexToString* – the opposite to StringIndexer.
- *OneHotEncoderEstimator* – encode a set of categorical features into a binary vector in which each 0/1 of the vector represents the presence of the corresponding feature.
- *Interaction* – Given two vector of double values, it computes the product of all combinations of values of both vectors.
- *Normalizer, StandardScaler, MinMaxScaler, MaxAbsScaler* – Given a vector of real values, these transformers help normalize and rescale their values. For example *StandardScaler* normalizes the value so that they have, for example, 0.0 mean and 1.0 standard deviation.
- *Bucketizer* – transforms a column of continuous values into buckets, where each buckets consists of a part of the range of the domain of the variables.
- *ElementwiseProduct* – computes element-wise product of two columns of real values.
- *SQLTransformer* – applies a SQL like transformation to a set of data frames.
- *VectorAssembler* – combines a list of columns to produce a single vector column that contains lists with the original values.
- *VectorSizeHint* – explicitly specifies the vector size of a column before it is completed (useful for example in streaming where the size is not known in advance)
- *QuantileDiscretizer* – bucketizes the input in *n* buckets, deciding the buckets on its own.
- *Imputer* – completes missing values in a dataset by computing the missing values as means or medians of the column or neighbours values.

- Feature **Selectors**
  - *VectorSlicer* – selects a sub-vector of the input sector.
  - *RFormula* – selects columns specified by a R model formula.

- *ChiSqSelector* – applies a *chi-squared test of independence* to decide the features to select.

- ***Locally Sensitive Hashing*** (LHS) – Hash data points into buckets (according to a family of LHS functions). Used in clustering.

2.3.3.2 Machine Learning algorithms (Estimators)

All of the algorithms described in this section try to predict the value(s) of a set of *dependant variables* (one or more), out from the values of a set of *independent variables* (usually more than the dependant variables). Two phases can be distinguished:

- ***Training***. In this phase, the algorithm learns (by an iterative process) out from a set of *labelled samples* (a *labelled sample* is a set of values of the *independent variables* for which the value of the *dependant variable* is known, it is, "labelled"). At the end of this phase the training process produces a *model*.

- ***Prediction***. In this phase the *model* is used to predict the dependant variable value for a given sample for which the label is unknown.

The algorithms described below use different techniques to produce the *model*.

Classification

- ***Logistic regression*** *(Binomial and Multinomial)*. The algorithm calculates (regresses) the coefficients involved in a linear combination of the values of the independent variables that are fed to a logistic function to predict the category of the *dependant variable* (two possible categories in a binary problem, and three or more in multi category problems). See https://en.wikipedia.org/wiki/Logistic_regression) for a comprehensive description of the algorithm. This is one of the most important methods in machine learning.

- ***Decision tree classifier***. These algorithms learn a decision tree in which each node represents a decision to be taken out from the value of a dependant variable. The leaves of the tree represent classifications categories of the sample. See https://en.wikipedia.org/wiki/Decision_tree_learning for a comprehensive description of the algorithm. Decision Trees tend to overfit to the training set.

  In this context, a set of algorithms know as *Tree Ensembles*, use multiple decision trees to improve the accuracy of just one single decision tree. *Tree Ensembles* implemented in Spark are *Random Forest* and *Gradient Boost* and are known to reduce overfitting of one single tree approach.

- ***Random Forest classifier.*** Are ensembles of multiple decision trees with the aim of reducing the overfitting. See https://en.wikipedia.org/wiki/Random_forest for a comprehensive description of the algorithm.

- ***Gradient-Boosted Tree classifier*** Are ensembles of multiple decision trees (usually of fixed size) that are iteratively improved by a gradient descendant algorithm optimizing a loss function. See https://en.wikipedia.org/wiki/Gradient_boosting for a comprehensive description of the algorithm.

- ***Multilayer Perceptron classifier***. It is a multilayer fully connected (all neurons in a layer are connected to all neurons in the next layer) neural network in which activation function of intermediate layers are sigmoid functions (logistic function) and activation of output layer is, either a sigmoid function (for two categories) or a softmax function (for multiple categories). See https://en.wikipedia.org/wiki/Feedforward_neural_network for a comprehensive description of the algorithm.

- ***Linear Support Vector Machine*** – constructs a set of hyperplanes in a n-dimensional space to classify a set of training samples. See https://en.wikipedia.org/wiki/Support_vector_machine for a comprehensive description of the algorithm.

- ***One-vs-Rest classifier*** – this technique (that is used to classify samples in multiple classes or categories) consists of training one single classifier for each category (labelling each sample as 1 if it belongs to the category and 0 otherwise). Once all classifiers have been trained (one per category) then, given a new sample, all classifiers are applied and the sample is assigned to the category of the classifier that returned a higher confidence. See https://en.wikipedia.org/wiki/Multiclass_classification#One-vs.-rest for a comprehensive description of the algorithm.

- ***Naive Bayes*** – it is a family of simple probabilistic classifiers that applies Bayes' theorems with strong (naive) independence assumptions between the features. See https://en.wikipedia.org/wiki/Naive_Bayes_classifier for a comprehensive description of the algorithm.

Regression

- ***Linear regression*** – applies the linear regression described in the previous algorithm of logistic regression.

- ***Generalized linear regression*** – these are linear models where the output may follow other probabilistic distribution different from Gaussian (the assumption under linear regression). These distribution functions may be one of the

following: Binomial, Poisson, Gamma, Tweedie and, of course, Gaussian. See https://en.wikipedia.org/wiki/Generalized_linear_model for a comprehensive description of the algorithm.

- ***Decision tree regression*** - applies a decision tree based regression described in the previous algorithm of decision trees classifiers.
- ***Random forest regression*** - applies a random forest regression technique described in the previous algorithm of random forest classifiers.
- ***Gradient-boosted tree regression*** - applies a gradient boosted tree regression technique described in the previous algorithm of gradient boosted tree classifiers.
- ***Survival regression*** – See http://spark.apache.org/docs/latest/ml-classification-regression.html#survival-regression for a description of the algorithm implemented by Spark.
- ***Isotonic regression*** – See http://spark.apache.org/docs/latest/ml-classification-regression.html#isotonic-regression for a description of the algorithm implemented by Spark.

2.3.4 Hyperparameter tuning

The training process of the algorithms described above need to be parameterized before starting the training. Although each algorithm defaults an initial setting of these parameters (with typical initial default values), results of trained model strongly depends of these parameters. Therefore a big data scientist will need to experiment with different parameter settings.

As an example of these parameters (known as *hyperparameters* of the algorithm), below it is shown the default hyperparameters' values of a *Random Forest Regression* (printed with *pyspark*).

```
# Define a random forest regressor
rf = RandomForestRegressor(....)

# Hyperparameters and their default values
{'cacheNodeIds': False,
 'checkpointInterval': 10,
 'featureSubsetStrategy': 'auto',
 'featuresCol': 'prediction',
 'impurity': 'variance',
 'labelCol': 'label',
 'maxBins': 32,
 'maxDepth': 5,
 'maxMemoryInMB': 256,
 'minInfoGain': 0.0,
```

```
'minInstancesPerNode': 1,
'numTrees': 20,
'predictionCol': 'prediction',
'seed': -814876731608538448,
'subsamplingRate': 1.0}
```

However, for example, maximum depth of the trees and the number of trees can have huge impact on the result. Obviously it is possible to run manually the experiments multiple times with different settings, selecting finally the best model found. Fortunately Spark's MLib offers an "automatic" research of the best settings. Let's see how.

With Spark MLib it is possible to define a *grid* of possible settings and let Spark run automatically repeatedly with the different settings. In the code below (python with pyspark), a grid of settings is defined. In this case *maxDepth* param will range in the values 5,6,7,8,9,10, while *numTrees* parameter will range in the values 10,11,12,13,14,15,16,17,18,19,20. Therefore a total number of 6x11 = 66 training experiments will be made, finally selecting the best model.

```
# Define a Param Grid
paramGrid = ParamGridBuilder() \
        .addGrid(rf.maxDepth, [5, 10]) \
        .addGrid(rf.numTrees, [10, 20]) \
        .build()
```

Once defined, the grid of settings a *TrainValidationSplit* estimator is built:

```
# Run automatically a validation over the training set and the grid
of params
trainValidationSplit = TrainValidationSplit(
        estimator=pipeline,
        estimatorParamMaps=paramGrid,
        evaluator=r2_evaluator)
```

And finally the *fit* function is called, with a final selection of the *best* model.

```
# Fit and select the best model
model = trainValidationSplit.fit(training).bestModel
```

Obviously, in this case, the fit process will take much more time than a single experiment, but this function allows exploring automatically with arbitrarily complex grids of parameters.

A very interesting possibility is to explore the possible combinations of hyperparameters setting in parallel, taking advantage of the parallel clusters of Spark. More on this is explained in next section.

2.3.5 Spark, TensorFlow, Keras and Deeplearning4j

In addition to Spark's MLlib, there well known open-source machine learning and deep learning frameworks, as for example:

- ***TensorFlow*** (https://www.tensorflow.org) machine learning framework originally originally by Google.
- ***Torch and Pytorch*** (https://pytorch.org/) open-sourced by Facebook in 2017.
- ***Caffe*** (http://caffe.berkeleyvision.org/) machine vision library for C/C++.
- ***Keras*** (https://keras.io/), high level API to use TensorFlow in Python.
- ***Eclipse Deeplearning4j***, deep learning platform for Java, well integrated with Kafka, Hadoop and Spark and capable of importing Keras trained models.
- *Etc.*

In general terms, Apache Spark (while providing the state-of-art machine learning library MLlib) is more oriented to real-time data processing and cluster computing, so allowing users to process big-data in multiple cluster, ensuring fault tolerance, etc. On the other hand, TensorFlow (and its API Keras), Deeplearning4J, etc, are very much oriented to define sophisticated and more complex machine learning algorithms.

In SAFIRE context, the emphasis is on Big Data, Real-Time ingestion and processing, so Spark (that in addition offers a quite good machine learning library MLib) is the natural answer. However for very complex learning tasks, a dedicated machine learning framework may be more appropriate.

Spark can be used in combination with TensorFlow, Keras, Deeplearning4j, resulting in a Big Data and Real-Time data processing with a variety of machine learning algorithms, MLib, TensorFlow, Keras, Deeplearning4j available depending the complexity of the learning task. In fact, machine learning frameworks needs big amounts of data for training, exactly what Sparks offers.

The following points show two cases using Spark and a cluster of machines to improve deep learning with tensor flow. These examples are taken from https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html in which a detailed explanation is given.

- ***Deploying large-scale models.***

  TensorFlow models can be embedded in pipelines to perform complex classification tasks. The model is distributed into multiple workers in multiple clusters by using the built-in broadcast mechanism of Spark. This architecture brings the possibility of applying deeplearning complex models to big data.

- ***Hyperparameters tuning in parallel.***

  In this case Spark's parallelization is used to find the best set of hyperparameters values for a neural network training.

  The TensorFlow library is installed in Spark's clusters as a regular python library and each cluster executes one training at a time for a given combination of hyperparameters. In this case, parallel searching, drastically reduces the time to discover the best model.

2.4 OTHER PREDICTIVE ANALYTICS SOFTWARE

There is a wide variety of Predictive Analytics Software packages available [30][31]. These packages can be classified into two groups: open-source software (basically free licence) and commercial software (pay licences).

Apart from licensing policy and fees, the main difference between them lies in the skills needed to use and customize them, and the amount of data they can handle efficiently. Generally speaking, open-source software requires more expertise to customize, train and handle the tools, while commercial software has more user friendly interfaces that make easier the task of modelling, visualizing, managing and training.

In addition to this, there is specifically a list of deep learning implementations that can be found in [32] reporting differences among most of the very well known packages such as Keras/Tensorflow, Caffe, Deeplearning4j, MatLab(c), Microsoft Cognitive Toolkit(c), Pytorch, Theano, Torch, etc. Some of these packages will be used in SAFIRE, especially Keras/Tensorflow. However, it is beyond the scope of this deliverable a detailed review of each software package.

In the following two sections we will give a brief list of the best known available open-source and commercial predictive analytics software packages.

2.4.1 Open-Source Software

- *Apache Mahout (https://mahout.apache.org)* – It is a project from the Apache Software Foundation that provides free implementations (most of them on top of Apache Hadoop) of machine learning algorithms for filtering, clustering and classification. Its algorithms are focused in Distributed Linear Algebra, Regression and Clustering. Mahout provides a Java API so that an end user can program its own application, therefore, it requires a deep knowledge of the API and it is better suited for an IT expert.

- *GNU Octave (https://www.gnu.org/software/octave)* – This tool and its Scientific Programming Language is a powerful mathematics oriented software with visualization tools, linear and non-linear solving packages, and many other

numerical packages, as for example statistics and machine learning toolbox. It is compatible (and one of the best free alternatives) with the very well-known MATLAB(c).

- *KNIME (https://www.knime.com/)* – It is specifically devoted to data analytics, reporting with a graphical interface that allows using the tool with minimal programming effort. KNIMES, written in Java and based on Eclipse, is used in a variety of areas like Customer Intelligence Analysis, Social Media Analysis, Finance Analysis, Manufacturing, Pharma and Health Care, Retails, etc. It allows processing large data sets, includes modules to connect with Big Data on Hadoop, supports major file formats (XML, JSON, images...), allows connecting to several databases, includes advanced predictive and machine learning algorithms, and integrates with machine learning libraries such as Keras, and Scikit-Learn. In addition to all this, it allows interactive design of dynamic workflows, interactive data-views and web-based reporting.

- *Orange (https://orange.biolab.si/)* – It is an open-source data visualization, machine learning and data mining toolkit. Similarly to KNIMES, it provides a visual interface to define data analysis tasks. It includes, among other features, data filtering, sampling, complex visualization, supervised learning algorithms for classification and regression, and special packages. For instance, these packages provide analysis tools for bioinformatics, networks, text mining, time series, etc.

- *R (https://www.r-project.org)* – It is a very well known GNU Package consisting of a programming language and a software environment for statistics supported by the "R Foundation for Statistical Computing". It is widely used by statisticians and data miners to develop statistics software and data analysis software.

- *Scikit-learn (http://scikit-learn.org/stable/#)* – It is a Python library for machine learning including Classification Algorithms (SVM, Nearest Neighbour, Random Forest, etc), Regression Algorithms, Clustering Algorithms, Dimensionality Reduction, etc.

- *Weka (https://www.cs.waikato.ac.nz/~ml/weka/)* – It is a collection of visualization tools, machine learning algorithms and predictive modelling written in Java and developed by the University of Waikato in New Zealand. Weka is integrated with Deeplearning4j for deep learning.

According to [33] and classified by categories, the open-source tools with higher score (Editor's rating) are Orange Data Mining, R Software, Weka, KNIME and HP Haven Predictive Analytics.

2.4.2 Commercial Software

The list of available commercial software is huge, as many companies are interested in the market of data analysis. In [30] a list of the most known open-source and

commercial software solutions can be found. In [33], an overview and a score of the best Predictive Analytics Tools, classified by categories, can be found. The list below represents some of the most relevant commercial tools.

- Alpine Data Labs
- Alteryx
- Angoss KnowledgeSTUDIO
- Actuate Corporation BIRT Analytics
- Dataiku DSS
- Google Cloud Prediction API
- IBM Analytics
- IBM SPSS Statistics and IBM SPSS Modeler
- KXEN Inc. Modeler
- Mathematica
- MATLAB
- Minitab
- LabVIEW
- Microsoft Azure Machine Learning
- Neural Designer
- Oracle Advanced Analytics
- Pervasive
- Predix
- Predixion Software
- RapidMiner
- RCASE
- Revolution Analytics
- SAP HANA and SAP Predictive Analytics
- SAS and its Enterprise Miner
- Sidetrade
- Stata
- Statgraphics
- Statistica
- Tibco Software

According to [33] and classified by categories, the commercial tools with higher score (Editor's rating) are Microsoft Azure Machine Learning, Dataiku DSS and Google Cloud Prediction API.

2.4.3 Interoperability of Predictive Analytics Software

Interoperability of Predictive Analytics Software requires a way to interchange models and data analysis between software packages. The answer to this need is PMML [34], a standard for statistical and data mining models (supported by over 20 vendors) that allows (a) developing a model with one software package, (b) exporting the model into a PMML file, and (c) import the model back into another software package.

PMML is a XML file (with a defined schema) that allows defining, and therefore interchanging, among others, the following concepts:

- *Field Scopes.*
- *Data Dictionaries.*

- *Mining Schemas.*
- *Transformations.*
- *Statistics.*
- *Functions.*
- *Bayesian Networks.*
- *Naive Bayes Models.*
- *Cluster Models.*
- *Regression Models.*
- *Neural Networks.*
- *Time Series.*
- *Vector Machines.*
- *Etc.*

In Spark, and therefore in SAFIRE, some of the machine learning algorithm models can be exported into PMML. The table below (borrowed from Spark's MLlib web page):

Spark's MLIB Model	PMML Model
KMeansModel	ClusteringModel
LinearRegressionModel	RegressionModel (functionName="regression")
RidgeRegressionModel	RegressionModel (functionName="regression")
LassoModel	RegressionModel (functionName="regression")
SVMModel	RegressionModel (functionName="classification" normalizationMethod="none")
Binary LogisticRegressionModel	RegressionModel (functionName="classification" normalizationMethod="logit")

Table 2-1. Spark's MLlib model supporting exporting into PMML.

Below is an example of source code (borrowed from Spark's MLIb web page, see https://spark.apache.org/docs/2.3.0/mllib-pmml-model-export.html) showing how to export into PMML.

```
import org.apache.spark.mllib.clustering.KMeans
import org.apache.spark.mllib.linalg.Vectors

// Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()

// Cluster the data into two classes using KMeans
```

```
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Export to PMML to a String in PMML format
println(s"PMML Model:\n ${clusters.toPMML}")

// Export the model to a local file in PMML format
clusters.toPMML("/tmp/kmeans.xml")

// Export the model to a directory on a distributed file system in PMML format
clusters.toPMML(sc, "/tmp/kmeans")

// Export the model to the OutputStream in PMML format
clusters.toPMML(System.out)
```

2.5 RELATED EU PROJECTS

This section briefly describes some EU projects related to SAFIRE that will be taken into account:

- **PrEstoCloud –** EU H2020-ICT-2016-1 (2016-2019). Dynamic and distributed software architecture that manages cloud and fog resources proactively, while reaching the extreme edge of the network for an efficient real-time Big Data processing.
- **MIKELANGELO** - EU H2020-ICT-2014-1 (2014-2017). Resource management layer for heterogeneous, cloud-based infrastructures, including methodologies, tools, implementations.
- **REPARA** - EU-FP7 (2013-2016). Software engineering methodology, development tools, computer hardware design and analysis, all working hand-in-hand with industrial end-users to achieve a unified programming model for heterogeneous computers developing also the required automated software support tools.
- **DREAMCLOUD** - EU-FP7 (2013-2016). Application performance analysis and tailoring for heterogeneous (embedded and HPC) infrastructures.
- **C2-Net Cloud Collaborative Manufacturing Networks– EU-FP7 (2015-2018).** The goal of C2NET Project is the creation of cloud-enabled tools for supporting the SMEs supply network optimization of manufacturing and logistics assets based on collaborative demand, production and delivery plans. C2NET Project will provide a scalable real-time architecture, platform and software to allow the supply network partners:
  - To create cloud-enabled tools.
  - To support the SMEs supply network optimization of manufacturing and logistics assets based on collaborative demand, production and delivery plans.

- **CREMA – Cloud Based Rapid Elastic Manufacturing -** EU H2020 (2015-2018). CREMA aims at simplifying the establishment, management, adaptation, and monitoring of dynamic, cross-organisational manufacturing processes following cloud manufacturing principles. CREMA will develop the means to model, configure, execute, and monitor manufacturing processes, providing end-to-end support for cloud manufacturing by implementing real systems and testing and demonstrating them in real manufacturing environments.
- **VF-OS** – Virtual Factories Operating System – H2020 EU. Vf-OS offers a manufacturing oriented cloud platform, supporting a multi-sided market ecosystem that provides a range of services for the connected factory of the future, allowing manufacturing companies to develop and integrate better manufacturing and logistics processes. VF-OS will enable the Manufacturing Operating system by providing the following functionalities:

  – Virtual Factory System Kernel.

  – Virtual Factory Device Drivers and Open APIs.

  – Virtual Factory Middleware and Databus.

  – Open Application Development Kit.

  – Cloud Manufacturing Framework.

  – Virtual Factory Components.

3. INNOVATION

As stated in SAFIRE Technical Annex Section 1.4.2, the main originality lies in solving the crucial problem of *how to support process/product optimisation in manufacturing* industry by combining:

– production situation modelling and monitoring, situation analysis and determination.

– predictive analytics.

– dynamic and predictable reconfiguration.

to achieve integrative solutions for:

– self-adaptive process/product reconfiguration.

– supporting feedback loops from products use to both process/product design and production control.

In order to achieve this goal, and specifically regarding Predictive Analytics Platform, this project provides the following innovations:

- A dataflow execution architecture to handle the data extraction, transformation and loading to the framework, differing from the traditional computational architecture of data. A cloud agnostic big data platform will be developed using the latest open source trends. From an architectural point of view the so-called Lambda architecture will be improved using the same engines and storage tools for both fast and slow data.
- Predictive Analytics Elasticity. A platform offering elastic, highly scalable, fault tolerant and high-throughput platform using a distributed, coordinated and clustered system. Exceptional workloads can run on a more powerful cluster by taking more computing resources from the public clouds using cheap instances. When those exceptional workloads have been processed the cluster size will be shrunk to the minimum size in order to save costs (see D.1.4 SAFIRE Concept, Section 4.1.3 Expected Innovation). This resource management will be managed by the platform, isolating the end user from the technical complexities.
- Predictive Analytics Services allows a non-expert in Predictive Analytics to define real-time data gathering and basic but powerful analysis that, otherwise, would require the collaboration of a data scientist expert. A variety of smart predictive models and techniques will be within the reach of manufacturing industry professionals.

Even if some of the features offered by the project can be already available as technical solutions, they are not within easy reach of SMEs in an industrial environment. As a result of SAFIRE developments, a powerful Predictive Analytics Platform will be available for non-experts to use in an industrial environment and with reasonable costs.

In the sections below, some details about those innovations are included.

3.1 DATAFLOW ARCHITECTURE

The platform capabilities will include aggregation, filtering mapping, reducing, etc. New Big Data analytics engines provide a unified engine for doing real-time and batch analytics, and for this project, an implementation on top of those engines that simplifies the overall big data architecture will be developed. From an architectural point of view the so-called lambda architecture will be improved using the same engines and storage tools for both fast and slow data.

3.2 PREDICTIVE ANALYTICS ELASTICITY

Big data platforms usually require big computing clusters with isolated workloads. In order to improve the computational efficiency of such clusters, dynamic resource managers will be used in this project apart from the standard in the Hadoop world. Such resource managers include Nomad, Mesos, Swarm or Kubernetes and all of them try to improve cluster resource utilization sharing computing resources between different tasks via isolation. Moreover, the developed platform will try to exploit the elasticity of the public clouds in order to reduce the costs from its usage (e.g. reduce the cluster size, use spot instances, etc). This feature could be implemented for example on Mesos using the maintenance primitives along with performance metrics.

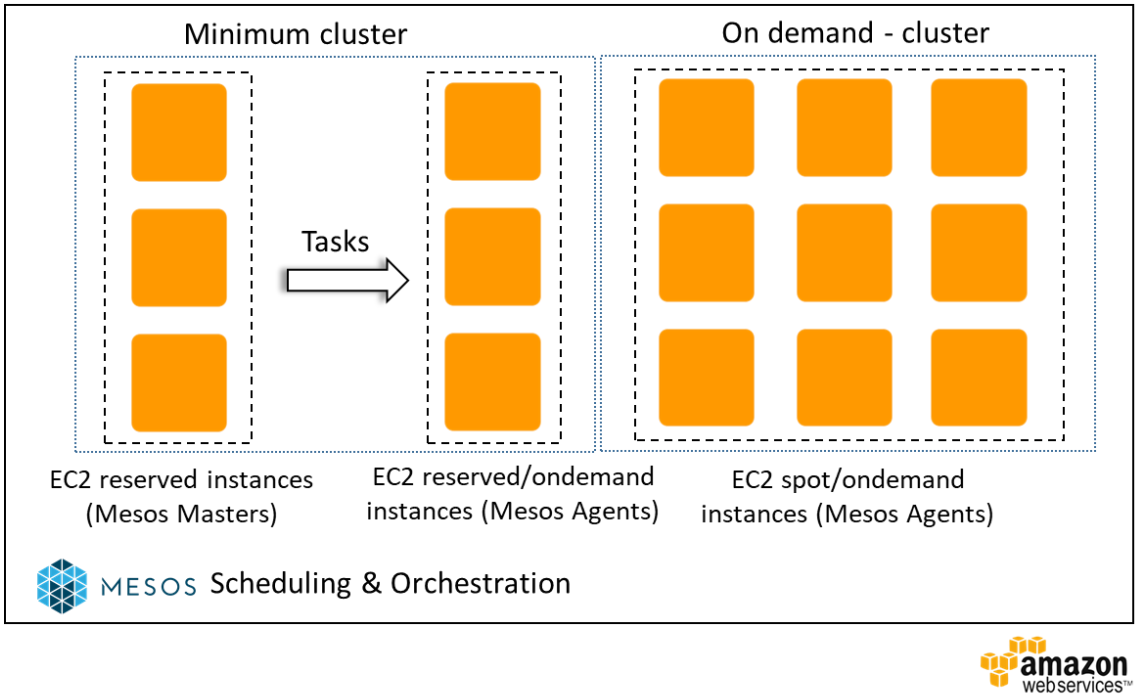


Figure 3-1. Using a Spot based cluster.

Finally, as there is a new research trend in scalable data stores called NewSQL where strong consistency and relational capabilities are added to highly scalable non-relational databases, in this project we will research the maturity and explore capabilities of these new approaches in an industrial environment.

3.3 PREDICTIVE ANALYTICS SERVICE

On the analytical area, we plan to develop new predictive analytics algorithms, which for example can improve the downtime of machines. We plan to explore weakly labelled data from Cyber Physical Systems (CPS) and smart products using a Deep Learning approaches for various tasks such as anomaly detection. Detecting anomalies from CPS data is usually difficult as the CPS usually stay on "normal behaviour". Therefore, algorithms that model the normal behaviour of the CPS will be developed using recurrent neuronal networks or long/short term memory neurons. Another way to look at this problem is to use Boosting-based methods that are providing impressive results for real world problems on Kaggle[1].

The predictive analytics service will be easy to use for non-expert industrial users. For example, end-users will be able to call pre-trained predictive models just passing the input data and asking for predictions, and the service will invoke the model, pass the data, and return the predictions to the client.

[1] https://www.kaggle.com/

4. PREDICTIVE DATA ANALYTICS PLATFORM MODULE SPECIFICATIONS

4.1 SPECIFICATIONS TO HANDLE GENERAL REQUIREMENTS

In this section, General Requirements, named as PA_GR* (Predictive Analytics General Requirement) and their corresponding General Specifications, named as PA_GS* (Predictive Analytics General Specification) are defined.

4.1.1 Requirements

As stated in SAFIRE deliverable *D1.4 Safire Concept* in section 4.1.1, the main general requirements in Predictive Analytics can be summarised in the following point:

- **PA_GR1** - Provide a Real-Time Big Data framework for industrial Data Processing and Analytics to continually improve the manufacturing processes and the final product design, production process and operation itself.

As discussed in *D1.4 Safire Concept* traditional data processing applications are not well adapted to work with large/huge and complex datasets. Frequently these datasets also grow in real-time. Therefore, SAFIRE requires an innovative framework for *real-time big data* processing and *analytics* that overcomes these limitations. The workflow of this framework poses the following requirements:

- **PA_GR2** - Ingestion: It is necessary to be able to define the input and output connectors and how data collection works.
  - Connectors: Define new input connectors from different kinds of production systems, cyber physical systems (CPS) and smart objects (e.g. intelligent products). Besides, integration with different legacy systems and availability on distinct platforms has to be considered.
  - Data Collection: Data ingestion encompasses structured and unstructured data sets, taken from static situational data or streaming situational data in real-time.
- **PA_GR3** –The framework should work in a reliable way across different computer nodes in order to have big data analytics capabilities, for what is termed a computing cluster. As the task of managing the system can have a great impact on these types of systems, a cluster kernel in charge of resource management will be provided. This cluster manager will provide different services such as efficient resource utilisation, task management or service discovery for the different analytics tasks that will run on the cluster.
- **PA_GR4** - Processing/Analytics: The stream data processing offers data aggregation, filtering, mapping, reducing, etc. in a near real-time context.
- **PA_GR5** – Be able to work with production and product data analytics to get more accurate predicted data by means of near real-time data processing.
- **PA_GR6** - Data can be processed and analysed both offline (offline learning) and in real-time.

- **PA_GR7** - An easily configurable dataflow is needed in order to route the information flow; therefore, tools providing such support will be integrated on SAFIRE.
- **PA_GR8** - The Big Data platform will be a PaaS, so it can be deployed both on-premises or on different public cloud providers.

4.1.2 Specifications

In this section, General Specifications (GS*) corresponding to one or more General Requirements (GR*) are defined.

- **PA_GS1** (Satisfies PA_GR1, PA_GR2, PA_GR3) – Safire Software Technology Architecture specification (defined in SAFIRE deliverable *D1.4. SAFIRE Concept*) defines an architecture that is scalable and supports data routing and transformation via NiFi and streaming apps via Kafka. This architecture can store data in a distributed way allowing replicated clusters.
  - Any data source (from different kind of production systems, cyber physical systems (CPS) and smart objects, e.g. intelligent products) can be ingested as long as it has its corresponding connector capable of sending data to SAFIRE via NiFi with a given format. Data ingestion includes structured and unstructured data sets, taken from static situational data or streaming situational data in real-time.
  - A cluster kernel in charge of resource management will be provided. This cluster manager will provide different services such as an efficient resource utilisation, task management or service discovery for the different analytics tasks that will run on the cluster. There can be replicated clusters to guarantee reliability.
- **PA_GS4** (Satisfies PA_GR5) – It is possible to aggregate, filter, map, reduce, etc, the stream data in a near real-time context.
- **PA_GS5** (Satisfies PA_GR5) – As long as data from production processes and the products themselves are uploaded via the corresponding connectors/NiFi and stored into Cassandra, the Predictive Analytics Platform will be able to work with them in near real-time.
- **PA_GS6** (Satisfies PA_GR6) – Incoming data can be analysed/predicted online in near real-time as long as a learning algorithm, executed offline, has learnt to analyse/predict form a collection of past data. Learning algorithms are not real-time as execution time depends on volume of training data.
- **PA_GS8** (Satisfies: PA_GR8) - The Big Data platform will be deployed in a cluster of virtual machines in which the platform will be hosted. It will consist of a variable number of virtual machines, with at least two virtual machines, one as master and another as a computer agent. For better availability, there should be 3 or 5 masters and an odd number of agents due to the need of a strict majority or quorum (an even number gives no benefit over having the previous odd number). Minimum virtual machine requirements are:

- Masters: 4 cores, 16 GB RAM, 120 GB hard drive (fast disks).
- Agents: 2 cores, 16 GB RAM, 60 GB hard drive.

4.2 SPECIFICATIONS TO HANDLE CROSS-COMPONENTS REQUIREMENTS

In this section, Cross Component Requirements, named as PA_CCR* (Predictive Analytics Cross Component Requirement) and their corresponding Cross Components Specifications, named as PA_CCS* (Predictive Analytics Cross Components Specification) are defined.

4.2.1 Requirements

- **PA_CCR1** - Data processing and analysis results must be stored in shared repository accessible via REST.
- **PA_CCR2** - Data to be shared can be published in real-time channels via the distributed messaging system.

4.2.2 Specifications

- **PA_CCS1** (Satisfies PA_CCR1) - Data processing and analysis results will be stored in Cassandra and data will be accessible via a spring web REST service.
- **PA_CCS2** (Satisfies PA_CCR2) – A Publish/Subscribe mechanism will be offered via Kafka.

4.3 SPECIFICATIONS TO HANDLE INDUSTRIAL BUSINESS CASE REQUIREMENTS

This section describes how and which of the requirements gathered in *D1.1 Application Scenarios Requirements Analysis* in (a) *Section 5.6 Data Mining and Analytics* and (b) *Section 5.10 Performance* are covered by these final specifications. In addition to this, the planned features of the early prototype as defined in *D1.4 SAFIRE Concept, Section 5.3.2 Full Prototype Content* have been taken into account.

As described in *D1.1*, three business cases involving three industrial partners will be used in SAFIRE to demonstrate the applicability of the solution. For each one of the business-case industrial partners, the following scenarios were chosen:

Electrolux:

- *Scenario 1*: improve device performance based on feedback
- *Scenario 2*: improve device performance based on historical data
- *Scenario 3*: adaptive control of devices

ONA:

- *Scenario 1*: advanced monitoring and data analytics services for EDM machines

- *Scenario 2*: smart workshop manager / automated EDM manufacturing line

OAS:

- *Scenario 1*: real/run-time reconfiguration
- *Scenario 2*: identification of (specified) events /event causes
- *Scenario 3*: knowledge generation to design

Analysing those scenarios resulted in the requirements described below.

4.3.1 Requirements

In this subsection, Business Case Requirements specifically related to Predictive Analytics are analysed. Naming of each requirement (i.e. U78) corresponds to naming in deliverable *D1.1 Application Scenarios Requirements Analysis*.

4.3.1.1 Data Mining and Analytics Requirements from Industrial Business Cases

Req. No.	Requirement	Overall Priority
U78	Supports data mining to extract useful patterns about operator behaviour	SHALL
U79	Supports data mining to extract useful patterns about machine status	SHALL
U80	Supports data mining to extract useful patterns about production process status	SHALL
U81	Provides support for selection of sensors / systems to be analysed	SHALL
U82	Provides support for selection of information sources to be analysed	SHALL
U83	Provides support for data/sensor composition functionality	SHALL
U84	Able to provide historical knowledge about system deviations or problems	SHOULD
U85	Able to provide decision support for production line selection	SHOULD
U86	Able to increase visibility of the production process	SHALL
U87	Supports analysis for algorithm definition for boiling/temperature control functionality	SHALL
U88	Supports sensitivity analysis to noise	SHALL
U89	Supports main variation factor identification and robust strategy for minimising	SHOULD
U90	Supports computational resources estimation of machines	SHOULD

Req. No.	Requirement	Overall Priority
U91	Supports estimation of performance decrease for algorithm complexity reduction	SHOULD
U92	Supports process repeatability and stability characterisation	SHALL
U93	Supports Design of Experiments (DOE) and Analysis of Variance (ANOVA) analysis	SHOULD

4.3.1.2 Performance Requirements from Industrial Business Cases

Req. No.	Requirement	Overall Priority
U115	Does not negatively affect the usual production processes	SHALL
U116	Support for scalability in the size of cloud and computing resources	SHALL
U117	Support for horizontal scalability to many machines	SHALL
U118	Capable of real-time data ingestion (registering data)	SHALL
U119	Capable of batch processing of data (offline analysis)	SHALL
U120	Capable of real-time data processing	SHALL
U121	Capable of providing real-time reconfigurations / optimisations (subject to network throughput limits)	SHALL
U122	Able to analyse relevant data within a given timeframe	SHALL
U123	Capable of storing up to 5 TB/year/machine with resource recycling facilities	SHALL
U124	Provides support for Machine Learning (Supervised / Unsupervised / Anomaly Detection)	SHALL
U125	Able to achieve required precision on cooking process estimation / optimisations	SHALL

4.3.1.3 Interface Requirements from Industrial Business Cases

Req. No.	Requirement	Overall Priority
U130	Able to access data stored in a	SHALL

Req. No.	Requirement	Overall Priority
	relational database	
U131	Able to receive and send data from/to a remote location	SHALL

4.3.2 Specifications

- **PA_BCS1 Real-Time Analytics** (Satisfies U79, U80, U118) **–** Basic real-time analytics such as KPI calculation.
- **PA_BCS2 Batch Analytics** (Satisfies U84, U80, U124, U115) **–** Basic offline analytics, as for example, predictive learning.
- **PA_BCS3 Scalable No-Sql Storage** (Satisfies U123, U118) – A NoSQL database based will be implemented in the full prototype.
- **PA_BCS4 Data Gathering** (Satisfies U130, U131) – Full data input will be provided for the full prototype operation evaluation.
- **PA_BCS5 Support for different IoT Protocols** (Satisfies U137) – Connectors will be able to store data in SAFIRE Platform via NiFi. SAFIRE Predictive Analytics Platform is not related to IoT protocols. It is responsibility of the corresponding connector to interface with the device/machine and store data via NiFi.

4.4 SPECIFICATION OF GENERIC FUNCTIONALITIES

A typical Predictive Analytics Process setup or configuration [35] has at least the following steps:

- **Define Project** – In this step, the outcome, scope, objectives and involved datasets are identified**.**
- **Data Collection** – Datasets have to be collected, as for example in the case of SAFIRE, through an online mechanism of production/process data gathering that is uploaded to the cloud via business case device connectors.
- **Data Analysis** – In this step, it must be possible to inspect, clean, filter, reduce, transform, etc the data so it is ready for user visualisation, analysis and predictive modelling task.
- **Statistics** – Statistical techniques can be applied to the data to explore relations, correlations, assumption, etc. so useful information is extracted from the data.
- **Predictive Modelling** – Provides the functions to create, train and test models that can learn to correlate/predict a given set of data from past/present sets of data. Main focus of Predictive Analytics module will be here.

- **Deployment** – Provides the ability to deploy analytical results into everyday decision making problems. For example, a given application may request to be notified when a given variable is predicted to have a given value.

A Predictive Analytics Platform has to provide functions to accomplish the most important steps described above. Next subsections describe the functionalities to be provided by the Full Prototype.

4.4.1 Data Collection/Storage Functionality Specification

This functionality is common to all modules and should allow collecting and storing data in the cloud. It will be possible to upload data collected from machines and store into the cloud via machine connector/NiFi/Kafka/Cassandra.

Figure below shows an example of a schema on how data is uploaded and stored for ONA business use case. This example will be described step by step.

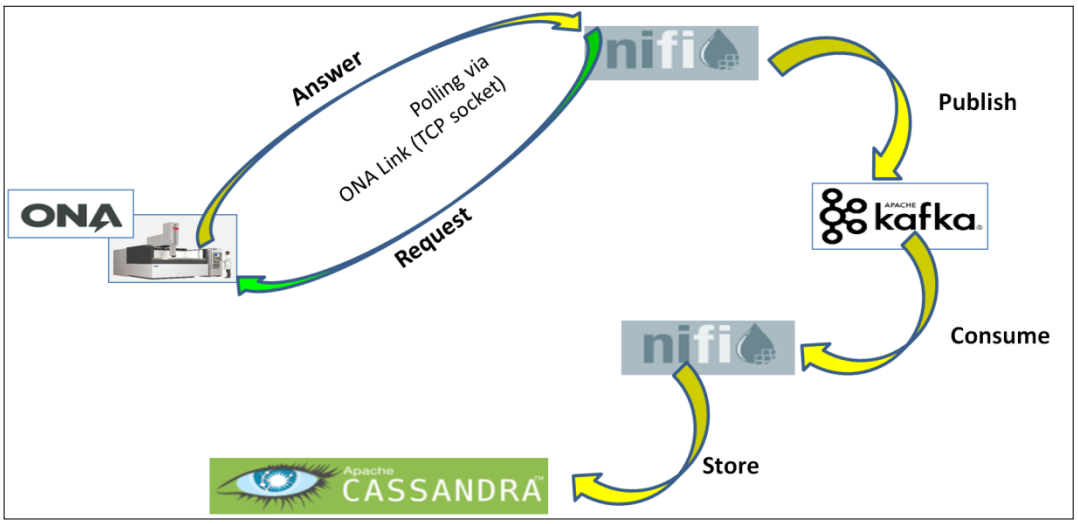


Figure 4-1 – Example of data flow from ONA machine into Cassandra via NiFi/Kafka.

In this example, ONA machine's connectors are implemented as a NiFi processor that can connect ONA machines via ONA Link protocol (via TCP socket). Figure below shows the NiFi processor *SfrOnaLinkRMCDmvarUserRequest* (groovy script) that can be configured to poll variables.

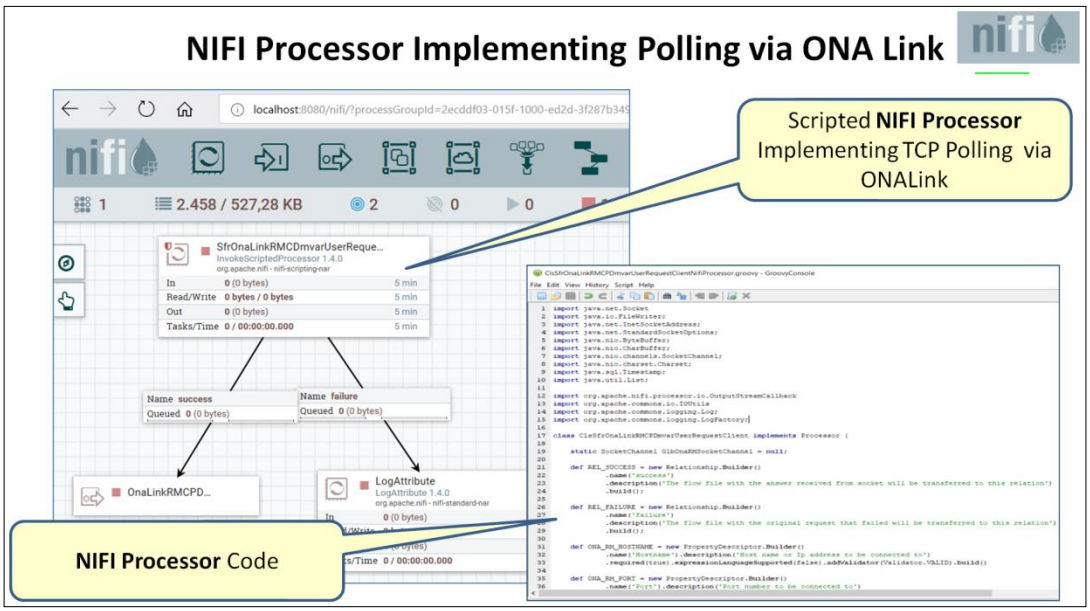
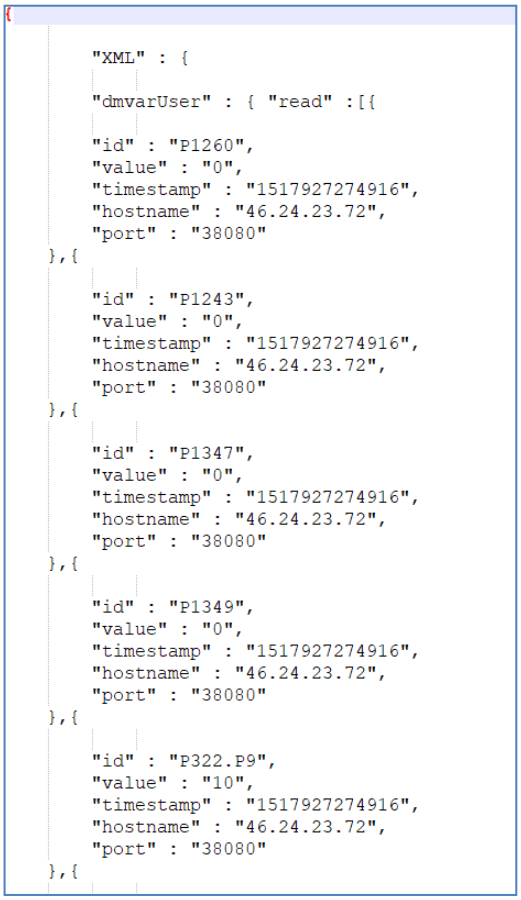


Figure 4-2 – NiFi processor implementing a connector to ONA machines via ONA Link

As the NiFi processor receives values of variables, it builds a *JSON* file containing the data and sends the file to the output port. Figure below presents details of a *JSON* file produced by the connector in which for each variable the following information is detailed:

- ***id***: name of the variable.
- ***value***: value read.
- ***timestamp***: time stamp of reception of the value by the connector.
- ***hostname***: IP address from which the variable was read.
- ***port***: port number of the socket.



```
{
    "XML" : {
    "dmvarUser" : { "read" :[{
    "id" : "P1260",
    "value" : "0",
    "timestamp" : "1517927274916",
    "hostname" : "46.24.23.72",
    "port" : "38080"
},{
    "id" : "P1243",
    "value" : "0",
    "timestamp" : "1517927274916",
    "hostname" : "46.24.23.72",
    "port" : "38080"
},{
    "id" : "P1347",
    "value" : "0",
    "timestamp" : "1517927274916",
    "hostname" : "46.24.23.72",
    "port" : "38080"
},{
    "id" : "P1349",
    "value" : "0",
    "timestamp" : "1517927274916",
    "hostname" : "46.24.23.72",
    "port" : "38080"
},{
    "id" : "P322.P9",
    "value" : "10",
    "timestamp" : "1517927274916",
    "hostname" : "46.24.23.72",
    "port" : "38080"
},{
```


Figure 4-3 – JSON file with data.

Once the data is received via NiFi, (a) another NiFi node publishes the data in a given Kafka topic, (b) the routing/distribution/scalability mechanism sends the data wherever is needed and (c) a given consumer can consume the data from Kafka´s corresponding topic. Figure below shows this flow.

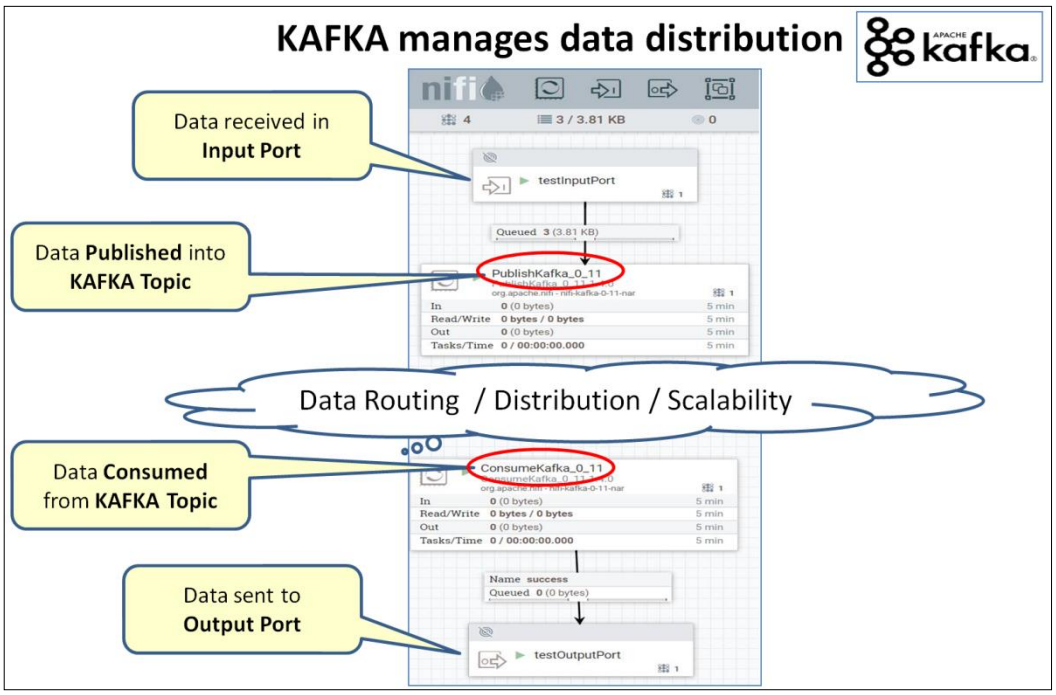


Figure 4-4 - Kafka Manages Data Routing, Distribution and Scalability.

Finally, a sequence of NiFi nodes consume the JSON file from Kafka and store the data into Cassandra. The following figure describes the steps given.

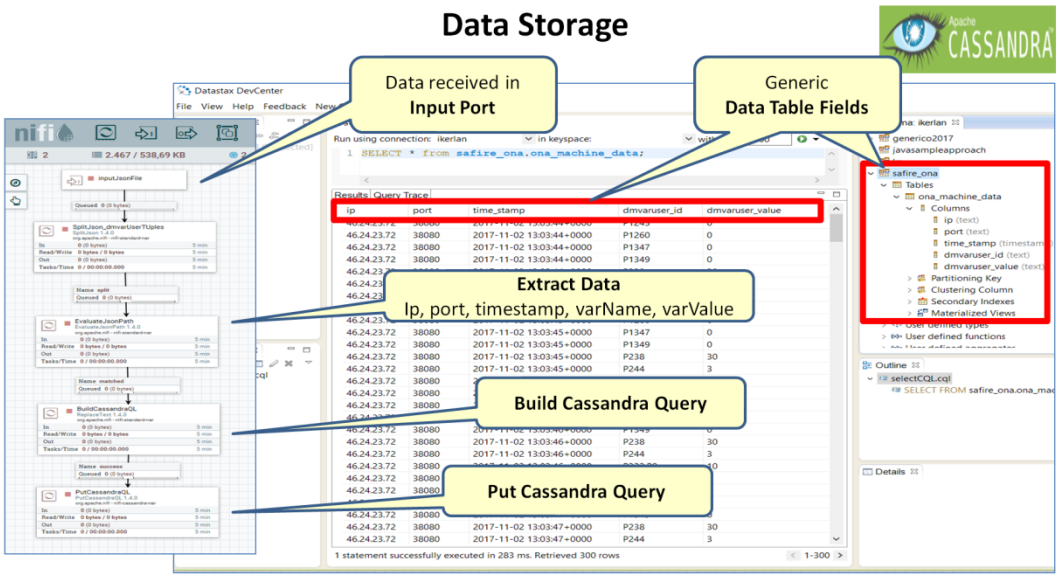


Figure 4-5 – NiFi sequence of nodes storing data into Cassandra.

- Each **JSON file** is received in the sequence input port.
- The file is "**decoded**" to extract the data (IP, port, timestamp, var name, value).
- A Cassandra **query** is built to put the data into Cassandra.

- Finally, a Cassandra Query **Put** is performed to store the data.

4.4.2 Data Query Functionality Specification

For the full prototype, as it was done in the early prototype, a REST Based data query functionality will be developed. This REST API will be compatible with the OpenAPI[2] specification and clients will be automatically generated by Swagger Codegen[3].

The main data query functionality available on the Full Prototype is summarized below:

- Query smart product & smart factory historic production data. E.g. obtain historic data for the given product.
- Query for Analytics results (real-time / batch). One example of this is the prediction of the boiling point based on previous data.

Currently, OpenAPI is under design and complete details will be depicted on the Full prototype specification.

4.4.3 Predictive Modelling Functionality Specification

4.4.3.1 Introduction

In the final prototype *online real-time prediction service functionality* will be implemented, following the SAFIRE architecture (Figure 4-6) that specifies a Predictive Analytics Service developed as Spring REST Web Services. The following subsections specify the services in detail.

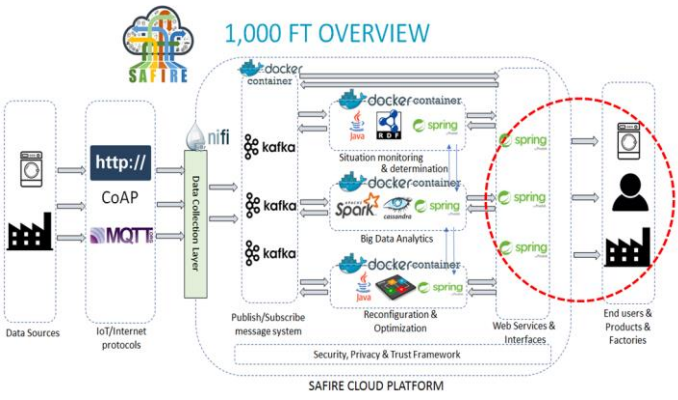


Figure 4-6 - Spring Predictive Analytics REST Web Services.

[2] https://www.openapis.org/

[3] https://swagger.io/swagger-codegen/

Prediction Service

The clients will be able to invoke and load previously trained models, not only Spark models but also Keras/Tensorflow models, as it is shown in Figure 4-7. The use-case of usage, shows the following:

- A *Client* requests a service (1) to the *Prediction Service* consisting on applying a given *predictive model* (trained with a given *backend*, such as *spark* or *keras*) to an *input sample* of values to get some *predicted values*.

- The *Prediction Service* receives the request and (2) invokes the *backend* to load the *predictive model*, (3) gets the predicted values and sends the predicted values back (4) to the *Client*.

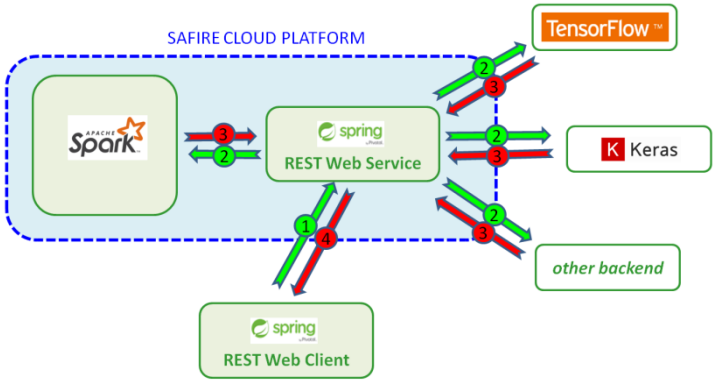


Figure 4-7 - Spring Web Service REST and Web Service REST Client Workflow.

In the following sections, both the service and the client will be described in detail.

Modelling and Training

As mentioned in previous section, generation of predictions with machine learning techniques require a *trained predictive model*. Therefore, two clearly separated activities are:

- *Definition* and *Training* of the *predictive model*.

- *Exploitation* of the model to generate *Predictions*.

Model definition and training is not an obvious task and usually requires expert knowledge. Spark provides dozens of sophisticated machine learning algorithms, transformations and multiple ways to define the architecture of a model. Other packages, such as keras, also offer a wide variety of alternatives to define, train and fine-tune deep learning algorithms.

These types of machine learning packages offer very powerful and sophisticated APIs for big data scientist. Therefore, it is not realistic trying to develop a *Predictive Analytics* ***Training*** *Web Service* with the aim to hide the complexity of such task to non-experts. Such a trial would end up:

- Or with (a) a very simplistic service with very limited functionality.
- Or with (b) a very complex service, even more complex that the API's themselves, and of course useless for non-expert people.

SAFIRE takes the following approach:

- *Regarding model definition and* ***training***, SAFIRE will define ***templates*** consisting in *source code* with examples that will allow non-expert users to define and train medium complexity models.
- *Regarding* ***prediction*** *generation, SAFIRE* will develop a REST Web Service that will allow non-expert users to easily generate ***predictions*** by invoking previously trained models.

Next sections describe ***prediction service specifications*** and some examples of the source code ***templates*** for ***training*** that will be developed in the full prototype.

4.4.3.2 Prediction Service

Predictive Analytics *Prediction* REST Web Service (or simply *Prediction Service*) will be implemented as a:

- *REST Web Service* developed in *Java* with *Spring*. The service will accept client's prediction requests and will answer with predicted values. This service will be accessible in two ways:
  - From a *Web navigator* such as Internet Explorer, Google Chrome, etc.
  - From a BC *REST Web Client* typically developed in java with Spring.
  - Any other SAFIRE modules.

  The service will:
  - Receive an input consisting mainly in a Spark ***`dataframe`*** in JSON format. The input dataframe will consist in a collection of Sparks's *Dataset*`<Row>` representing the samples for which a prediction is required.
  - Invoke a predictive modelling to predict the values according to the input ***`dataframe`***.

- Return the values as additional ***columns*** to the input ***dataframe***.

- *REST Web Clients* developed in *Java* with *Spring*. As mentioned earlier, clients can be BC applications, or other SAFIRE modules. A template of *Spring java REST Web Service's Client* will be developed in Java with Spring so that prediction service can be easily accessible for non experts, writing their own specialized instances of clients in a easy way.

Service Specifications

This section specifies the input parameters of the service and the specification of the answer returned by service.

Service name

- ***String*** `SafirePrdAnalyticsPredictor`

Parameters in the Request

- ***String*** `ipAddress` – Identifies the ip address where the service is located.
- ***String*** `port` – Connection port to the service.
- ***long*** `clientId` – Identifies the client's request. Can be any number provided by the client. This identification will be included back with the answer.
- ***String*** `clientTopic` – Client topic is a string provided by the client. It is simply a complement to the client's request and might be the empty string. This topic will be included back with the answer and can help the client to indentify better the answer. An example of client's topic may be "*Boil_detection_25-Oct-2018_16-51-00*" that identifies a boiling experiment.

- ***String*** `modelName` – Upon request, the service will (a) invoke and load a previously trained predictive analytics model and (b) will call the model to predict values according to *dataFrameRowDataJSON* parameter (see below).

- ***String*** `backendName` – Indicates the backend that will process the invocation. Allowed values are: ***spark*** or ***keras***

- ***String*** `dataFrameColNamesJSON` – Contains the dataframe column names in JSON format, according to the following syntax:

  ```
  {"dataFrameColNames":["name1","name2",....]}
  ```

  Example (three columns case):

  ```
  {"dataFrameColNames":["id","text","label"]}
  ```

- ***String*** `dataFrameColTypesJSON` – Contains the dataframe column types in JSON format. Allowed types are *integer*, *double*, *string*, *arrayInteger*, *arrayDouble*. Syntax is as follows:

```
{"dataFrameColTypes":["type1","type2", ...]}
```

Example (three columns case):

```
{"dataFrameColTypes":["integer","string","double"]}
```

- ***String*** `dataFrameRowDataJSON` – Contains the dataframe rows in JSON format. Each row must have the number of values specified in *dataFrameColNamesJSON* with its corresponding type specified in *dataFrameColTypesJSON*. Syntax is as follows:

```
{"dataFrameRowData":
      [[row1data1, row1data2, ...],
      [row2data1, row2data2, ...],
      [row3data1, row3data2, ...],
                  ...
      ]}
```

Example 1 (two rows with three columns of type *integer*, *string*, *double*):

```
{"dataFrameColNames":["id","text","label"]}
{"dataFrameColTypes":["integer","string","double"]}
{"dataFrameRowData":
      [[7,"this is an example ssd", 1.0],
       [8,"another text", 0.0]]}
```

Example 2 (two rows with one column of type *arrayDouble*):

```
{"dataFrameColNames":["currentValues"]}
{"dataFrameColTypes":["arrayDouble"]}
{"dataFrameRowData":
      [[[1.456, 2.3456, 3.2345, 1.3456]],
      [[2.3737, 4.2829, 1.2876, 8.7625]]}
```

Answer given by the service

The service will always return a JSON string containing the following fields:

- `long callCount` - Represents an automatic counter with the number of times the service has been requested since it was started (just informative purpose).
- `long clientId`- The client identification that was provided by the client in the request.
- `long clientTopic` - The client topic that was provided by the client in the request.
- `String modelName` – The predictive model that was provided by the client in the request.
- `String backendName` – The backend that was provided by the client in the request.

- `String dataFrameRowDataPredictionJSON` – In this parameter, the service returns in this parameter the predicted values for each row received in the request's param `dataFrameRowDataJSON`. The syntax of the JSON string (similar to that of `dataFrameRowDataJSON`) is the following:

```
{"dataFrameRowDataPrediction":
      [[Row1Prediction1, Row1Prediction2, ...],
       [Row2Prediction1, Row2Prediction2, ...],
       [Row3Prediction1, Row3Prediction2, ...],
                    ...
      ]}
```

  Example: (predicted values for three rows, where each predicted value is a double):

```
{"dataFrameRowDataPrediction":
      [[0.9878],
       [0.45627],
       [0.87265]
      ]}
```

  Note: The number of predicted values per row and their types is implicitly defined in the predictive model, but not defined in the request. Therefore, the client receiving the answer must know the expected number and types of fields.

- `String errorDescription` – The description of the error when the service execution fails (retCode <> 0).
- `int retCode` – Return code value is 0 when the service execution succeed, and non-zero otherwise.

Example of Request

The client sends a request as follows (Electrolux case example):

```
http://localhost:8080/SafirePrdAnalyticsPredictor?
clientId=1&
clientTopic=Boil_detection_26-10-2018_10-57-41&
modelName=electroluxNNTraineModelCurF08.h5&
backendName=keras&
dataFrameColNamesJSON=
      {"dataFrameColNames":["currentValues"]}&
dataFrameColTypesJSON=
      {"dataFrameColTypes":["arrayDouble"]}&
dataFrameRowDataJSON=
      {"dataFrameRowData":[[[1.54418102,  1.48782741, ... ]]]}
```

Example of Answer

The service processes the request and answers with the following:

```
ClsSafirePrdAnalyticsPredictorWebServiceAnswer {
        callCount=1,
        clientId=1,
        clientTopic= Boil_detection_26-10-2018_10-57-41,
        modelName=electroluxNNTraineModelCurF08.h5,
        backendName=keras,
        prediction={"dataFrameRowDataPrediction":[[0.9015398025512695]]},
        errorDescription="",
        retCode=0}
```

In this particular case, the answer contains the prediction of the single sample passed as parameter being boiling (90,15%).

Invoking the Prediction Service from a Web Navigator

The Prediction Service will also be callable from a Web navigator. Figure 4-8 shows a call to the service executed from a web navigator (in this case Google Chrome) and the response given by the service (the same as shown in Section 4.4.3.1).

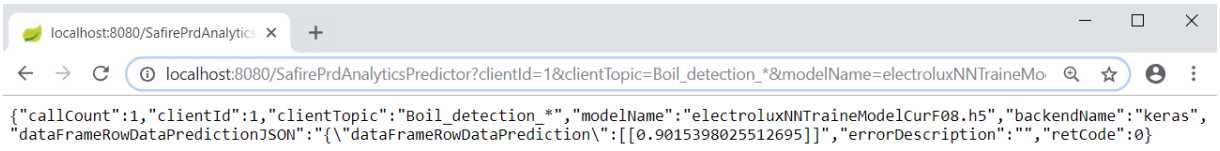


Figure 4-8 – Predictive Analytics Web Service Call from a Web Navigator.

Template for invocation from a Java Client Specifications

SAFIRE full prototype will develop Source Code templates for an easy development of REST java clients. These templates will be fully explained in the final full prototype development report. As an example of specifications, a template will be composed by the two classes described below.

Class ClsSafirePrdAnalyticsPredictorWebServiceAnswer

This class represents a Java client. Only the section with *TODO* must be modified by the end-user using the template:

```
@SpringBootApplication
public class ClsSafireWebServiceRestClientTemplate  {

        private static final Logger log =
                LoggerFactory.getLogger(ClsSafireWebServiceRestClientTemplate .class);
```

```
public static void main(String args[]) {
        SpringApplication.run
                (ClsSafireWebServiceRestClientTemplate .class);
}

@Bean
public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
}

@Bean
public CommandLineRunner run(RestTemplate restTemplate) throws Exception {

        return args -> {

                // Call service
                CallService(restTemplate);
        };
}

private void CallService(RestTemplate restTemplate) {

        // TODO
        // Define call parameters
        String port = "8080";
        long clientId = 1;
        String topic = "Boil_detection_29-oct-2018_12-08-00";
        String modelName = "electroluxNNTraineModelCurF08.h5";
        String backendName = "keras";

        // TODO
        // Generate the sample to predict
        // The key dataframe elements must be defined
        // The columns names, types and data values
        String dataFrameColNamesJSON =
                        "{\"dataFrameColNames\":[\"currentValues\"]}";
        String dataFrameColTypesJSON =
                        "{\"dataFrameColTypes\":[\"arrayDouble\"]}";
        String dataFrameRowDataJSON =
                        "{\"dataFrameRowData\":"
                        + "[[[1.456, 2.3456, 3.2345, 1.3456]],"
                        + "[[2.3737, 4.2829, 1.2876, 8.7625]]}";

        // Encode the data frame elements
        // This is necessary as they contain reserved chars for http requests
        dataFrameColNamesJSON =
                UriUtils.encodeQueryParam(dataFrameColNamesJSON,"UTF-8");
        dataFrameColTypesJSON =
                UriUtils.encodeQueryParam(dataFrameColTypesJSON,"UTF-8");
        dataFrameRowDataJSON =
                UriUtils.encodeQueryParam(dataFrameRowDataJSON,"UTF-8");

        // Build the call to the prediction service
        String serviceCall =
                "http://localhost:" + port + "/SafirePrdAnalyticsPredictor?" +
                "clientId=" + String.valueOf(clientId) + "&" +
                "clientTopic=" + topic + "&" +
                "modelName=" + modelName + "&" +
                "backendName=" + backendName + "&" +
                "dataFrameColNamesJSON=" + dataFrameColNamesJSON  + "&" +
                "dataFrameColTypesJSON=" + dataFrameColTypesJSON  + "&" +
                "dataFrameRowDataJSON=" + dataFrameRowDataJSON;

        // Call the prediction service
        ClsSafirePrdAnalyticsPredictorWebServiceAnswer answer =
                restTemplate.getForObject
                        (serviceCall,
                        ClsSafirePrdAnalyticsPredictorWebServiceAnswer.class);
```

```
                // TODO
                // Process the answer
                // In this case just print to log
                log.info(answer.toString());
        }
}
```

Class ClsSafirePrdAnalyticsPredictorWebServiceAnswer

This class represents the answer given by the prediction service and does not need any modifications, and can be used as-is:

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class ClsSafirePrdAnalyticsPredictorWebServiceAnswer {

    // Represents an automatic counter
    // with the number of times the
    // service has been requested
    private long callCount;

    // Id value passed by the caller
    // Will be returned back as it is
    private long clientId;

    // Topic value passed by the caller
    // Will be returned back as it is
    private String clientTopic;

    // Prediction Model name
    // requested by the caller
    private String modelName;

    // Prediction engine backend
    // requested by the caller
    // Allowed values are: spark, keras
    private String backendName;

    // List of Predicted Data Frame Rows values
    // Contains a JSON list with the Predictied Rows
    // produced by the model. It is responsible
    // of the caller to interprete the meaning of
    // the values
    private String dataFrameRowDataPredictionJSON;

    // Error description
    // when retCode != 0
    private String errorDescription;

    // 0-Success, <>0-Error
    private int retCode;



    public ClsSafirePrdAnalyticsPredictorWebServiceAnswer() {
    }

    public long getCallCount() {
        return callCount;
    }
    public void setCallCount(long callCount) {
        this.callCount = callCount;
    }

    public long getClientId() {
        return clientId;
    }
    public void setClientId(long clientId) {
        this.clientId = clientId;
```

```
    }

    public String getClientTopic() {
        return clientTopic;
    }
    public void setClientTopic(String clientTopic) {
        this.clientTopic = clientTopic;
    }

    public String getModelName() {
        return modelName;
    }
    public void setModelName(String modelName) {
        this.modelName = modelName;
    }

    public String getBackendName() {
        return backendName;
    }
    public void setBackendName(String backendName) {
        this.backendName = backendName;
    }

    public String getDataFrameRowDataPredictionJSON() {
        return dataFrameRowDataPredictionJSON;
    }
    public void setDataFrameRowDataPredictionJSON(String dataFrameRowDataPredictionJSON) {
        this.dataFrameRowDataPredictionJSON = dataFrameRowDataPredictionJSON;
    }

    public String getErrorDescription() {
        return errorDescription;
    }
    public void setErrorDescription(String errorDescription) {
        this.errorDescription = errorDescription;
    }

    public int getRetCode() {
        return retCode;
    }
    public void setRetCode(int retCode) {
        this.retCode = retCode;
    }

    @Override
    public String toString() {
        return "ClsSafirePrdAnalyticsPredictorWebServiceAnswer {" +
                "callCount = " + Long.toString(callCount) +
                ", clientId = " + Long.toString(clientId) +
                ", clientTopic = " + clientTopic +
                ", modelName = " + modelName +
                ", backendName = " + backendName +
                ", prediction = " + dataFrameRowDataPredictionJSON +
                ", errorDescription = " + ((retCode != 0) ? errorDescription : "Ok") +
                ", retCode = " + Long.toString(retCode) +
                '}';
    }
}
```

Speed Specifications

The SAFIRE project aims at real-time processing and therefore Web Service execution time must meet that requirement. However, real-time is a concept relative to the application and the requirements can be different for each application. For example, in the case of Electrolux boiling detection, real-time means basically the order of one

second. Execution speed depends obviously on the connection but also in the predictive model complexity.

As a general requirement, for medium size models and good quality connection, real-time requirement will be understood as execution time in the order of a few seconds.

4.4.3.3 Modelling and training Templates

SAFIRE's full prototype will develop Source Code templates for an easy development of predictive model *definition* and *training*. Examples developed for Business Cases will be developed as instances of theses templates and will be fully explained in the full prototype report.

As an example of template, below is the python source code for a simple template of a *logistic regression* (this function is part of several source code files).

```
def trainModelLR(dataFrame, dataFrameFeatureColNames):

    # Assemble the input to produce the features column
    assembler = VectorAssembler(inputCols=dataFrameFeatureColNames, outputCol="features")

    # TODO
    # Select the machine learning algorithm and its parameters
    # In this case a Logistic Regression has been selected
    lr = LogisticRegression(maxIter  = 100, regParam = 0.01)

    # Chain in a pipeline the transformations
    # and machine learning algorithm
    pipeline = Pipeline(stages = [assembler, lr])

    # TODO
    # Create a Parameter Grid for Cross Validation
    # Assign a range to the hyper parameter for fine-tuning
    paramGrid = (ParamGridBuilder()
                .addGrid(lr.regParam, [0.01, 0.1, 0.3, 0.5]) # regularization parameter
                .addGrid(lr.maxIter, [10,25, 50, 100]) # regularization parameter
                .addGrid(lr.elasticNetParam, [0.0, 0.1, 0.2]) # Elastic Net Parameter(Ridge=0)
                .build())

    # Define cross validation model
    crossval = CrossValidator(estimator=pipeline,
                              estimatorParamMaps=paramGrid,
                              evaluator=BinaryClassificationEvaluator(),
                              numFolds=5)

    # Fit (train) the model
    model = crossval.fit(dataFrame)

    # Return the model bestModel
    return model
```

As another example of template, below is the python source code for a simple template of a *random forest tree* (this function is part of several source code files). It is interesting to note here that, following Spark's philosophy, it is very easy to interchange the algorithms to use to experiment with different alternatives. The code of the *logistic regression* and the *random forest tree* is very similar.

```
def trainModelDT(dataFrame, dataFrameFeatureColNames):
```

```
# Assemble the input to produce the features column
assembler = VectorAssembler(inputCols=dataFrameFeatureColNames, outputCol="features")

# TODO
# Select the machine learning algorithm and its parameters
# In this case a Decision Tree Classifier has been selected
dt = DecisionTreeClassifier()

# Chain in a pipeline the transformations
# and machine learning algorithm
pipeline = Pipeline(stages = [assembler, dt])

# TODO
# Create a Parameter Grid for Cross Validation
# Assign a range to the hyper parameter for fine-tuning
paramGrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [5, 10, 15, 20])
             .addGrid(dt.maxBins, [5, 10, 20, 40])
             .build())

# Define cross validation model
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=5)

# Fit (train) the model
model = crossval.fit(dataFrame)

# Return the trained model
return model
```

4.4.3.4 *Testing with Electrolux BC*

Implementation in the final prototype will be tested with the Electrolux Boiling Point detection test case. Figure 4-9 shows the on-line detection process. First (1) the cooking process is continually (second by second) uploading currents data (currents in the coil), (2) the predictive analytics *prediction service* is called to decides if (with the data available so far) the water is boiling, and (3) when the water is boiling, the cook is notified.

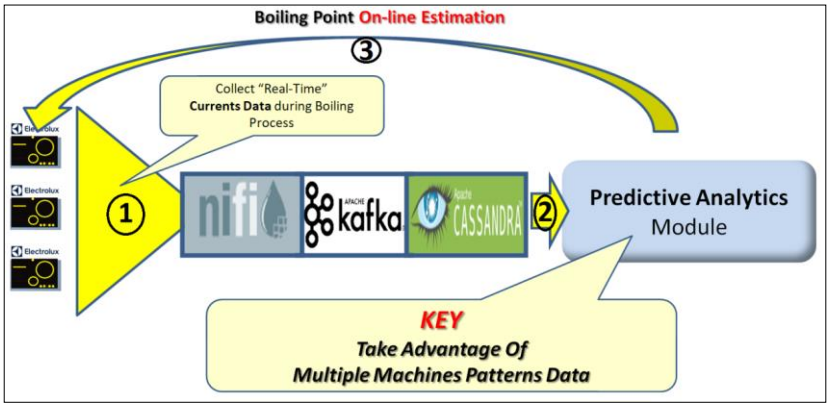


Figure 4-9. *Electrolux – Online estimation of boiling point.*

Predictive Analytics source code templates will be used to experiment with different deep learning alternatives to show the power of SAFIRE to predict the boiling point of a cooking process (Figure 4-10). Alternatives to test in full prototype will be:

- *Spark* – Define Regressions, Decision Trees, etc.
- *Keras/TensorFlow* – Define neural networks alternatives to those already experimented in the early prototypes (with good results).

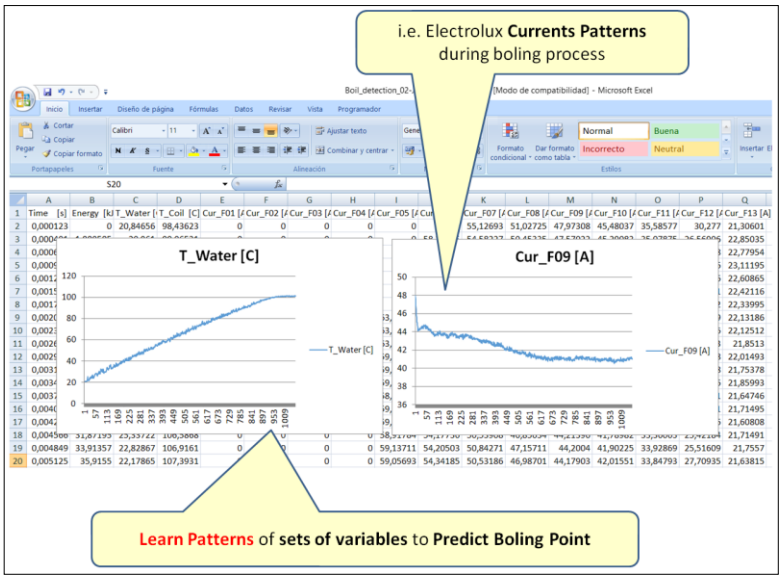


Figure 4-10. *Electrolux Boling Experiment showing detail of water temperature and current F09.*

4.4.3.5 Testing with ONA Electroerosion BC

Implementation in the final prototype will be tested, among others, with the ONA Electroerosion WEDM cutting process test case, trying to detect the event of width changing during cutting (Figure 4-11).

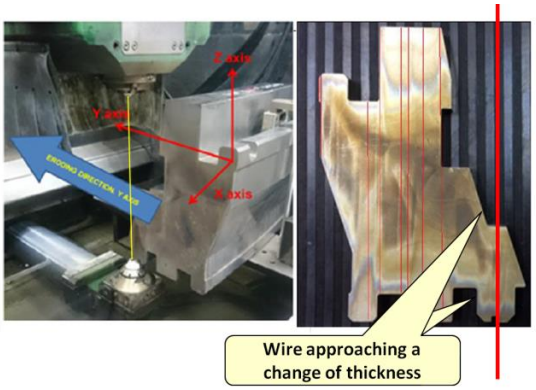


Figure 4-11: WEDM cutting with changing thickness parts.

WEDN process is generated by short electrical discharges between the cutting wire of the machine and the part to be machined through a dielectric fluid (deionised water). Figure 4-12 shows the voltage/time profile of several discharges.

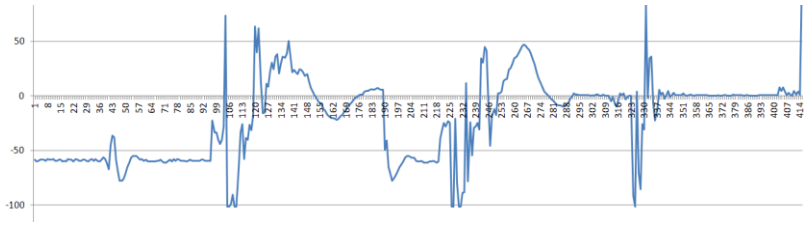


Figure 4-12: Voltage profile of several discharges.

Predictive Analytics work initiated in the ONA test case in the Early prototype, trying to predict the width changing from the discharges voltage patterns, will be completed in the full prototype by using source code templates to generate experiment with different deep learning alternatives to show the power of SAFIRE to predict the part's width change in advance. Alternatives to test in full prototype will be:

- *Spark* – Define Regressions, Decision Trees, etc.
- *Keras/TensorFlow* – Complete neural networks experiments initiated in the early prototype.

4.4.4 Data Quality Assurance Specification

Big data technologies were developed because traditional technologies (and the human beings using them) could not handle such amounts of data, and they carry out a great deal of automation and algorithmic decision making. This lesser human involvement

means that the quality of the data carries even greater importance than it did before data went “big”.

There is no consensus on how to define data quality. However, it is commonly measured in six dimensions in the literature: completeness, timeliness, conformity, integrity, consistency and accuracy. The following sections describe how data quality issues could raise in any of these dimensions, and how they are tackled in the data ingestion module of the SAFIRE project.

Figure 4-13. Dimensions of Data Quality.

Completeness

Completeness refers to all necessary data being present. Data can be complete without all fields having values because some fields are optional, such a person’s middle name or secondary phone number.

In SAFIRE’s data ingestion module, data is dumped into two independent services: a relational database for persistence, and a publish/subscribe broker for further redistribution of data. The check for completeness is carried out by the relational database. In most relational databases, fields of a table can be declared either as allowing a missing (null) value or not. A null value represents a field without a value. In such cases, if a row with a null value for a field not declared as allowing null is attempted to insert, the insertion will fail and raise and error.

Timeliness

Timeliness refers to whether the data is available or not when it is expected to be. For example if data from Monday is available at midnight on Tuesday to generate daily reports. Timeliness issues can raise from a wide variety of problems: a thunderstorm could take down the power of a server, another server might be running out of memory or disk space, slowed down significantly and stopped accepting new incoming data… if any link in a chain fails, it will result in the rest of the data pipeline not being able to comply to timeliness. This dimension is one of the hardest to ensure because full connectivity is usually out of the developer's control and issues can keep happening even if the rest of the system has been thoroughly tested and corrected. Also, any other dimension not being met will cause timeliness issues because the data pipeline will be halted by the issue.

In SAFIRE, we first tackled this by adding a monitoring node within the data ingestion system. This monitor observes the industrial data incoming from their respective API and waits until the data stream is stopped. If no new messages are received within a small but significant amount of time, an alarm is raised and the operator gets an email with the latest log file from the data ingestion system. The stream can be cut by different errors that can be grouped into two groups:

- Expected errors such as disconnections from either the server, or local network.
- Unexpected errors that have not been handled yet.

Expected errors are already handled by the system and trigger a reattempt after a prudent delay of a few seconds or minutes so as to avoid unnecessary attempts until the connection is restored. Unexpected errors, on the other hand, are unrecoverable until they have been analyzed and the system has been adjusted to handle them. In these instances, the operator receiving the latest log file ensures that the log containing the error will not be lost if the operator is not quick enough to identify the cause in the time it takes the temporary log file to be overwritten.

If the stream is restored, for example when a reconnection attempt has succeeded, the operator also receives an email with a notification so as to avoid unnecessary workload if the error has been handled.

However, this is just one small attempt to ensure data timeliness. The system that was just described is an internal mechanism; it is part of the system that it monitors. If the entire server or datacenter fails, the monitor system will shut down as well, and no alert will be generated. This system has to be complemented with an external monitoring system that checks whether the data ingestion module has placed the data where it is supposed to do.

Conformity

The conformity dimension refers to whether data complies with a set of predefines standards. Examples of these standards can be dates following the same *"yyyy-MM-dd"* format or the messages following the same JSON structure.

The data ingestion system makes several checks in this regard:

- As soon as JSON messages are received through the data stream, it is checked that they do contain the expected data node.
- When messages are processed, it is done so in a way that if they do not conform to the expected format, the processing will fail and raise an alert.
- Dates are handled using the expected format. The process will fail if the format is not the expected.

Integrity

This dimension refers to each piece of data being connected to other data. For example, a violation of this dimension would be, in the domain of manufacturing, a machine having a location field pointing to a location not present in the database.

Just like with the Completeness dimension, the control over the integrity dimension is carried out by the relational database. Relational databases describe tables and the relationships between them, and if configured to do so, enforce that these relationships are met. Returning to the example presented before, a relational database will not allow the insertion of a *machine* record, if the *location* of this record points to a location not present in the *location* table.

Consistency

Data being consistent means that all systems reflect the same information. This does not necessarily only refer to two identical fields in two datasets having the same value, but also cases where one value can be inferred from another, for example birthday and age.

This usually requires creating a monitoring system independent of any data repository that periodically checks the consistency of data across different systems by checking it against a predefined set of specific rules.

In SAFIRE, the data ingestion module has such a monitoring system. However, none of the use cases require of value inference so only direct comparisons between data items in different systems are made.

Accuracy

Data accuracy is one of the hardest dimensions to measure, because it is hard to assess if data is accurate or not without assuming the first data entry point already receives accurate data. However it is a very important dimension, because if data is detected not to be accurate, the depending data pipelines should be killed.

On the one hand, data can be checked against a predefined schema to assess the accuracy of the structure of the data. On the other hand, checking for values requires other approaches. Some data fields can only have a finite set of values. In these cases, at

least a sanity check can be performed to see if whether the actual value belongs to this finite set. With numerical fields that can take an infinite number of values, checking these values can get tricky. These fields present variability by nature, and it can be hard to define how much of variability is acceptable: too low of a threshold and a lot of false positives will be generated, leading to *incident fatigue* and the human operator ignoring future alerts to his or her own discretion; too high of a threshold and most incidents will go unnoticed, damping the effectiveness of the monitoring. A middle ground can be achieved establishing two thresholds. If the variability crosses the first threshold (e.g. 10%), the process will continue but a human operator will be alerted and urged to check whether the variability is legitimate. If the variability crosses the second threshold (e.g. 30%), the human operator will also be alerted, but in this case the process will be stopped.

In SAFIRE's data ingestion module the data's structure is checked against schemas in various steps of the process. The values of discrete fields are handled by the integrity checking measures of the relational database as all the possible values for discrete variables are stored in their respective tables. Finally, regarding numeric fields that can take an infinite number of values, a threshold-based monitoring system has been put in place. This monitoring system will be configured with acceptable ranges for each of the monitored fields. The two threshold system described in the paragraph above will be implemented.

4.4.5 GDPR Compliance Specification

Currently, none of the use cases of the SAFIRE project handle personal data. However, if future cases where the SAFIRE technologies and methodologies are applied, do handle personal data, and this data pertains to EU citizens, the businesses handling this information will have to comply with Regulation (EU) 2016/679, also known as the General Data Protection Regulation or GDPR.

This regulation came into effect on May 25th, 2018 and aims to protect personally identifiable data (PII) every step of the way while giving the consumer ultimate control over what happens to that data. Any business handling PII of European Union citizens must adhere to this regulation, even in the company itself resides outside of the EU.

Consent must be requested in a clear, easy to understand way, with users knowing exactly what they are giving their consent to. Consumers must be provided with tools to control, monitor, check, and delete data related to them if they want to. In fact, revoking consent should be made as easy as giving it.

The GDRP also regulates the protection of the PII. The regulation promotes pseudonymisation of data: where business logic data is anonymised by removing the identifiable data but keeping some sort of way to single out an individual user, for example using a "user id", and keeping the PII data somewhere else. Other accepted approaches are the complete anonymisation of data so it cannot be tied back to an individual, or encryption. For some types of organizations the GDPR requires hiring a Data Protection Officer (DPO). Furthermore, the regulation requires mandatory breach notification to affected individuals within 72 hours of the discovery.

Another aspect related to big data and covered by the GDPR is profiling: algorithmic inference drawn from data about an individual, a tool widely used in big data. The GDRP regulates the use of profiling by trying to distinguish benign and harmful uses of it and allowing citizens not to be subject to fully automated profiling. The definition of harmful uses of profiling was discussed throughout the development of the regulation as "which produce(s) legal effects concerning him or her or similarly significantly affects him or her". The goal of the regulation is not to forbid the use of profiling, but providing the affected individuals with information about the logic behind it, the significance and consequences of it for the individual, disclosing the use of such automated decision-making upon request, and the basis to request human intervention in the process, and providing the individual with the right "to express his or her point of view and to contest the decision".

Currently, none of the use cases of the SAFIRE project handle personally identifiable data. Given the manufacturing application field of the project, it is unlikely that even future implementation cases will handle such data. However it is possible that such unforeseen new cases might come to be. In these cases, the involved companies must comply to the GDPR if they handle data pertaining EU citizens.

4.5 SPECIFICATION OF HIGH-LEVEL ARCHITECTURAL DESIGN

The Full Prototype architecture consists in a set of Virtual machines running in a secured public cloud environment, more concisely in a Virtual Private Cloud that is secured as it is running inside a private isolated network. Each machine will be provisioned with different software tools and configured to support most of the platform's required features.

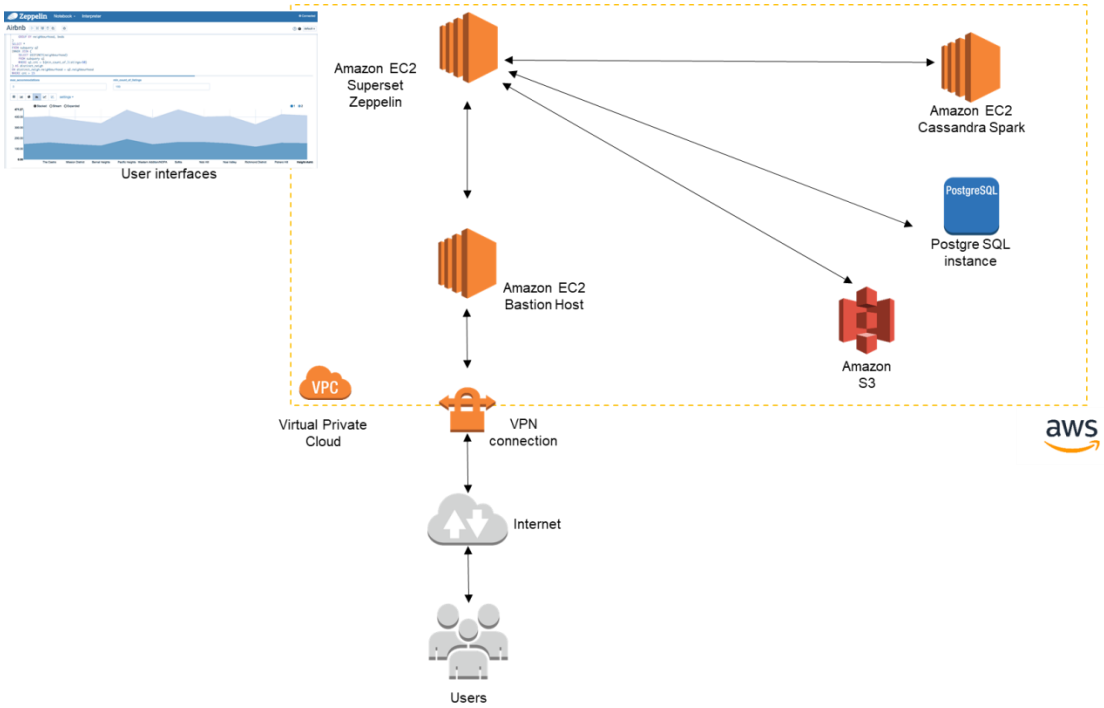


Figure 4-14. *Early end Full Prototype architecture.*

In Figure 4-13 the overall architecture can be seen. Below is the description of the different components:

- Bastion host: This machine will be the entry point of the platform for external users providing a secured VPN tunnel for accessing the private network and therefore allowing access to the different services.

- GUI Machines (Superset/Zeppelin): this machine will be the entry point for both data science prototyping via Zeppelin and advanced visualization via Superset.

- Cassandra/Spark: this machine will be provisioned with the No-SQL database along with the distributed analytics engine.

- PostgreSQL: this is the relational SQL database in charge of storing relational data of the platform. In this case, the one provided by Amazon Web Services will be used as it will allow us to use a standard database without taking care of its maintenance.

- Amazon S3: Amazon S3 is a Software as a Service (SaaS) that provides storage service through web service interfaces. It allows storing data in a reliable way easily and has support for almost all the Big Data landscape tooling.

5. TECHNOLOGY SPECIFICATION OF SOFTWARE TOOLS

This section provides a list of software tools to be used for the Predictive Analytics Platform. All tools are open-source software.

- **Java**

  Programming Language – www.java.com

- **Python**

  Programming Language - https://www.python.org/

- **IDE Eclipse**

  Development Environment – http://www.eclipse.org

- **Apache Tomcat**

  Runtime Environment/Application Server - http://tomcat.apache.org/

- **Apache NiFi**

  Scalable data routing, transformation, and system mediation logic - https://nifi.apache.org/

- **Apache Kafka**

  Distributed Streaming Platform - https://kafka.apache.org/

- **Apache Spark**

  Unified analytics engine for large-scale data processing - https://spark.apache.org/

- **Apache Cassandra**

  Distributed NoSQL database management system- https://cassandra.apache.org/

- **Apache Zeppelin**

  Web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala and more- https://zeppelin.apache.org/

- **Apache Superset**

  Business intelligence web application - https://superset.apache.org/

- **Redis**

  In-memory data structure store, used as a database, cache and message broker- https://redis.io/

- **Terraform**

  Tool for building, changing, and versioning infrastructure as code, safely and efficiently - https://www.terraform.io

- **Ansible**

Software that automates software provisioning, configuration management, and application deployment - https://www.ansible.com/

- **Keras**

  High Level Neural Networks - https://keras.io/

- **Tensor Flow**

  Machine Learning Software - https://www.tensorflow.org/

- **Deeplearning4j**

  Library of deep learning for Java - https://deeplearning4j.org/

- **Spring**

  Web Application Framework - http://www.springsource.org/

- **Hibernate**

  Data Persistence - http://hibernate.org/

- **Docker**

  Software Containerization Platform - https://www.docker.com/

- **GIT**,

  Version Control - http://git-scm.com/ http://git-scm.com/

6. FULL PROTOTYPES FEATURE SET

The full prototype presents the current feature set:

- Integration with Apache Kafka for data ingestion.
- Real-time/batch analytics capabilities using Spark Manchine Learning MLlib library and advanced analytics using Tensorflow & Keras.
- Data Science rapid Prototyping using Apache Zeppelin, this tool allows basic visualizations too.
- NoSQL storage using Cassandra.
- Platform High Availability.
- SQL storage using PostgreSQL.
- Advanced Visualization capabilities using Apache Superset.
- REST APIs for interaction with other SAFIRE modules.
- Scripting using Terraform and Ansible for deployment of the platform on Amazon Web Services

7. REQUIREMENTS COVERAGE

7.1 DATA MINING AND ANALYTICS REQUIREMENTS FROM INDUSTRIAL BUSINESS CASES

This table represents the coverage for the full prototype.

Req. No.	Requirement	Overall Priority	Coverage
U78	Supports data mining to extract useful patterns about operator behaviour	SHALL	True
U79	Supports data mining to extract useful patterns about machine status	SHALL	True
U80	Supports data mining to extract useful patterns about production process status	SHALL	True
U81	Provides support for selection of sensors / systems to be analysed	SHALL	True
U82	Provides support for selection of information sources to be analysed	SHALL	True
U83	Provides support for data/sensor composition functionality	SHALL	False
U84	Able to provide historical knowledge about system deviations or problems	SHOULD	True
U85	Able to provide decision support for production line selection	SHOULD	False
U86	Able to increase visibility of the production process	SHALL	True
U87	Supports analysis for algorithm definition for boiling/temperature control functionality	SHALL	True
U88	Supports sensitivity analysis to noise	SHALL	False
U89	Supports main variation factor identification and robust strategy for minimising	SHOULD	False
U90	Supports computational resources estimation of machines	SHOULD	False
U91	Supports estimation of performance decrease for algorithm complexity reduction	SHOULD	False
U92	Supports process repeatability and stability characterisation	SHALL	False
U93	Supports Design of Experiments (DOE) and Analysis of Variance	SHOULD	True

Req. No.	Requirement	Overall Priority	Coverage
	(ANOVA) analysis		

7.2 PERFORMANCE REQUIREMENTS FROM INDUSTRIAL BUSINESS CASES

This table represents the coverage for the full prototype.

Req. No.	Requirement	Overall Priority	Coverage
U115	Does not negatively affect the usual production processes	SHALL	True
U116	Support for scalability in the size of cloud and computing resources	SHALL	True
U117	Support for horizontal scalability to many machines	SHALL	True
U118	Capable of real-time data ingestion (registering data)	SHALL	True
U119	Capable of batch processing of data (offline analysis)	SHALL	True
U120	Capable of real-time data processing	SHALL	True
U121	Capable of providing real-time reconfigurations / optimisations (subject to network throughput limits)	SHALL	True
U122	Able to analyse relevant data within a given timeframe	SHALL	True
U123	Capable of storing up to 5 TB/year/machine with resource recycling facilities	SHALL	True
U124	Provides support for Machine Learning (Supervised / Unsupervised / Anomaly Detection)	SHALL	True
U125	Able to achieve required precision on cooking process estimation / optimisations	SHALL	True

7.3 INTERFACE REQUIREMENTS FROM INDUSTRIAL BUSINESS CASES

This table represents the coverage for the full prototype.

Req. No.	Requirement	Overall Priority	Coverage
U130	Able to access data stored in a relational database	SHALL	True
U131	Able to receive and send data from/to a remote location	SHALL	True

8. CONCLUSIONS

This document has described the Final Specifications of the full prototype of Predictive Analytics Platform module to be developed in SAFIRE. The present deliverable is the second incremental outcome (referenced as D2.2.2 in the technical annex with name D2.5) of task *T2.2 Specification of Predictive Analytics Platform* and contains a) high level architectural design and b) specifications of functionalities regarding realisation of Predictive Analytics Platform Full prototype.

These specifications address requirements collected within WP1 regarding the real-time big data predictive analytics platform to tackle the development of a full prototype of the platform.

9. REFERENCES

[1]. H. Shvachko, H. Kuang, S. Radia und R. Chansler, "The Hadoop Distributed File System.," Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Sytems and Technologies (MSST), pp. pp 1-10, 2010.

[2]. Amazon AWS, [Online]. https://aws.amazon.com

[3]. Google Cloud, [Online]. https://cloud.google.com/

[4]. Microsoft Azure, [Online]. https://azure.microsoft.com/

[5]. IBM Watson, [Online]. http://www.ibm.com/big-data/us/en/big-data-and-analytics/watson-foundations.html

[6]. SAP, [Online]. http://www.sap.com/pc/analytics/predictive-analytics.html

[7]. GE Predix Platform., [Online]. https://www.gesoftware.com/predix

[8]. MongoDB, [Online]. https://www.mongodb.com

[9]. Cassandra, [Online]. http://cassandra.apache.org/

[10]. [Online]. Available: https://www.mongodb.com/blog/post/salamander-using-open-source-solutions-visualise-and-improve-banks-critical-internal

[11]. [Online]. Available: https://academy.datastax.com/resources/ing-groep-nv-exploiting-hotel-cassandra

[12]. J. Shute und e. al., "F1: A Distributed SQL Database That Scales." Proc. VLDB Endowment, pp. 6(11), 1068–1079, 2013.

[13]. L. Jay und e. al., "Recent advances and trends in predictive manufacturing systems in big data environment manufacturing Letters." 2013.

[14]. Flink, [Online]. http://flink.apache.org/

[15]. Spark, [Online]. https://spark.apache.org

[16]. Lambda Architecture, [Online]. http://lambda-architecture.net/stories/2016-07-16-IoT-analytics-platform

[17]. Kappa Architecture, [Online]. http://milinda.pathirage.org/kappa-architecture.com/

[18]. No Lambda Architecture, [Online]. http://es.slideshare.net/helenaedelson/nolambda-combining-streaming-adhoc-machine-learning-and-batch-analysis

[19]. V. Vavilapalli und e. al., „Apache Hadoop YARN: yet another resource negotiator.," In: Proceedings of 4th ACM Symposium on Cloud Computing (SoCC 2013)., 2013.

[20]. B. Hindman und e. al., "Mesos: A platform for fine-grained resource sharing in the data center." Technical Report UCB/EECS-2010-87, EECS Department, University of California, Berkeley, May 2010.

[21]. Data Centre Operating System. [Online] https://dcos.io/

[22]. "Predictive analytics Techniques". [Online] https://en.wikipedia.org/wiki/Predictive_analytics#Machine_learning_techniques

[23]. "A Tour of Machine Learning Algorithms". [Online] https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

[24]. "Logistic Regression",[Online] https://en.wikipedia.org/wiki/Logistic_regression

[25]. "Multinomial Logistic Regression", [Online] https://en.wikipedia.org/wiki/Multinomial_logistic_regression

[26]. "Decision Trees", [Online] https://en.wikipedia.org/wiki/Decision_tree

[27]. "Multivariate Adaptive Regression Splines", [Online] https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_splines

[28]. "Multivariate adaptive regression splines and neural network models for prediction of pile drivability", Geoscience Frontiers, Volume 7, Issue 1, January 2016, Pages 45-52. Elsevier.

[29]. "Artificial Neural Networks", [Online] https://en.wikipedia.org/wiki/Artificial_neural_network

[30]. "Predictive Analytics Tools", [Online] https://en.wikipedia.org/wiki/Predictive_analytics#Tools

[31]. "Top Predictive Analytics Software", [Online] https://www.predictiveanalyticstoday.com/top-predictive-analytics-

software/#toppredictiveanalyticssoftware
[32]. "Comparison of Deep Learning Software", [Online] https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software
[33]. "Predictive Analytics Tools", [Online] https://www.predictiveanalyticstoday.com/predictive-analytics-tools/
[34]. "Predictive Model Markup Language",[Online] http://dmg.org/pmml/v4-3/GeneralStructure.html
[35]. "Predictive Analytics Process", [Online] https://en.wikipedia.org/wiki/Predictive_analytics#Predictive_Analytics_Process