PROTEUS

Scalable online machine learning for predictive analytics and real-time interactive visualization

687691

D5.3 Architecture design for supporting incremental visual methods

Lead Author: TREE
With contributions from: DFKI, BU
Reviewer: DFKI

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M24 (30th November 2017)
Actual delivery date:	M24 (30th November 2017)
Version:	1.2
Total number of pages:	21
Keywords:	Big Data, data visualization, architecture

Abstract

Advanced visualization of data analytics in real-time, user experience and usability are still open issues in the context of Big Data that also concerns the Arcelor Mittal dataset and use cases in PROTEUS. To meet the requirements of data analysis tasks for real-time data, visualization platforms need to be able to draw high volumes of data in real time while providing user interaction with a very low latency. This deliverable describes the design and implementation of the 3-layer architecture used in PROTEUS. This architecture has been designed for supporting incremental visual methods that allow the analysts to explore both data-at-rest and data-in-motion with interaction times lower than 1 second.

Executive summary

Visualization of Big Data is still an open research question, especially when dealing with massive, unbounded streaming data at high frequencies that needs to be transformed, processed and merged with big volumes of batch data such as in the case of PROTEUS. Current visualization architectures add excessive complexity and costs to the development and maintenance of the system being developed, or they do not fully comply with the low-latency requirements presented in PROTEUS. High latency in a streaming visualization platform may cause delays that impede fluent interactions, hindering usability of the tools and making impossible to perform effective data analysis to make decisions on time.

The current deliverable analyses the existing Big Data architectures that are commonly used for processing both streaming and batch data, and later introduces the 3-layer architecture that has been designed for PROTEUS. This architecture has been designed and implemented during WP5 and attempts to solve the drawbacks of the Lambda and Kappa architectures, focusing on the requirements of real time interactive visualization. The 3-layer architecture in PROTEUS is composed of three main components:

- The *Data Collector* is in charge of iteratively getting new data from data sources (both static and streaming) and serving them properly and efficiently to the incremental analytics engine and the visualization layer. It also collects and store processed data and machine learning models, acting as a shared data layer for all the components of the architecture.
- The *Incremental Analytics Engine* processes data using classical analytics algorithms and also online machine learning algorithms that cover the gaps identified in T5.1. It outputs partial results which will be visualized by the third layer.
- The *Visualization Layer* enables views of the intermediate results at various iterations and allows the users to track and interact with those results in real time. It can combine the intermediate results with raw data, giving the user a *bigger picture perspective* for better understanding of the data.

To develop such architecture, we have developed the following contributions to the visual analytics ecosystem as part of WP5:

- Proteic.js: a general purpose data visualization library for the web, built for supporting high volumes of both streaming and static data.
- PROTEUS Visual Analytics Platform: a web application that provides a dashboard or user interface for performing incremental visual analytics on the PROTEUS data.

Document Information

IST Project Number	687691	**Acronym**	PROTEUS
Full Title	Scalable online machine learning for predictive analytics and real-time interactive visualization		
Project URL	http://www.proteus-bigdata.com/		
EU Project Officer	Martina EYDNER		

Deliverable	**Number**	D5.3	**Title**	Architecture design for supporting incremental visual methods
Work Package	**Number**	WP5	**Title**	Real time interactive visualization

Date of Delivery	**Contractual**	M24	**Actual**	M24
Status	version 1.2		final ■	
Nature	report ■ demonstrator □ other □			
Dissemination level	public ■ restricted □			

Authors (Partner)	Jorge Yagüe París (TREE)			
Responsible Author	**Name**	Jorge Yagüe París	**E-mail**	jorge.yague@treelogic.com
	Partner	TREE	**Phone**	

Abstract (for dissemination)	Advanced visualization of data analytics in real-time, user experience and usability are still open issues in the context of Big Data that also concerns the Arcelor Mittal dataset and use cases in PROTEUS. To meet the requirements of data analysis tasks for real-time data, visualization platforms need to be able to draw high volumes of data in real time while providing user interaction with a very low latency. This deliverable describes the design and implementation of the 3-layer architecture used in PROTEUS. These architectures have been designed for supporting incremental visual methods that allow the analysts to explore both data-at-rest and data-in-motion with low latency.
Keywords	Big Data, data visualization, architecture

Version Log			
Issue Date	**Rev. No.**	**Author**	**Change**
2017-10-04	0.0	Jorge Yagüe París	First draft
2017-10-25	0.1	Jorge Yagüe París	Complete draft
2017-10-31	0.2	Jorge Yagüe París	Internal revision (TREE)
2017-11-06	0.3	Jorge Yagüe París	Finalisation
2017-11-06	0.4	Marcos Sacristán Cepeda	Minor fixing and release for review
2017-11-07	0.5	DFKI	Reviewed version
2017-11-17	1.0	Jorge Yagüe París	Final version
2017-11-28	1.1	Bonaventura Del Monte	Reviewed version 1.0
2017-11-28	1.2	Jorge Yagüe París	Final version for submission

Table of Contents

List of figures and/or list of tables

Abbreviations

API: Application Programming Interface

HDFS: Hadoop Distributed File System

HTTP: HyperText Transfer Protocol

JSON: JavaScript Object Notation

PEACH: Proteus Elastic Cache

POJO: Plain Old Java Object

SOLMA: Scalable Online Machine Learning Algorithm

SQL: Structured Query Language

SPA: Single Page Application

STOMP: Simple Text Orientated Messaging Protocol

Definitions

Throbber: animated element in a graphical user interface to show that a computer program is performing an action in the background.

1 Conventional Big Data Visualisation Architectures

Advanced visualization of data analytics in real-time is still an open issue in the context of Big Data that especially challenges user experience and usability. How does Big Data change the nature of visual interaction? The interactivity requirement creates special challenges when it comes to Big Data. Interaction is a necessary condition for data analysis tasks, especially when using exploratory visual tools. However, most of the state-of-the-art tools or techniques do not properly accommodate Big Data [2].

As the visualization research and industry communities reorient their software to address upcoming challenges, they must successfully deal with diverse architectures, distributed systems, various heterogeneous data sources, massive parallelism, multiple input, output devices, streaming data, and interactivity. As previously described in D5.2, dealing with Big Data is characterised by four challenges, called the four V's of Big Data. These are the volume of observations per dataset, the variety of the data formats handled, the velocity at which data is generated as well as processed and being sure that the data we are dealing with is true and that we can extract value from it. Among these challenges, the original three V's identified by Gartner (volume, variety, velocity) [1] are of special relevance for the topic of data visualization from a technical perspective, since they disrupt the traditional way of performing data visualization. In general terms, this process can be summarised in two steps: loading data into memory and later applying a visualization algorithm to the data [2].

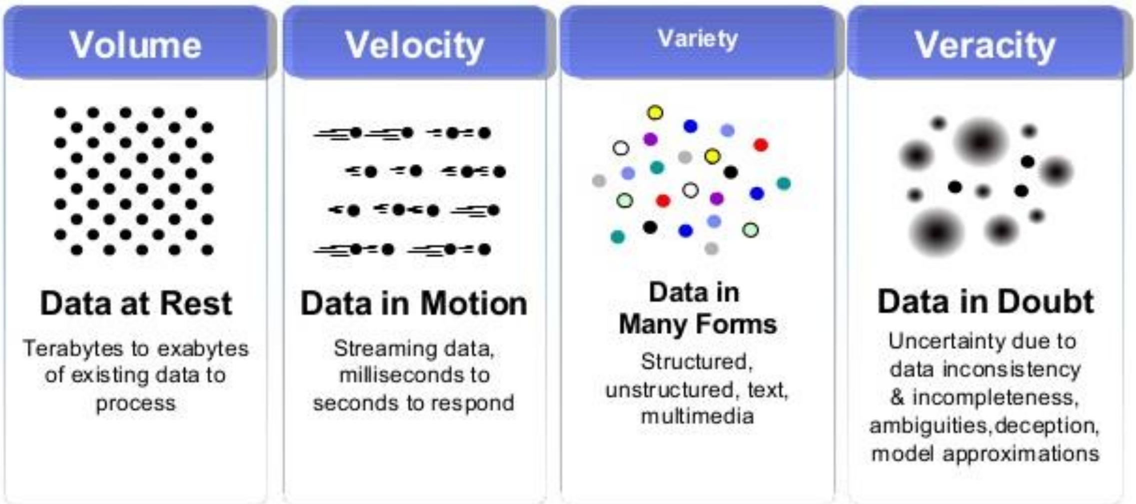

Figure 1: The four 'Vs' of Big Data

The first step is a non-issue with moderately sized datasets having enough memory, but becomes a deal breaker in big datasets that cannot fit the memory of the machine using for visualization. This problem becomes even harder with real-time streams or data, either bounded or unbounded, since the visualization will have to wait for all data to be loaded before being rendered, thus breaking the interactivity and delaying decision making processes.

The second step of data visualization is also impossible to apply to big data with traditional approaches. The sheer amount of data to process, and its real-time component often involves long processing times and latencies that again breaks interactivity and delays decisions. Moreover, the result of applying these algorithms is always rendering a representation of the data, and it may occur that the number of observations to be visualised is larger than the number of pixels available [3] in the screen, leading to deceitful visualizations.

Big data visualization can deal with these issues by providing processing scalability and visual scalability [2]. The first one refers to the ability to apply the visualization algorithms with enough performance to deal with the volume and velocity, so the latency does not disrupt the interactivity. The second property is the capability to effectively display large datasets, in terms of either the number or the dimension of individual data elements.

In the following paragraphs, we describe the most common architectures for streaming Big Data and its limitations for real-time interactive visualization.

1.1 Lambda Architecture

This architecture, defined by Nathan Marz [4], is based on the combination of the batch and stream processing in the same architecture, to achieve a generic, fault-tolerant, scalable architecture. It was design based on the experience of processing large amounts of data in a distributed environment at Twitter. Its objective was to being able of managing the trade-off between availability and consistency in the context of the CAP theorem, which states that a database cannot guarantee consistency, availability and partition-tolerance at the same time. This theorem still holds true in the Lambda Architecture, but with its separation between batch and streaming processing pipelines it enables levelling consistency and availability as desired, given that partition tolerance is required given the large amounts of data to be stored.

Figure 2 shows a high-level overview of the Lambda Architecture, which can be divided in three main components described in the following paragraphs.

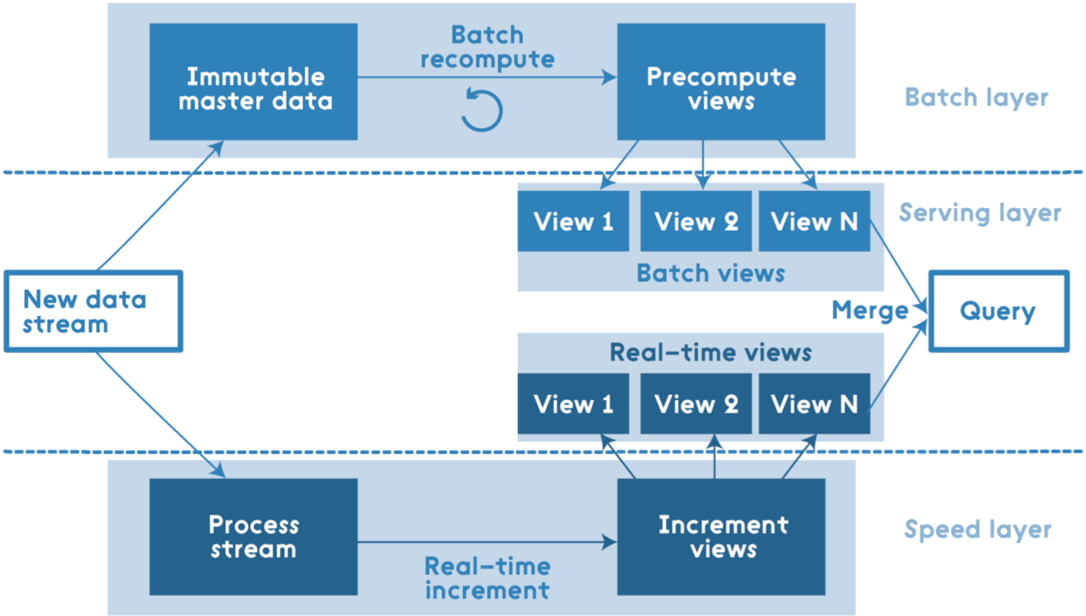

Figure 2: Overview of the Lambda Architecture

1.1.1 Batch layer

It manages the master dataset where all data in the system resides. This dataset is an immutable, append-only set of all the raw data entering the system from all the sources, either streaming or static. This layer is also in charge of pre-computing query functions on the batch data, called batch views.

1.1.2 Speed layer

This layer deals only with recent data coming from streams that require low latency processing and querying. It uses incremental algorithms for continuously deliver incremental views and compensate for the high latency of updates in the batch layer.

1.1.3 Serving layer

This layer indexes the batch views so that they can be queried with low latency and also performs the merge between the results from batch views and real-time views in case it is required by the use case. All data that enter the system are collected and dispatched by a message transport to both the batch layer and speed layers.

A common implementation of the lambda architecture involves using Kafka as the message transport that distributes incoming data to an HDFS cluster that manages the dataset, and a real-time processing engine such as Spark or Flink that generates the real-time views. Batch processing is done in the batch layer and may be performed by the same processing engine as in the speed layer, or by an entirely different one. In either case, the pipeline is independent in both layers, and thus, involves maintaining two separated code bases. The serving layer commonly involves merging the views from both processing layers into one or several low-latency databases to allow fast querying and visual analytics. A common approach for this layer is to use an Elasticsearch database that offers great integration through Kibana dashboards.

An implementation of the Lambda Architecture with two separated processing pipelines as described in this section covers a good set of the requirements for the industrial case presented in Proteus, however it entails some drawbacks for the visual analytics aspect of the project that we solve with the 3-layer architecture defined in the next section of this document. One of these drawbacks is the requirement of building two separated and complex distributed systems, for the batch and streaming pipelines, duplicating the costs of maintaining the code. This problem lead to the creation of the Kappa Architecture described in the following paragraphs.

1.2 Kappa Architecture

The Kappa Architecture was defined at LinkedIn [5] as an alternative to the Lambda Architecture, motivated by the complexity and high maintenance costs of the latter. Instead of using one system to handle real-time processing and another to handle reprocessing with batch streaming, the Kappa Architecture uses only a real-time processing engine that is also used to reprocess historical data by saving the stream in a message transport such as Kafka and replaying it at high speed when needed. The steps required for this are the following:

- Use Kafka or a similar message queue system that allows for multiple subscribers and to retain the full log of the data that may be reprocessed in a certain time.
- When the reprocessing is needed, start a second instance of the stream processing job that starts processing from the beginning of the retained data. The result of this is output to a new table.
- When the second job has caught up with the first, the queries are directed to the new table.
- The first job is stopped and the old table deleted.

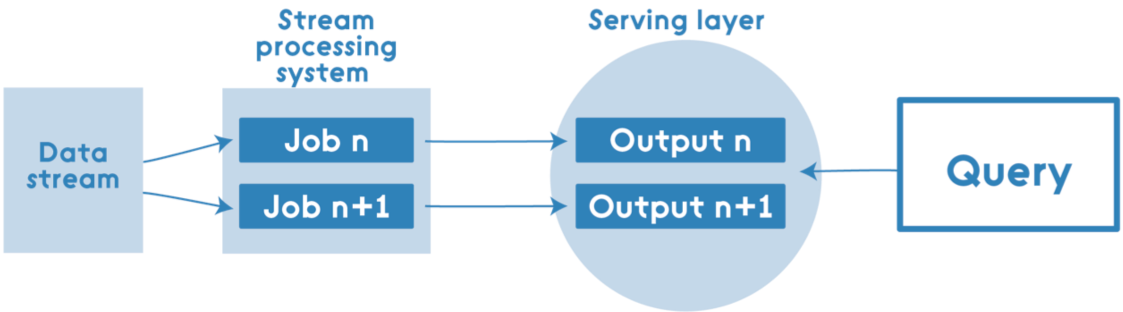

Figure 3: Overview of the Kappa Architecture

This architecture requires a lower number of systems to be developed or maintained compared to the Lambda Architecture, although it requires double the amount of storage space in the output database while the reprocessing job is running. Another drawback is that it also requires a database with support for high volumes of writes. Therefore, the Kappa Architecture is not more efficient in terms of performance than the Lambda, but it reduces the cost of development and maintenance. It is an architecture that should only be considered in *applications that do not require unbounded retention times or allow for efficient compaction* **¡Error! No se encuentra el origen de la referencia.**.

2 The 3-layer architecture

In the previous section, we have described the Lambda and Kappa architectures, often employed in Big Data systems, and the challenges of developing low-latency incremental visualizations for the PROTEUS industrial case using such architectures. To confront these challenges we have defined a Big Data architecture, mainly inspired in the Lambda architecture that is designed to support incremental visual methods. This incremental visual approach consists of three main layers: *Data Collector*, *Incremental Analytics Engine* and *Visualization Layer*.

- The *Data Collector* is in charge of iteratively getting new data from data sources (both static and streaming) and serving them properly and efficiently to the incremental analytics engine and the visualization layer. It also collects and store processed data and machine learning models, acting as a shared data layer for all the components of the architecture.
- The *Incremental Analytics Engine* processes data using classical analytics algorithms and also online machine learning algorithms that cover the gaps identified in T5.1. It outputs partial results, which will be visualized by the third layer.
- The *Visualization Layer* enables views of the intermediate results at various iterations, and allows the users to track and interact with those results in real time. It can combine the intermediate results with raw data to enhance insight into the data.

The following figure identifies how these layers have been implemented in the PROTEUS architecture. Components coloured in purple have been developed within the project:

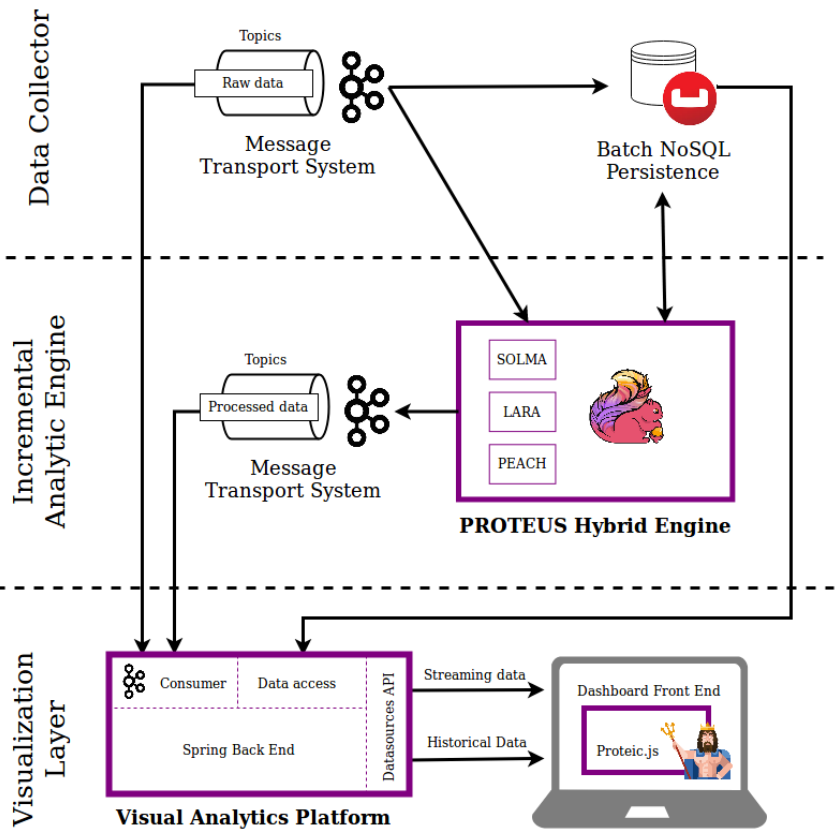

Figure 4: The 3-layers incremental visualisation architecture in PROTEUS

As shown in the previous figure, the Data Collector Layer collects and stores raw data entering the system and also processed data so it can be shared between the other two layers. This shared data layer allows the PROTEUS Hybrid Engine to perform operations with both streaming and batch data, while connection with the Visualization Layer enables visualizations with both kinds of data and mixing raw and process data.

This architecture has been designed to support the requirements of real-time interactive visualization of incremental results in PROTEUS with a very low latency. This has been achieved following the guidelines specified in deliverable D5.2, which describes the technologies, techniques, data formats and protocols used to develop the visualization layer.

The following sections provide further details about each of the elements in the architecture and its implementation to support real-time interactive visualisations in PROTEUS. The individual components used and developed for implementation of the visualization layer has been described more thoroughly, since they play the most prominent role to meet the visualization requirements. Information about the deployment of the architecture in a production environment and how they are integrated for the prototypes can be found in deliverable D2.9.

2.1 Data collector

This layer mainly deals with the challenges associated with the velocity and volume properties of the data. It involves the connection to sources of raw data, so it can be collected and distributed to other components of the architecture. This is achieved by a message transport technology in charge of writing data to distributed topics in the cluster where other components will read and treat them accordingly. The data collector is also in charge of gathering data emitted from the Visual Analytics Engine, such as intermediate results and machine learning models, so it can be aggregated with raw data in the visualization layer. This enables better insight for operators in the industrial setting, who may find useful correlations in the manufacturing process as they can visualize raw data from the coils being produced with the analytics results about the whole dataset of past coils.

All data collected in the PROTEUS is also integrated by this layer into a master dataset stored in a NoSQL database to enable querying from the visualization layer. This database also stores the batch datasets used to generate the machine learning models at the incremental analytics engine.

As the entry point of all data in the system, technologies chosen for this layer must be as fast and efficient as possible to avoid bottlenecks that may halt the rest of the pipeline or cause data loss. In this case, we have implemented this layer using the technologies described in the following sections.

2.1.1 Message transport system

The message queue has been implemented using Apache Kafka but any other message queue system may be chosen to perform this role, for example, Apache Flume or RabbitMQ. We chose Kafka over others because it is an open source technology that is becoming an industry standard for message processing [8]. Also, Kafka is a general-purpose solution, while Flume is more oriented towards the Hadoop ecosystem and works with a push model in which data sinks must keep with the pace of events, causing bottlenecks at event peaks [9]. RabbitMQ is also a proven and well-known technology, but it does not reach the performance of Kafka **¡Error! No se encuentra el origen de la referencia.**.

The streams of data received at the message queue are written to the following distributed topics according to the coil production data flow (for further details see deliverable D2.8, section 1.2, pp. 11):

- **Real-time**: manages 1-dimensional and 2-dimensional time-series data produced and available in real time during the coil production.
- **HSM**: manages all HSM data, produced as aggregated information at the end of each coil production and available only once the coil production has finalised.
- **Flatness**: manages the flatness data variables, produced as measures of the flatness of the resulting coil, and available only after a certain delay after the coil production has finalised.

The Incremental Analytics Engine also writes the incremental result of each algorithm to different topics in the queue system. All the topics will later be read by two consumers developed purposefully for the kind of data handled in the project. The first one treats the unbounded stream of data for the incremental analytics engine, while the second one writes the stream to a distributed NoSQL database where it is merged with all the historical data.

2.1.2 NoSQL database

PROTEUS provides innovative solutions based on incremental visual methods that allow end-users to explore data in real time efficiently to make well-informed decisions. This is based on a hybrid approach, i.e.

not only for data-in-motion but also for both data-at-rest. For the historical or batch data, we have chosen Couchbase Server as the technology for the storage of historical records of raw data and results from the incremental analytics engine in a reliable way. This database also stores batch data from the HSM dataset. The Data Collector is in charge of iteratively handling new data from data sources (both static and streaming), managing updates and queries towards serving them properly and efficiently for incremental analytics operations.

2.2 Incremental analytics engine

The next step in the data pipeline is to process the data so it can be cleaned and modelled to the visualization layer requirements. It is also in charge of performing analytic operations that will be visualized by users, providing them with a deeper understanding of the data. In order to deliver the results in real time while enabling low-latency user interaction, this engine must use incremental approaches to the processing that generate partial results as new data enters the platform. This kind of processing is achieved by applying online scalable algorithms over distributed stream processing technologies that follow hybrid architectures.

Instead of using two parallel processing engines like traditional Lambda Architectures, one for processing batch data and another one for streaming data, we have developed a hybrid processing engine based on Apache Flink to process both kinds of datasets. The PROTEUS Hybrid Engine is a fork of Apache Flink that features an extended support to operations that mix streaming and batch processing. This is achieved through side inputs for stream operators, i.e., an operator processes records coming from the stream as well as preloaded historical data retrieved from the NoSQL database. As a result, side inputs enable hybrid processing within a stream operator.

Results generated at this layer are directed again to the Data Collector layer, implemented as a Kafka cluster, where results from each algorithm will be written in different topics. The visualization layer will read from these topics incrementally to generate real-time visualizations assuring low-latency user interaction with partial results. Besides, these results will also be stored in the NoSQL database as batch data in order to support later visualization of historical results.

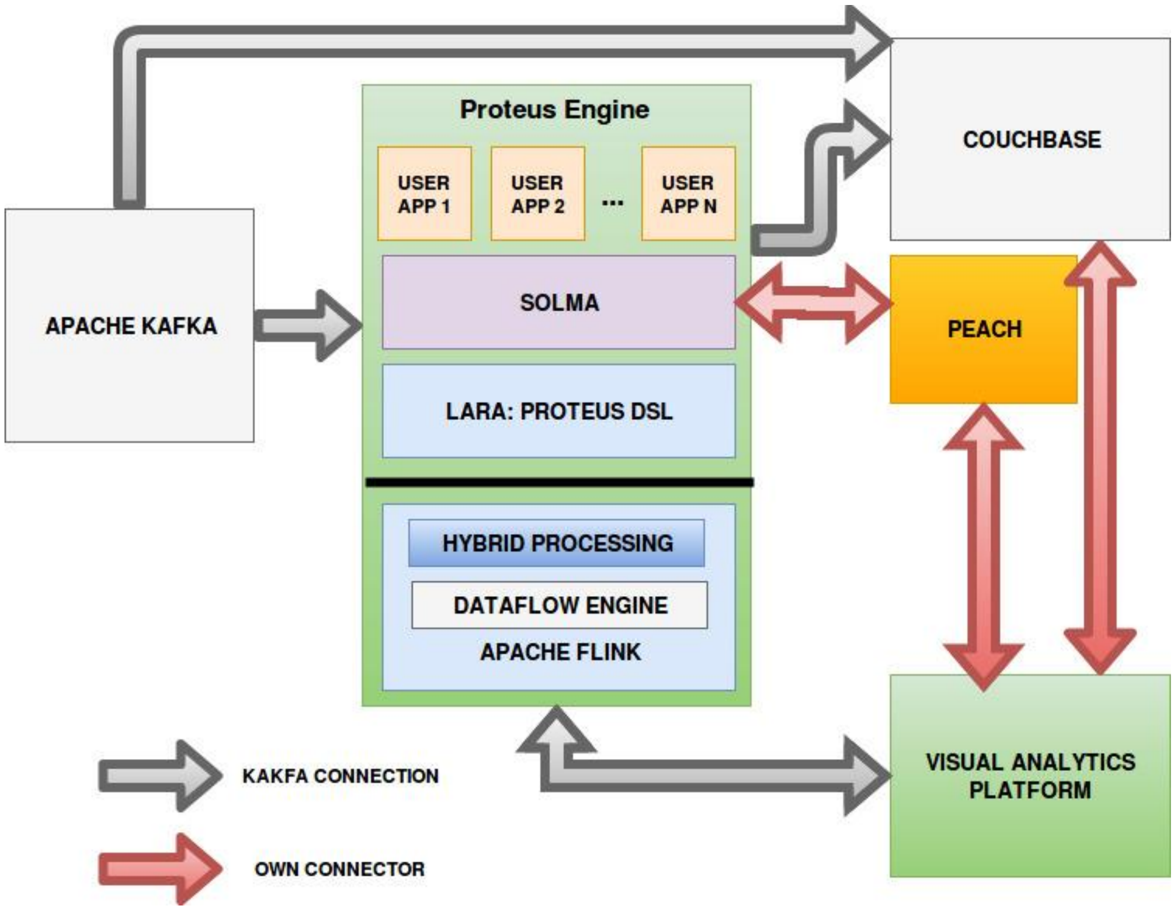

Figure 5: Architecture of the Incremental Hybrid Engine in PROTEUS

The architecture of this layer, depicted in Figure 5, has been implemented using the following technologies developed in PROTEUS as the core contributions (see the WP3 and WP4 deliverables for the details):

- **PROTEUS Hybrid Engine:** a fork of Apache Flink. It features an extended support to operations that mix streaming and batch processing. This is achieved through side inputs for stream operators, i.e., an operator processes records coming from the stream as well as preloaded historical data. As a result, side inputs enable hybrid processing within a stream operator.
- **LARA:** Lara is an engine-independent declarative language that allows programmers to author complete end-to-end data analysis programs that are automatically parallelized. The main observation behind Lara is that data analytics first perform some pre-processing, which naturally fits the dataflow paradigm, and then they perform some machine learning tasks, which are expressed as linear algebra operations with iterations. Therefore, Lara aims to provide a common environment that allows to define pipelines that mixes relational and linear algebra
- **PEACH:** Peach (Proteus Elastic Cache) is a distributed key-value cache that can be used both inside and outside of the Proteus scope. The cache aims to provide low latency responses on a distributed elastic deployment with fault-tolerance capabilities. As a generic design, the cache could be integrated within Apache Flink to speedup computing processes.
- **SOLMA:** scalable online machine learning library (including classification, clustering, regression, ensemble algorithms, graph oriented algorithms, linear algebra operations, and anomaly and novelty detection) implemented on top of Apache Flink using the hybrid processing capabilities.

Deliverable D2.9 describes in section 2.2.3 each of these contributions and their deployment in production with more detail. Further information about each contribution can be found in the following deliverables:

- D4.3: description of the novel online learning algorithms implemented in SOLMA.
- D3.7: design and implementation of the declarative language LARA.
- D3.2: design and implementation of the hybrid engine.
- D3.5: design and implementation of the state management system PEACH.

2.3 Visualization layer

Big data visualizations require a set of features that are uncommon in traditional visualization tools. For instance, they should be able of drawing high volumes of both processed and raw data in real time, and provide querying capabilities so users can get an overview of the dataset while being able to explore the data in detail. Interface and presentation technologies provide users with capabilities to visualize and sense data from mobile devices to full immersive systems. Interaction technologies let users change their visualization needs and adapt it according to their context.

As mentioned in the previous section the Data Collector Layer aggregates all data in PROTEUS, including raw data (streaming or batch) and results from the Incremental Analytics Engine Layer. This means that the visualization platform is able to access both raw data and processed data in real time and it can generate rich visualizations combining both data sources thanks to the low latency of the architecture and technologies used. Connection to the NoSQL database allows querying of historical and batch data, allowing for comparison of ongoing streaming visualization with previous results and fast decision making for operators in the industrial scenario.

This layer is designed as a client-server web application that is connected to topics in the message queue of the Data Collector layer, and also to the historical data in the NoSQL database. The client or front end of this application is a dashboard where operators can monitor the coil production process and query historical data, while the server or back end is in charge of providing authentication, authorisation, and access control to the platform and to provide services for querying and connecting the dashboard to the Data Collector layer. Performance of the protocols and technologies chosen for this layer are crucial, since it lives outside of the cluster infrastructure and the front end will be run as a JavaScript application on commodity hardware like an ordinary laptop. WP2 will define KPIs to measure such performance.

To meet these criteria, we have implemented such architecture by developing the Visual Analytics Platform that integrates the following contributions in PROTEUS:

- The *back end* of the visual analytics platform is built as a Java EE application on top of the Spring Framework and can be deployed in an Apache Tomcat Server.
- *PROTEUS Dashboard* is the user interface for the platform. It is a cross-platform application that can run either as a web application or a desktop application by means of the Electron framework. The application allows the user to access a dashboard through a login screen where he can create and customize visualizations of high-dimensional and streaming datasets.
- The *Datasources API* is part of the back end in the Visual Analytics Platform. It allows bidirectional communication of the visualization at the client side of the platform with the server side through various protocols, namely WebSocket, HTTP or HTTP/2.
- *Proteic.js* is the data visualization library for the web created for the interactive visualization of the Proteus data and the results of the incremental analytics engine. It is integrated in the Proteus Dashboard to display data retrieved from the back-end of the visual analytics platform and has been designed to support high volumes of data with optimal performance.

In the following sections we describe the previous contributions in more detail. Information about the deployment in production of these technologies can be found in deliverable D2.9, and further details about the technologies in the Visual Analytics Platform, and the guidelines followed for its design are included in deliverable D5.2.

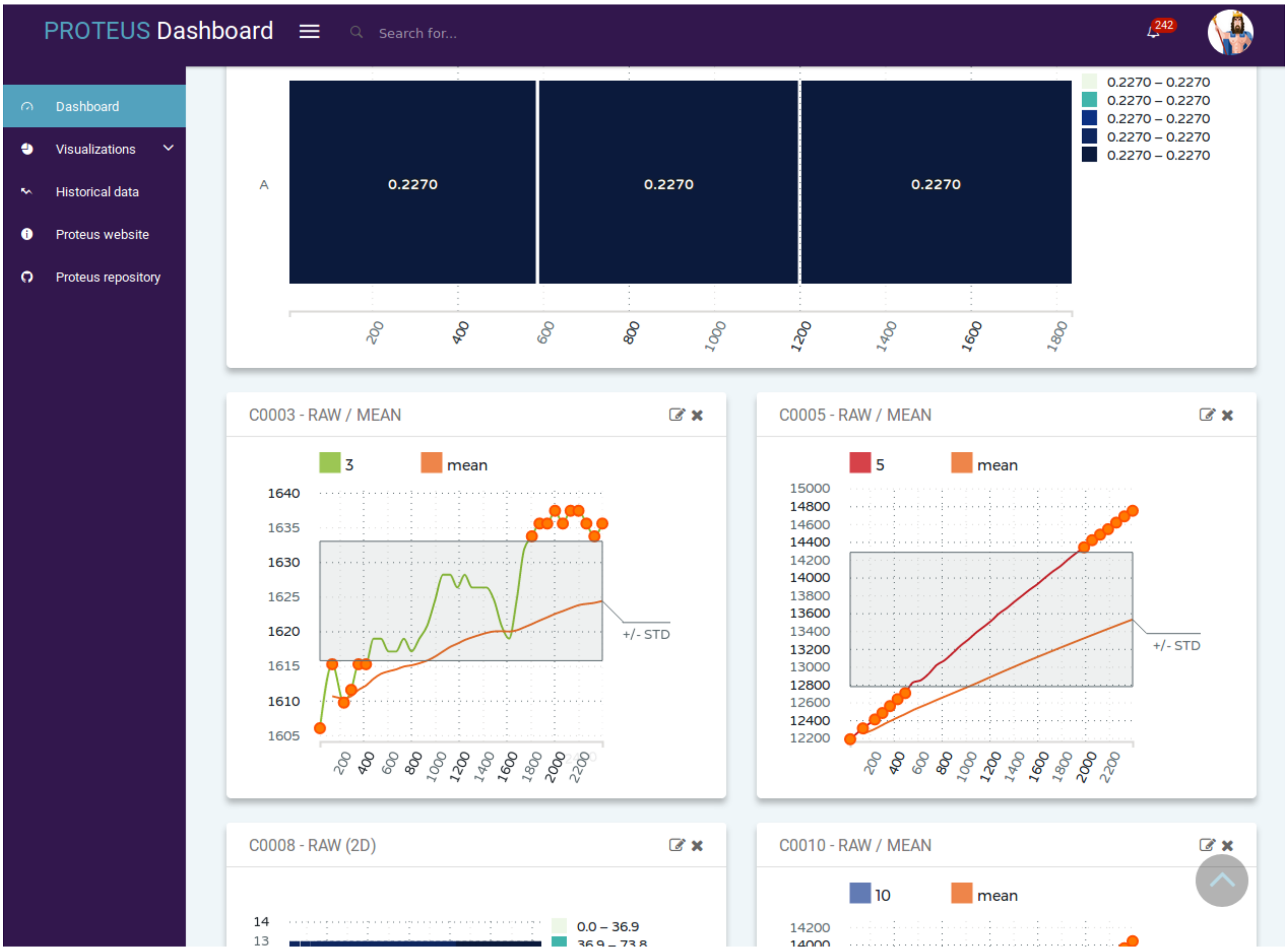

Figure 6: User interface of the Visual Analytics Platform

2.3.1 Visual Analytics Platform: back end

The backend for the Visual Analytics Platform is implemented as a Java EE application, built on top of the Spring Framework and deployed in an Apache Tomcat Server. It provides security features via Spring Security, querying for the historical database using Spring Data and N1QL code in order to retrieve data from CouchBase server, and a Kafka consumer to collect the messages from the processing engine.

2.3.1.1 Security

The Visual Analytics Platform incorporates the Spring Security framework to preserve confidentiality of the PROTEUS dataset. This framework provides customizable authentication, authorization, access control, and protection against attacks like session fixation, click-jacking, cross site request forgery, etc.

2.3.1.2 Database connection

The visual analytics platform is able to query historical data stored in the CouchBase server using N1QL (declarative query language that extends SQL for JSON documents), but it could be connected to other kinds of databases for other use cases, both relational and non-relational. To enable these queries, the backend employs Spring Data with the Couchbase connector [12], which provides a common programming model based on POJO for the data access layer to abstract the programming model from the underlying data store [13].

2.3.1.3 Kafka Consumer

All data in the cluster infrastructure is written into Kafka topics, including raw data (real-time, HSM, flatness) and process data (results from the algorithms in the Incremental Analytics Engine). These topics are later read by a Kafka consumer in the visual analytics platform, and its data can be accessed from the

PROTEUS Dashboard using the Datasources API through various endpoints for different communication protocols, for instance STOMP and WebSocket.

2.3.2 Visual Analytics Platform: PROTEUS Dashboard

Proteus Dashboard is the user interface for the visual analytics platform. It is a cross-platform application that can run either as a web application or a desktop application through the Electron framework. The application allows the user to access a dashboard through a login screen where he can create and customize Proteic visualizations that can then be connected to a set of data sources in the back-end of the visual analytics platform. The dashboard is based on the ng2-admin framework [14], which is built with the following cutting edge, open source technologies:

- Angular: open-source web application framework based on TypeScript for developing front-end web applications, usually in the form of Single Page Applications (SPA). This is used in PROTEUS for building the web interface of the Visual Analytics Platform. The interface is designed as a set of reusable modules that allow creating, editing and deleting visualizations contained in cards.
- Bootstrap 4: open-source CSS framework for designing websites and web applications. It brings responsive and reusable front-end components that ease the design process of the platform.
- Electron: open-source framework for the development of desktop applications using front and back end components originally developed for web applications. This allows the Visual Analytics Platform to be used as a standalone desktop application.

2.3.3 Datasources API

The connection between client and server in the visualization layer is achieved by means of a *Datasources APIs*, which enable bidirectional communication of the visualization at the client side of the platform with the server side. This API can be used from the client-side to list all the available data sources, either batch or streaming, and receive data through various protocols, for example, WebSocket, HTTP or HTTP/2. Thanks to WebSocket connections, we aim to achieve low-latency visualization of the streaming datasets since it provides asynchronous and bidirectional communication between client and server, allowing data to be drawn incrementally as messages arrive to the visualization [11]. Performance of this technique will be evaluated as part of WP2.

2.3.4 Proteic.js

Proteic is the data visualization library for the web created for the interactive visualization of the Proteus data and the results of the incremental analytics engine. It is integrated in the Proteus Dashboard to display data retrieved from the back-end of the visual analytics platform. The library features a convenient declarative API for the creation of multiple kinds of visualizations suited for high volumes of both streaming and static data. Its API enables the retrieval of data over the various communication protocols available in the Datasources API of the back-end component.

This component is built and distributed as a standalone open source library that can be used in any data visualization scenario for the web. It is implemented in the TypeScript language, and follows the latest standards and good practices in web development to achieve high performance, compatibility and ease of development. The design process of the library accounts for usability and accessibility of the various visualizations. It also follows design guidelines like responsive web design and offering alternative colour palettes for colour blind-users.

The Proteic library brings the following features to the visual analytics platform:

- Time series visualizations using dynamic and interactive line charts, stream graphs, and area charts. These charts include a set of tools for time series analysis such as dynamic thresholds, outlier alerts based on thresholds, dynamic error or confidence bands and pause / resume capabilities.
- User interaction features like brushing, zoom, highlighting, pause / resume; following the Shneiderman’s mantra: *Overview first, zoom and filter, and details on demand [7]*.

- Usability and accessibility features, such as interactive legends, annotations, accessible colour palettes, customizable themes, error warnings and a throbber that is shown while the visualization is waiting for data to arrive form data sources.
- High-dimensional geometry and multivariate data visualization by means of parallel coordinates visualization, including features for visualization of missing data.
- Innovative real-time visualizations, namely real-time heatmaps and swimlane charts.

3 Conclusions

Visualization of very large data sets is still an open research question, especially when dealing with streaming data at high frequencies that needs to be transformed, processed and merged with big volumes of batch data like it is the case in PROTEUS. These operations on streaming data cause delays that impede fluent interactions, hindering usability of the tools and making impossible to perform effective data analysis. Existing big data architectures, like the Lambda Architecture and Kappa Architecture, which adds excessive complexity in the first case, or do not fully meet the requirements in the PROTEUS industrial use case in the second case, since it requires processing of both data-at-motion and data-at-rest.

This deliverable describes the 3-layer architecture that has been in designed for PROTEUS as part of WP5, its implementation and several contributions developed to support such architecture. Besides the big data architecture itself, other contributions in WP5 include:

- Proteic.js, a general purpose data visualization library for the web, built for supporting high volumes of both streaming and static data.
- PROTEUS Visual Analytics Platform: a web application that provides a dashboard, or user interface for performing incremental visual analytics on the PROTEUS data.

References

[1] D. Laney, "META Delta," Appl. Deliv. Strateg., vol. 949, no. February 2001, p. 4, 2001.

[2] P. Godfrey, J. Gryz, P. Lasek, and Y. University, "Interactive Visualization of Large Data Sets," IEEE Trans. Knowl. Data Eng., vol. 11, no. 12, pp. 1041–4347, 2015.

[3] B. Shneiderman, "Extreme visualization," Proc. 2008 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '08, pp. 3–12, 2008.

[4] N. Marz, "How to beat the CAP theorem," 2011. [Online]. Available: http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html.

[5] J. Kreps, "Questioning the Lambda Architecture," 2014. [Online]. Available: https://www.oreilly.com/ideas/questioning-the-lambda-architecture.

[6] W. Wingerath, F. Gessert, S. Friedrich, and N. Ritter, "Real-time stream processing for Big Data," it - Inf. Technol., vol. 58, no. 4, pp. 186–194, 2016.

[7] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," Proc. 1996 IEEE Symp. Vis. Lang., pp. 336–343, 1996.

[8] "The Continued Rise of Apache Kafka – Charting Stacks." 2017. [Online]. Available: http://redmonk.com/fryan/2017/05/07/the-continued-rise-of-apache-kafka/.

[9] "Flume or Kafka for Real-Time Event Processing | LinkedIn," 2015. [Online]. Available: https://www.linkedin.com/pulse/flume-kafka-real-time-event-processing-lan-jiang/.

[10] "Understanding When to use RabbitMQ or Apache Kafka." [Online]. Available: https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka. [Accessed: 24-Nov-2017].

[11] C. Marion and J. Jomier, "Real-time collaborative scientific WebGL visualization with WebSocket," in Proceedings of the 17th International Conference on 3D Web Technology - Web3D '12, 2012, p. 47.

[12] "Spring Data Couchbase." [Online]. Available: https://projects.spring.io/spring-data-couchbase/

[13] C. Walls, Spring in Action. 2014.

[14] "ngx-admin" [Online]. Available: https://github.com/akveo/ngx-admin