VELaSSCo

***V**isual Analysis for **E**xtremely **La**rge-**S**cale*
***S**cientific **Co**mputing*

D2.2 – Specification of Big Data Architecture

Version 3.0

Deliverable Information

Grant Agreement no	619439
Web Site	http://www.velassco.eu/
Related WP & Task:	WP 2, D2.2, T2.2
Due date	September 30, 2014
Dissemination Level	
Nature	
Author/s	Ivan Martinez, Tomas Pariente,
Contributors	Alvaro Janda, Andreas Dietrich, Benoit Lange, Olav Liestol, Miguel A. Pasenau de Riera, Jochen Jaenisch

Approvals

	Name	Institution	Date	OK
Author	Ivan Martinez Tomas Pariente	**ATOS** **ATOS**	30/09/2013	
Task Leader	Ivan Martinez	**ATOS**		
WP Leader	Toàn Nguyên	**INRIA**		
Coordinator				

Change Log

Version	Description of Change
Version 0.1	First draft version of the document
Version 0.2	Draft including Olav and Benoit contributions
Version 0.3	Draft including Andreas contribution for visualization chapter
Version 1.0	Semi-final version including Alvaro contribution in HPC section and DEM data.
Version 2.0	Release version including contributions for visualization engines section and FEM data of Miguel A. Pasenau and contributions in general section of ATOS.
Version 3.0	Final Releases including review updates

Table of Contents

1. Introduction

Based on the literature surveyed in task T2.1, this document specifies in the scope of task T2.2 the general system architecture from a big data perspective.

A main focus in the distributed development of a system is the specification of the architecture and the integration processes to support the integration of several components into a single system. It is also relevant to propose a modular configuration of the system to fit with the different use cases defined in the project.

Architecture and integration standards are separated into different aspects of development. The architecture describes the structure of a system while the integration focuses on the process of fulfilling this architecture. The architectural standards focus on the correlation of all services and components and the "integration" of external legacy systems into the VELaSSCo system while the integration aspects focus on the set of tools, procedure and interfaces to support the communication and computation of VELaSSCo. Therefore the ICT architectural and integration aspects focuses on two main processes:

- the definition of an integrated architecture supporting the conceptual structure of the VELaSSCo system and,
- the set of processes and tools to support the distributed development and further combination into one system.

In the case of VELaSSCo the architecture proposed should be able capable of handling efficiently the various data produced by the software tools, taking into account the various execution overheads of each particular processing tool, while ensuring a consistent and effective use of up-to-date data by the user, as they are made available.

Therefore, this document reports on the specification of the big data architecture most suitable for VELaSSCo simulation data environment, taking into account the deliverable D2.1 regarding State-of-the-Art of Big Data approaches, such as the Lambda architecture, which finally has been selected to instantiate the VELaSSCo Architecture.

2. Concepts of VELaSSCo Architecture

The goal of the current section is to introduce the core concept of VELaSSCo Architecture which is the Lambda Architecture Approach next to some relevant concepts associated to VELaSSCo project from a conceptual and technical perspective.

The Architecture of VELaSSCo platform will be based on Lambda Architecture (LA) proposed by Marz [1]. According Marz "*LA aims to satisfy the needs for a robust system that is fault-tolerant, both against hardware failures and human mistakes, being able to serve a wide range of workloads and use cases, and in which low-latency reads and updates are required. The resulting system should be linearly scalable, and it should scale out rather than up*".

Figure 1 shows the LA from a high-level perspective:

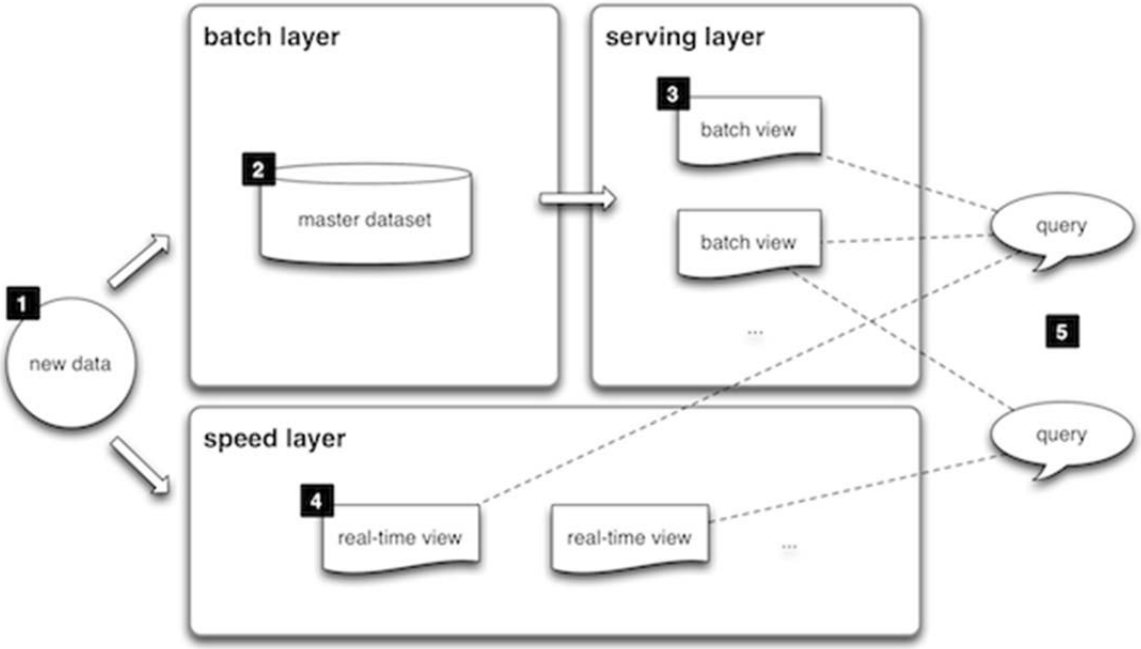


Figure 1: Overview of the Lambda Architecture [1].

Essentially, the Lambda Architecture comprises the following components, processes, and responsibilities:

1. Data Sources: Are the sources of all data to be made available to the solution. The Big Data solution must have the ability to capture any data source. To this solution usually have connectors or adapters to receive, capture data already stored in external databases, text files or applications or circulating information generated by sensors, social networks, mobile devices, etc. In this way the data is they make available the following layers of either architecture for storage and subsequent processing (batch layer) or real time processing (speed layer).
2. Batch layer: This layer is responsible for storing the data in what is called Master Dataset and pre-processing of information for generating pre-computed views.

Hadoop's *HDFS* is typically used to store the master dataset and perform the computation of the batch views using *MapReduce*.

3. Serving layer: This layer must be able to combine the results of the pre-processed data in Batch processing layer with the data exposed through the Speed layer. It must also be able to index and display the data so that they can be consulted.
4. Speed layer: Due to the high degree of latency batch processing layer, and since there will generally be some data need to be exposed to a faster mode, a layer of processing real-time information is required. To this layer is called Speed layer and is responsible for processing real-time data stream that needs to be exposed immediately. As the data stream is accessible information is automatically analyzed and pre-computed, generating views that expose these data for consultation.

   Precisely because the functionality is intended to cover, the data exposed through the Speed layer are transient and are available only temporarily; while not precisely be through batch processing layer.
5. Queries: Lambda architecture answered any kind of incoming query by merging results from batch views and real-time views.

2.1. VELaSSCo Lambda Architecture Instantiation

Subsequently, the Lambda architecture instantiation proposed for VELaSSCo from a service point of view is presented in Figure 2:

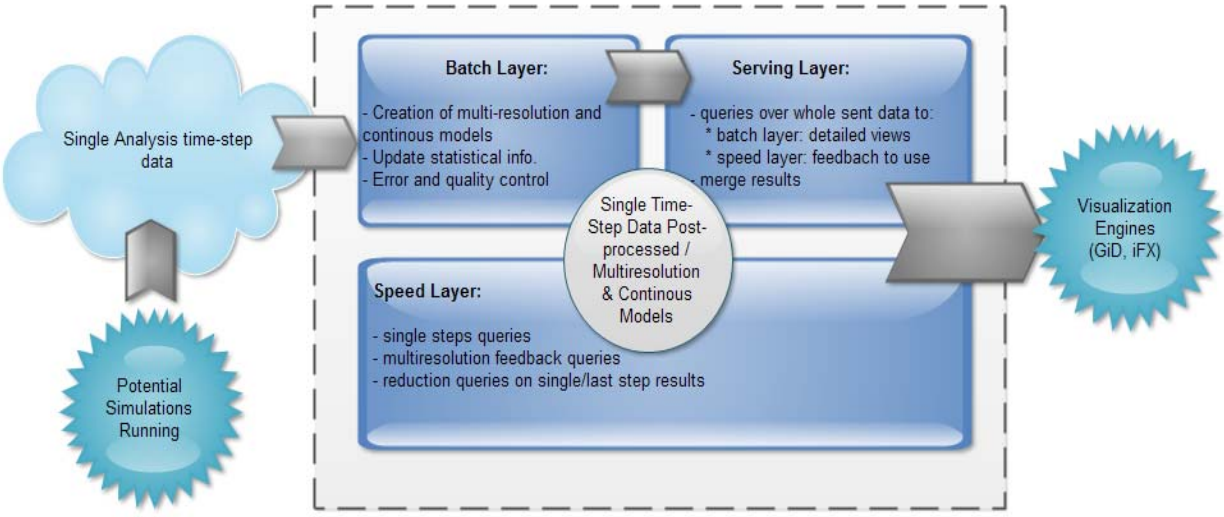


Figure 2: VELaSSCo LA instantiation

To handle, store, and visualize the amount of data that will be generated from the future engineering simulations, it is imperative to use large scale distributed computing

infrastructure such as used in Big Data infrastructure. The architecture diagram of the big data infrastructure in the context of engineering application is shown in the Figure 2. It primarily consists of three layers: batch, speed, and service layers.

The roles and functions of these layers are summarized as follows:

1. The engineering data from the simulation is constantly sent to the HPC clusters with the help of the petabyte sized data interface, thus creating a constant growing master dataset. With this constantly growing dataset, the batch layer starts.
2. The batch layer pre-computes query functions (based on the domain knowledge) on subsets of data of the master dataset, and then store the partial results in flat files, called batch views. The views are created in order to achieve low response time in implementing a send result approach (for more details see the visualization engines section).
3. In order to correctly, and quickly send the target files based on the users queries, these files are indexed by the database that is specialized for engineering simulations. The service layer holds the database.
4. As soon as the request for the visualization enters the service layer, the database fetches appropriate computed partial results available through the indexing mechanisms. These results are either sent to the clients, where the GPUs available on the client side can make visualizations out of the sent results by leveraging the newest OpenGL features , or streamed via the rendering nodes of the HPC clusters to the potential clients
5. While the batch layer performs its pre-computation, if any new data arrives, then they are sent to the speed layer. The speed layer runs incremental algorithms and helps in visualizing only the real-time (recent) data.
6. Visualization queries are resolved by getting results from both the speed layer and batch layer.

3. Specification of VELaSSCo Architecture

This section presents specifications of the architecture for the VELaSSCo platform. This platform is designed to support engineering data used and produced by simulations. As stated in the literature, data produce by scientific communities grows in an exponential way, and at the same time, the analysis algorithms are more and more complex and take more time. Thus to enable more efficient computations, it is necessary to use large computing infrastructures instead of a single node. Conceptually, most simulations produce raw data, composed by lists of integers, floats and doubles. An effort is required to manage this information and produce useful visualizations. Big data frameworks include a building block in charge of data analytics conceived as a sophisticated process that extracts pieces of information from the data, and apply specific computations to extract the desired features. A global overview of the platform is presented in Figure 3.

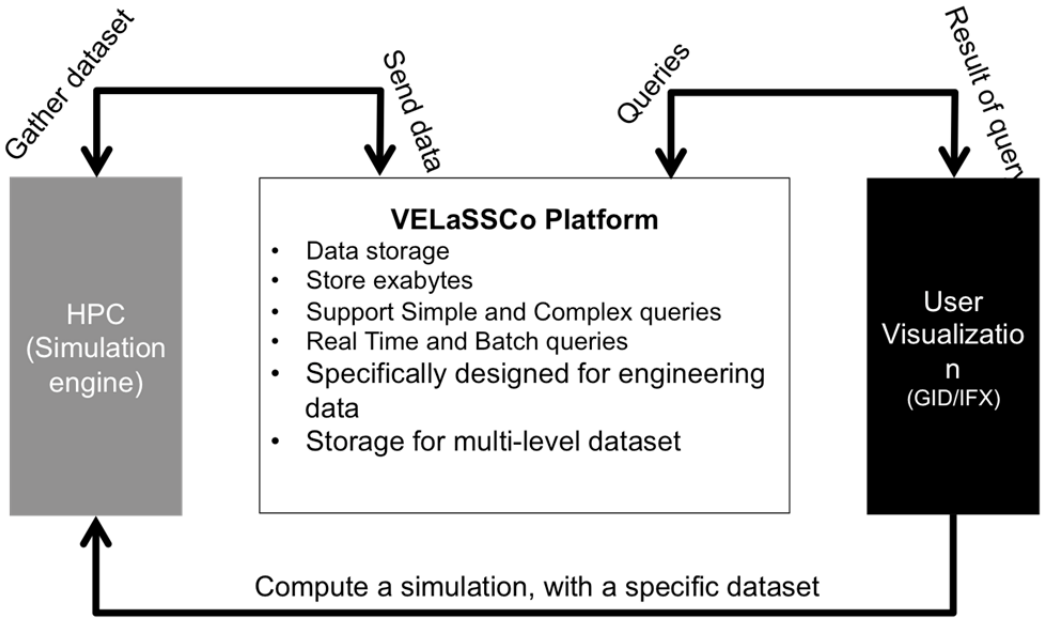


Figure 3.VELaSSCo infrastructure.

With the VELaSSCo project, our consortium tries to propose an answer to different requirements of scientific visualization and big data production.

The first requirement of the architecture is to be effective on most scientific hardware. As stated in Deliverable 2.1, the traditional architecture of Big Data is composed by some commodity nodes (with simple hardware capabilities, low speed network (typical network connection) and local efficient storage). While for scientific usage, compute facilities are composed by a small set of high performance nodes, managed by a resource scheduler. In a commodity center, the number of nodes is the most important aspect as BigData performance scales linearly with the number of nodes, while in HPC centres the focus lies in a high speed network which communicates several thousand compute nodes but with a 'centralized' viewed storage, i.e. with almost no local storage which is scarcely used. The targeted architecture depends on the available infrastructures, thus it is necessary to find an

efficient solution to make compatible a BigData framework on a HPC cluster. The software stack used for this project has also to support most of the requirements of big data. That means: how to dispatch efficiently the computations among nodes, how to use the advanced features of modern and old CPUs, and how to deal with the fault tolerance and failovers. Small changes in the HPC infrastructure may be needed for a successful deployment for an efficient adapted BiGData solution, such as the incorporation and usage of local storage on some of the computation nodes of an HPC cluster. The compute intensive simulations running on an HPC cluster will send the data to our solution, deployed in the same cluster, and the user can access and analyze this data in an efficient way through a specific smart and user-friendly interface. The solution needed for this project may also be transportable to a cloud oriented infrastructure: the users will be unaware on how and where the computation is performed, but accessible through the same user-friendly interface.

For the second (Big Data) requirements, our architecture has to deal with data location and storage, see Figure 4. In this project, a specific HPC cluster will produce simulation data. To be able to perform large simulations, engineers decomposes the problem data into as many sub-domains, also called partitions, as compute nodes are used to solve the simulation. The each compute node outputs the results of the simulation for its sub-domain. For this purpose, Flume seems to be the most suitable tool to ingest the several partial results into the BiGData system. With this strategy, maybe it is necessary to use an aggregating solution to inject data into the VELaSSCo platform. But the BigData framework also splits a data set among several data-nodes for efficiency and fault-tolerance purposes. May be is it possible to take profit of the smart partition the simulation engineer has done and use it to distribute the simulation results among the data-nodes. In a high performance simulation environment, each compute node deals with a sub-domain of the dataset, and produces its own result file (one for each time-step). Flume grabs these files from the compute nodes and stores them into the VELaSSCo platform using agents. Flume can be in charge of the aggregation of the same time-step and to store it. For the storage part, we plan to use an existing layer of the Hadoop distribution. Hadoop provides an abstract class for file system storage; this enables a high extensibility of the storage system. The Flume agents will put the data into the storage system through this abstract file layer. Flume supports this Hadoop storage layer and is able to use existing components like Hbase to sort efficiently the data. The data can be written into native format (tabular), with an efficient hash table (using Hbase). With none tabular data, it is possible to use directly the HDFS file system in RAW format.

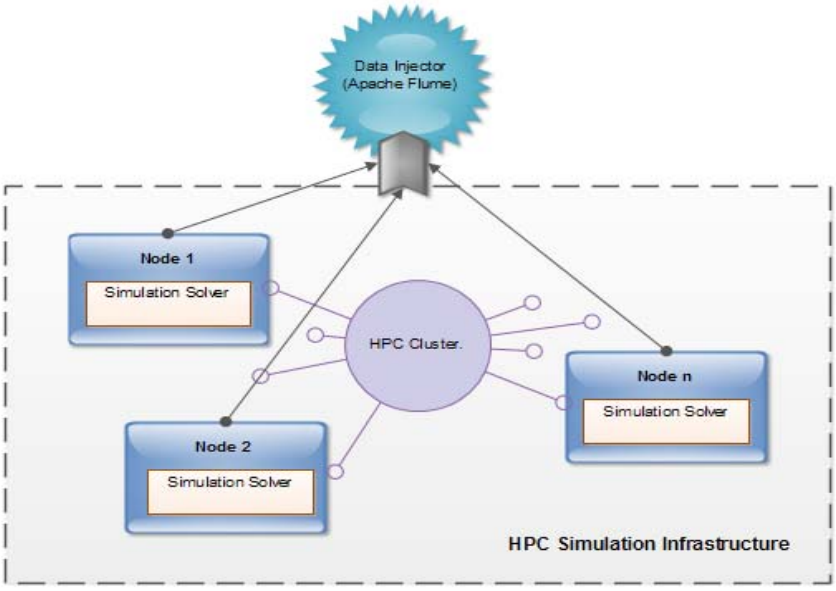


Figure 4: HPC Simulation Infrastructure

The third requirement concerns the query engine layer. The visualization client of the user will communicate with this layer to analyze the simulation data and get the results of these analyses for visualization. A user request, for instance display the average damage of a certain surface of the model, will issue one or more queries to the 'query engine layer'. Within this layer, a user query is analyzed and, depending on the work-load it represents, splitted into different sub-queries which require the collaboration of the Batch, Real Time (RT) and Analytics modules. At the end the sub-queries access the data layer, where a “search“ engine gathers information from the storage using a specific pipeline. Information is then delivered to the user into large pieces of data (Chunk). It is the most traditional layer used by database systems.

Heavy queries are queued in the Batch module and the same query is sent to the Analytics+RealTime modules but on a simplified model of the simulation data in order to fulfill the third requirement: ensure that the user is able to interact with the data. For the real time queries, a specific engine is in charge of fast streaming of data, and requires a dedicated format. To be able to reach this capability, pre-computed data and efficient data structures need to be stored into the data layer. The analytics module is in charge of all calculations and analysis tools of the query engine, example of these analytics tools are:

- Computation of multi-resolution model,
- The temporal and spatial statistical averaging of the results on particle simulations
- Extraction of features such as skins of volume data, streamline and iso-surface calculations, etc.

For the final requirement, the architecture has to support extensibility. Obviously, all components needed to fulfill the other requirements are not yet developed; an extensible platform will help us to integrate and manage them. To deal efficiently with BigData

requirements and extensibility, the VELaSSCo platform uses Hadoop. With its high extensibility it is possible to extend Hadoop with for example the support of the EDM DB on the data layer. Hadoop extensibility also allows deploying this ecosystem on top of any IT system. With this extensibility, it is also possible to deploy components regarding to the data set. For example, it possible to deploy a VELaSSCo platform to only stores none tabular data, thus Hadoop plugin will only be related to such kind of data.

3.1. VELaSSCo Platform

The VELaSSCo platform is the core block in the VELaSSCo architecture described in the section 3. From a layered perspective, the platform is composed of three layers:

- The first layer, I/O or Data layer is composed by a gathering tool (Flume) next to a monitoring tool to know the status of different process. The data query layer is connected to the I/O interface with the Hadoop ecosystem. Example of this interfaces are Hive Storm, Hbase, etc.
- The second layer, query engine layer, concerns the analytics, batch and real time modules of the platform. This layer is managed by YARN [5]. It is the specific resource manager for Hadoop 2.x. With this software Hadoop supports any kind of computational model. This strategy enables to support more complex computational models, and MapReduce is not anymore the only option. Thus other tools like multi-resolution, convertors, etc. are feasible.
- Finally, the storage component, concerns the storage system of the platform. With Hadoop extensibility, most existing storage systems are supported. VELaSSCo will not only work with Hadoop storage system but also provide new storage facilities with the EDM DB. This strategy is similar to the HadoopDB approach.

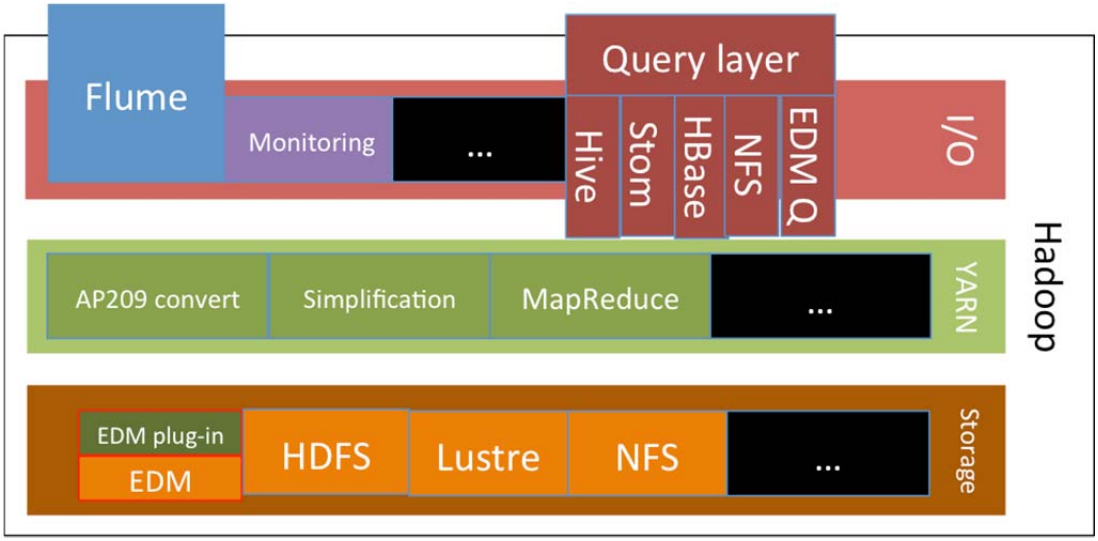


Figure 5. VELaSSCo core.

A logical view of the platform disclosing the different layers from which the platform is composed is depicted in Figure 6.

There will be two layers:

- Data Layer: which is responsible to store, access and translate the data and will answer the data queries coming from the "Query Engine Layer" and, eventually, when data is ingested in the platform, it may generate simplification queries, to create coarser views of the simulation data (corresponds to the I/O and Storage layer of Figure 5);
- Query Engine Layer: which is the responsible for receiving the user requests (VELaSSCo queries) from the visualization client, process them, generating data queries to the Data Layer, processing the returned results, eventually merging them, and format this results in a gpu-friendly way for the visualization client (corresponds to the Yarn layer of Figure 5).

In addition to these two layers, two libraries/modules will allow the connection between the platform and the final user:

- Data Ingestion library: which will provide a mechanism to ingest data from the computer engineering simulation solvers to the VELaSSCo platform. After establishing the communication with the Data Layer, including user validation, the ingestion module will send the data to the Hadoop Abstract File System. The Data Layer will store the data in P4 / GiD format if the HDFS is used as storage or translate the data to STEP/AP209 when using the EDM-plugin and EDM-DB system.
- Platform Access library: will communicate the visualization client and translate user requests into VELaSSCo queries, receive the results and show them to the user. The module will also generate VELaSSCo queries automatically, for instance when the user just navigates through the simulation model, or to ensure interactivity it will handle coarser models and issue full-detailed queries when the user zoom on parts of the simulation model.

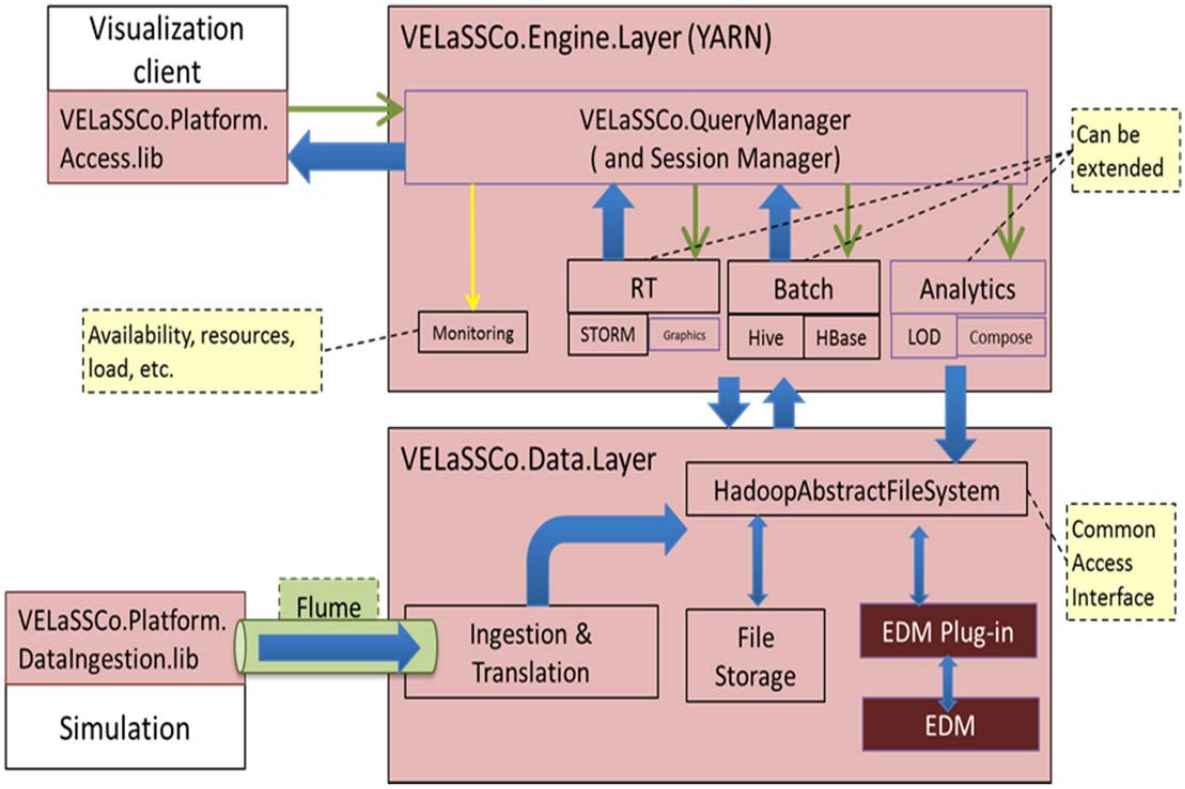


Figure 6. Layers and modules involved in the platform

Inside the Query Engine layer following modules are to be developed:

- QueryManager: responsible for managing the VELaSSCo queries, evaluate their costs and issue Fast queries and, eventually, Batch queries. If a query is too expensive, i.e. it will take too much time to evaluate, this module will launch both, a simplified version of the query (using a simplified version of the simulation data) to provide feed-back to the visualization client and a full detailed batch query, which will access the full-detailed simulation data. The queryManager will also synchronize and control the visualization parameters and issue visualization queries to the Real Time module, like the 'traveling' through the model for an interactive navigation.
- Analytics module: responsible to perform data analytics over the stored simulation results. It will also provide a cost estimation of the data analytic query to help the Query Manager evaluate in which mode is the analytics to be evaluated: in Speed mode or in Batch mode. This module will also generate coarser versions of the simulation data in order to provide fast feedback to the user and the Discrete to continuum transformation needed by the DEM simulation data.
- Batch module: responsible to perform heavy data analytics and to provide feedback on the progress of the query.
- Real Time module: responsible of providing feed-back to the user whether in full detail (for instance of detailed views of navigation through the simulation model),

simplified version (for heavy queries and when the coarse view is present) or progress indicator of the batch queries being executed; also for preparing the data for a fast visualization in the visualization client.

As already explained the communication between the Data Ingestion library (glued to the simulation program) and the Data Layer will be done using Apache Flume, and the communication between the visualization client + platform access library and the Query Engine Layer will be done using Apache's Thrift.

As for the communications between modules, as they are modules integrated inside a Hadoop framework, it will follow Hadoop recommendations just adapted for the FEM/DEM analytics needs.

The data to be passed between components will be reduced to basic types (string, integers, long integers (int64, doubles) or basic type lists/arrays.

The GPU-friendly format used to communicate the results of the query between the Real Time module and the visualization client will be detailed in a following annex, but can be viewed as a byte-stream or a byte-array

3.2. Communication pipelines

A detail overview of the platform is presented in Figure 7.

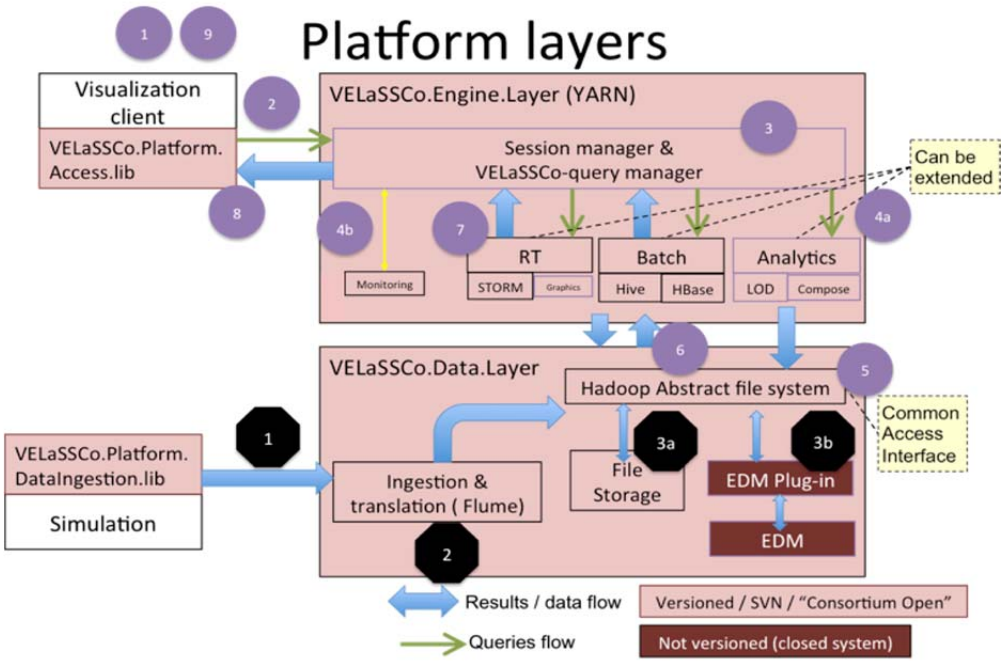


Figure 7. Communication Pipelines.

This schema gives an overview of all the communication pipelines. It presents a deeper representation of the infrastructure with the communication data flow. Blue arrows represent the data transfers, while green arrows represent queries.

In the engine layer (Yarn), some examples of plug-ins are presented; each module (batch, RT and analytics) can be extended with other applications. This strategy allows bringing new capabilities to the platform in the future. In Figure 7, two execution queries are presented: on the server side (black octagons) and on the user side (purple circles).

The first pipeline is representative of a storage plan. For this purpose, only compute nodes are involved in the process. In (1), data is produced by different nodes using their simulation engine. Then, Flume agents aggregate this data and write it into Hadoop through the abstract file system. This abstract file system then writes data into the appropriate file storage system, for example on a standard storage system (HDFS, Lustre … 3a) or using EDM, with a conversion from a file format to EDM/Step AP209 (3b).

For the second workload, a query based on visualization is performed, and transmitted to the query layer (2) using Thrift. The user account is validated and the query is transformed into an analytic query (for example mesh simplification) (query 4a). The analytics module gathers information from the file system and processes it. Eventually a quick simplification is performed and the produced data is stored (and obviously dispatched to the correct storage system) (5 and 6). Then the produced data is sent to the real time and transmits the result data, with Thrift, in a suitable format for the visualization engine and finally, the rendering is performed by the viewer (8 and 9). Information concerning the query and progress is sent to the monitoring tool (4b) which also monitors availability and resources statistics of the platform

This example shows how storage and queries are processed in the platform. An important point of this architecture is the support for extensibility. All necessary plug-ins can be deployed, regarding which data will be stored and which computations are performed.

To avoid multiple connections and multiple protocols, the query related components are in charge of communication between the user and all compute models. With this strategy users can perform Hive, Hbase or any kind of other query in a transparent manner.

The generic base of existing software developed for the Hadoop ecosystem provides multiple options and opportunities to adapt the VELaSSCo platform to a large number of applications and user requirements.

To validate some of these components, we performed an experimentation based on a new tool called “GDF” (**G**eneric **D**eployment tool **F**or Hadoop)[6]. It allows the deployment of a set of Hadoop components, based on a description of user requirements. This strategy allows supporting any IT system. Each required tool is then deployed over the user specific IT system: each layer of the platform (from the IO to storage) will be fed by the tool. In the current state of the project, some components are missing: they are not yet developed (for

example: the global query solver, EDM plug-ins, etc.). These specific parts will be a major contribution of the project.

3.3. Architecture Views

In the current section, we present two examples of the architecture with some specific plugins: one for EDM and one for open-source software.

3.3.1 View based on the EDM Database.

Considering the simulation process, it can be considered that all cluster nodes will run the entire number of time steps. The solvers will send their data to the platform by means of the Data Injector building block (a detailed explanation of this component is provided in section 5.1). They write results in GID-format or HDF5. The result data for completed time steps will be extracted from the HPC while the remaining simulation time steps are still being processed. Figure 8 shows how the result data are then converted to AP209, merged and stored into EDM DB.

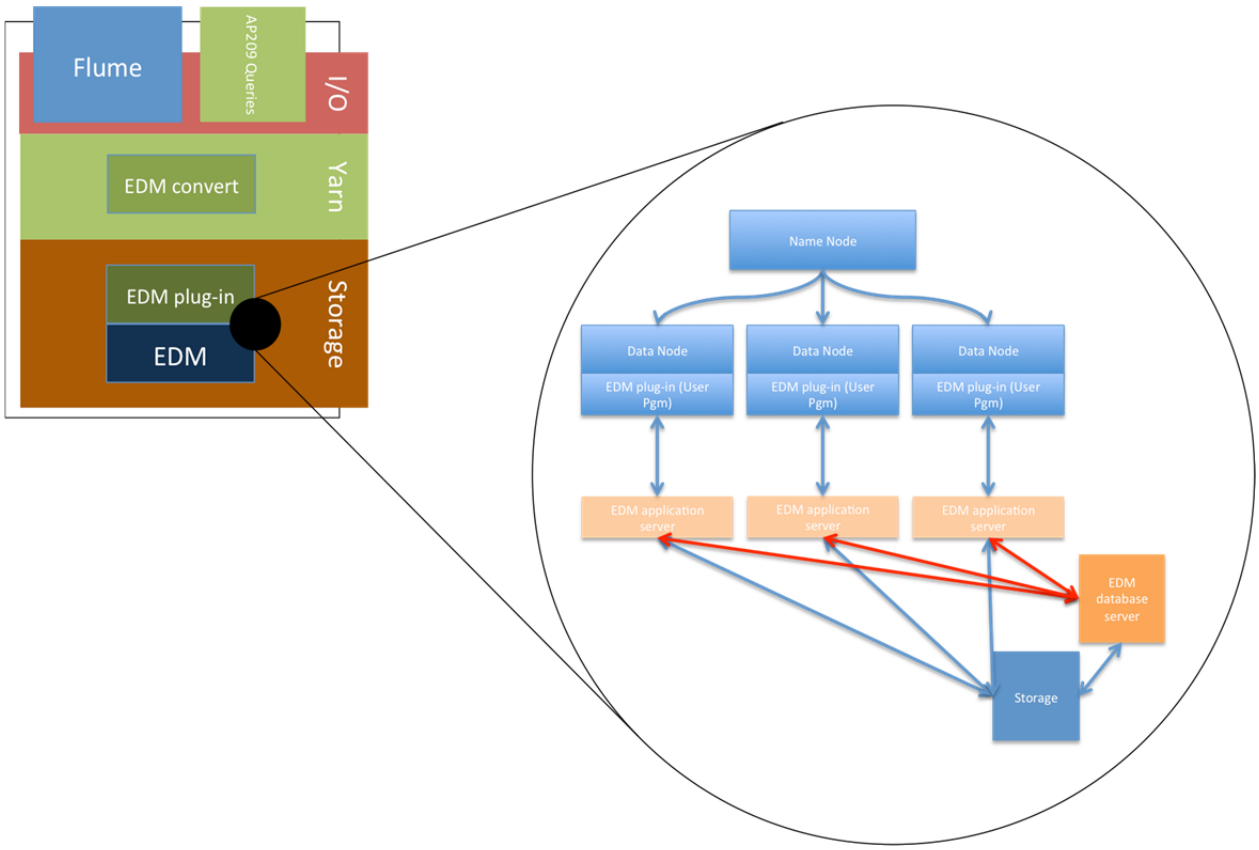


Figure 8. EDM Zoom In perspective.

The conversion into AP209 is done before they are imported into EDM. The EDM DB may be located centrally or distributed over several cluster nodes, whatever gives best performance.

The merge may be done either right behind the solvers (Map/reduce and HBase) or after the import into EDM; EDM has specific merge functionality. In the case of CIMNE solver, GID has already a method to merge data in GID-format, but that works only on single cluster nodes.

Data Injector includes a set of synchronizers that will store the raw data without merging into HBase. From there they may be read and converted for storage in AP209/EDM by means of EDM plugin.

Data shall be permanently stored in the most efficient way. EXPRESS and EDM seem to be an appropriate solution for standards compliant data. Files of proprietary formats can also be stored in EDM.

The existing visualization engines GiD and iFX will retrieve the result data by real-time and batch queries from the distributed EDM by means of the EDM plugin building block which will provide a query solver based on Hadoop.

3.3.2 View based on a full open source software stack.

In this architectural view all the components are included in the open source platform. With this strategy, the full power of the Hadoop ecosystem is released: the system is based on file the storage system, and all the plugins have already been efficiently developed and disseminated worldwide for such all kind of data. Also, a huge client base exists today worldwide including IT giants, e.g., Google, Yahoo, Amazon, to name a few.

In this example Figure 9, the Flume agent performs the data gathering. Data is stored into HDFS using the Hbase data layout. Hive can be extended to store data into an Hbase format. And Flume agents can be connected to the HBase storage system. A multi-resolution agent creates different levels of detail for each dataset. The physical storage used is based on HDFS. In this architecture, for the query model, two layers are available: one using Hive and the other using Hbase. Computation is performed using the new computation scheduler of Hadoop named YARN [5].

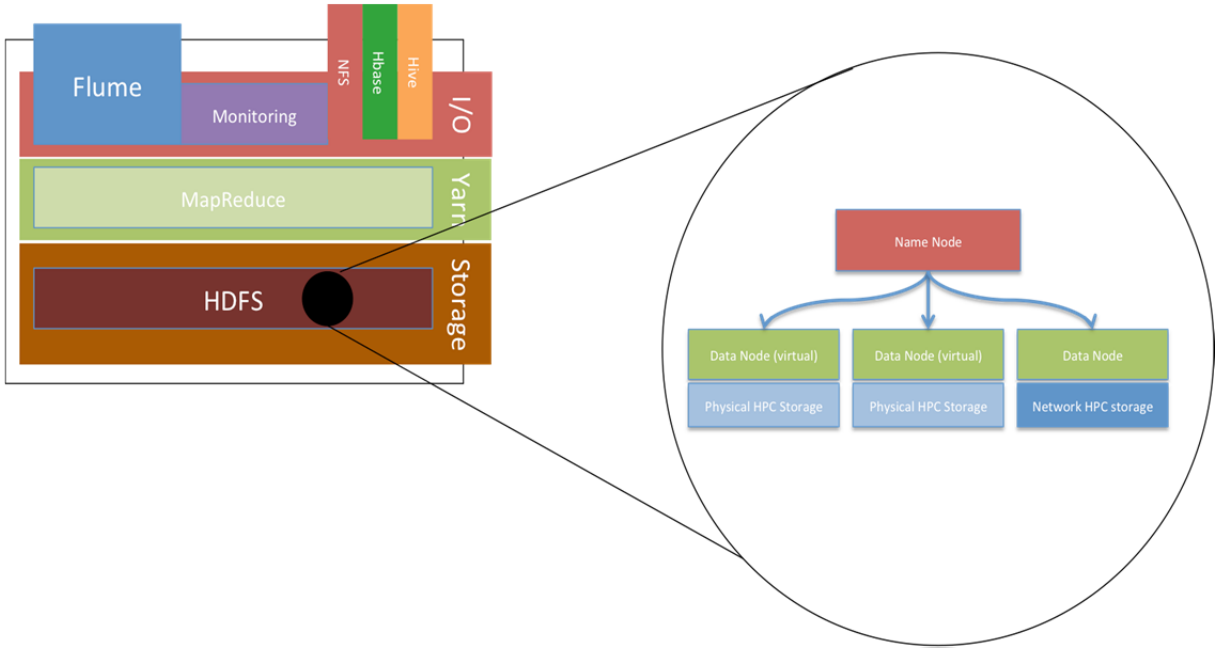


Figure 9. Example of the open source stack

For the visualization software, the connection to the platform is performed using an extension based on the Thrift tool [7]. Thrift is able to provide all the necessary classes to create a client server application, using any computer programming language.

Moreover, to increase the computation capabilities of such a solution, it is possible to use virtualization based on Linux "containers" [9], which, based on our experiments, brings a higher efficiency than virtual machines, e.g., Virtual Box [8]. Except for the Flume agent, this architecture is already deployable using our Hadoop deployment tool.

All these components have been validated during a comparative study, and usages of both plugins (Hive and Hbase) are relevant. Hbase is more suitable to extract specific content from a dataset, while Hive is more relevant for extracting a whole dataset. In a current paper [6], we are making an evaluation of three of these components for read operations on different architectures: Hive, Hbase and NFS, using both containers and virtual machines for performance comparisons [6].

4. HPC Simulation Infrastructure

The first prototype of VELaSSCo platform will be installed and tested in two dedicated nodes that have already purchased. The specifications of theses nodes are described in Table 1. These two nodes are already installed and working in the framework of Eddie cluster of the Edinburgh Compute and Data Facility (ECDF), the institutional service provider of High Performance Computing and Research Data Services to the University of Edinburgh.

The Eddie cluster is composed of worker nodes that run an industry standard Linux based operating system, specifically Scientific Linux release 6.5. Currently, Eddie cluster comprises two main parts:

- Mark2Phase1 - 130 IBM dx360M3 iDataPlex servers, each with two Intel Xeon E5620 quad-core processors. All these nodes are connected by a GigabitEthernet network with a 10 Gigabit network core.
- Mark2Phase2 - 156 IBM dx360M3 iDataPlex servers, each with two Intel Xeon E5645 six-core processors. All these nodes are connected by Gigabit Ethernet network, and 68 of them are also connected by a QDR QLogic Infiniband network- for Message Passing Interface (MPI) jobs.

Eddie can also deal with distributed memory parallel tasks where fast inter-node communication is required. The Infiniband equipped portion is especially suited to this role. There are 68 nodes in this section giving the capability to run up to 816 way parallel jobs, although in general we expect that most such jobs will utilize 256 or fewer cores.

Eddie is connected to a large amount of high performance disk storage via a number of dedicated machines. These machines then share the data across the cluster using a parallel file system. For VELaSSCo platform, it is planned up to 200TB (currently 50 TB) of data storage as working capacity for visualization of data sets, with data backup for appropriate project golden copy data and dedicated LTO6 tape jukebox to ingest the provided data sets. The use of a dedicated tape jukebox will allow easy ingest of data, while a high bandwidth connection with the ECDF “Eddie” cluster and the HECToR and ARCHER UK supercomputers will allow easy transfer of data to and from high-performance HPC storage.

CPU	2 x Intel Xeon ES-2680v2 2.8GHz (10-core, 20-threads)
RAM	384 GB RDIMM, 1600 MHz RAM
Hard disks	2 x 146 GB, SAS 15K RPM Hard drive 2 x 1.2TB SAS 10K RPM Hard drive
External storage	50 TB (extendible up to 200 TB) of Network-Attached Storage (NAS)

Table 1. Specifications of two dedicated nodes for VELaSSCo platform

5. Big Data Framework: Data Injection, Storage and Computation

As it was described in detail in section 3, the core of the VELaSSCo architecture is the Big Data Framework in which the VELaSSCo is based on. From the functional point of view the Big Data framework will provide support for data injection, storage for raw data and computation. Next subsections provide a detailed description of each one of these functional blocks.

5.1. Data Injection

Data injection functionality will be provided by Atos through DataBlaster REST service, which basically aims to receive the input data from different simulations, managing and splitting them into different threads to optimize the process and finally store this information into a database. Figure 10 displays the three main blocks which compose the workflow:

Figure 10. DataBlaster main workflow and components.

- Simulation Input data represents the information generated by each simulation process and sent to the event manager through the API REST service.
- The Event Manager is implemented with Flume which basically allows multiplexing the input into different pipelines.
- And finally the HBase table, which is created to store the information delivered by Flume after processing.

5.1.1 Apache Flume Pipelines

Apache Flume is a distributed and secure to collect, aggregate and move large volumes of data from different sources logs from a centralized data warehouse system.

Using Apache Flume not only adheres to the aggregation of data from logs. Because data sources are configurable, allowing Flume be used to collect data from events related to network traffic, social networking, e-mail messages to almost any possible data source.

The use of Flume within VElaSSCo will focus on providing the required pipelines to process simulation data created, in order to split it efficiently in several pieces that will be eventually stored in a NoSQL database, in a predefined schema. To do so, one

of the requirements on VELaSSCo is creating one or multiple agents with Flume to synchronize on HBase repository. To do so, the pipelines are provided after multiplexing channels for agent. This is the case when we want to use the same agent over different channels to communicate sources and sinks like in Figure 11:

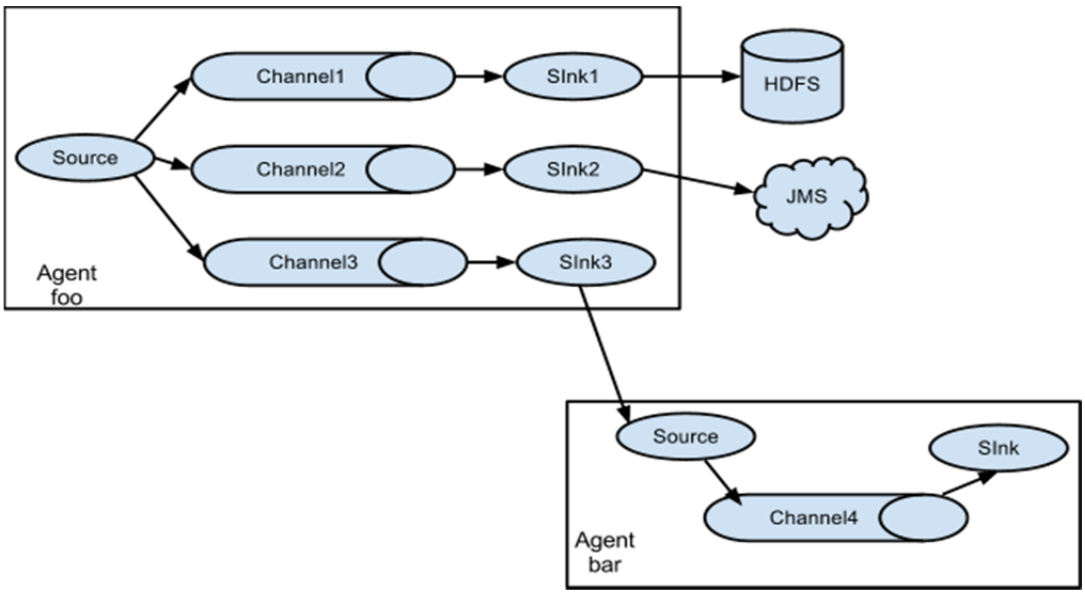


Figure 11. Flume multiplexing channels.

To do so, we need to change configuration file for every different channel:
 # cat conf/channel1-flume-conf.properties:

```
agent.sources=avroSource
agent.channels=memoryChannel
agent.sinks=hbaseSink
agent.sources.avroSource.type=avro
agent.sources.avroSource.channels=memoryChannel
agent.sources.avroSource.bind=0.0.0.0
agent.sources.avroSource.port=61616
agent.sources.avroSource.interceptors=i1
agent.sources.avroSource.interceptors.i1.type=timestamp
agent.channels.memoryChannel.type=memory
agent.sinks.hbaseSink.type=hbase
agent.sinks.hbaseSink.channel=memoryChannel
agent.sinks.hbaseSink.table=velassco_simuldata
# filling first column
agent.sinks.hbaseSink.columnFamily=simulationInfo
agent.sinks.hbaseSink.batchSize = 5000
```

Also we need to add specific sink in order to split the input into different table columns:

```
# splitting input parameters
agent.sinks.hbaseSink.serializer=org.apache.flume.sink.hbase.RegexHbaseEventSerializer
agent.sinks.hbaseSink.serializer.regex=(.+)-(.+)-(.+)$
agent.sinks.hbaseSink.serializer.colNames=simulId,resultPart,raw_data
```

Therefore, the Event Manager used is Flume, so the pipelines are created multiplexing different channels for the same agent, assigning different port for each one of the pipeline. Figure 12 represents the pipelines within the event manager:

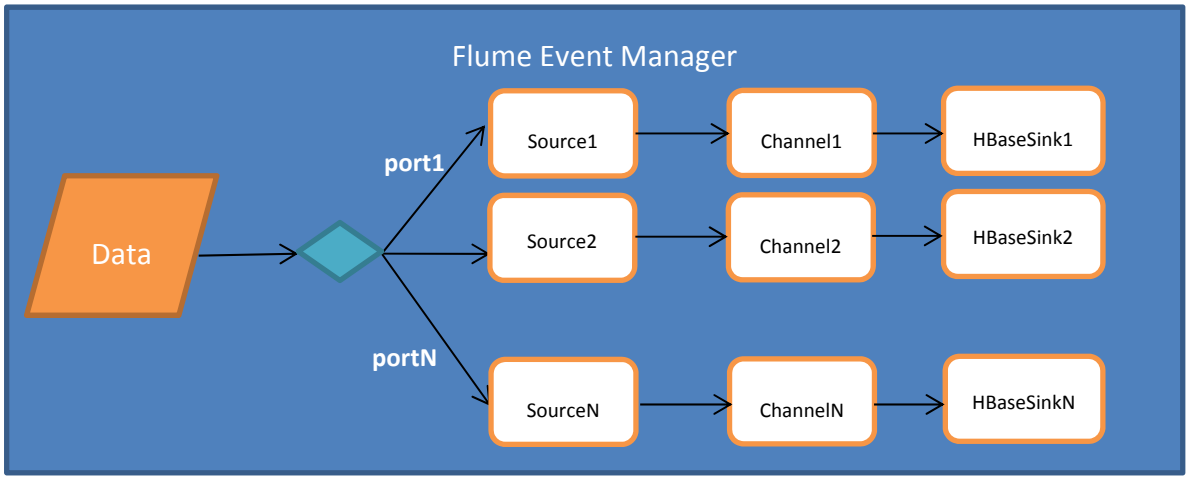


Figure 12. Flume Event Manager.

5.2. Data Storage

The current section provides a detailed description about the different types of storage included in the VELaSSCo architecture based on NON SQL in the case of Simulation Raw Data and based on EXPRESS (ISO 10303) in the case of AP209.

5.2.1 HBase as NON SQL Storage for raw data

Apache HBase is the Hadoop database, a distributed, scalable, big data store. Apache HBase use is recommended when you need random, real-time read/write access to your Big Data. The use of HBase on VELaSSCo is dedicated to store the output data return from Flume through the different pipelines. To store this information, first it is necessary create a schema like Hbase table with the required column fields:

- create 'velassco_simuldata','simulation_info'

Concretely, the fields which are going to be stored here are:

- simulId: Identifer of simulation part

- partNumber: The number of the part stored
- Sim_raw_data: this is the raw data to be stored. It could be a large amount of information so it is important to divide it into several pieces.

All the fields described above are store into a single column called 'simulation_Info'.

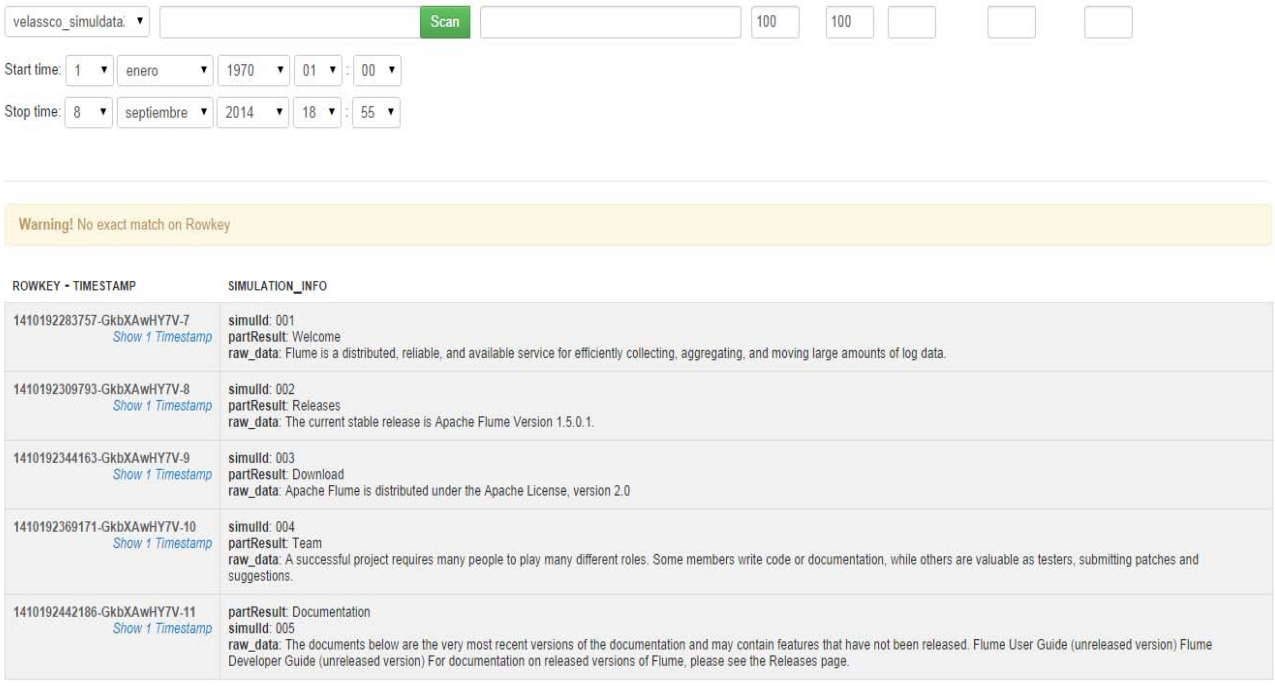
VELaSSCo simuldata database has been created in order to store separately every set of data sent through flume into different columns. The main goal is being able to deal with concurrent access to database to store a large number of registries with fields described above in 'simulation_info' column:

For example, one registry would look like:

rowkey - timestamp	simulation_info
1410192283757-GkbXAwHY7V-7 Show 1 Timestamp	simulId: 001 partResult: Welcome raw_data: Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.

In row registry above, it is displayed the rowkey – timestamp auto generated when data was inserted and 'simulation_info' columns with field values for simulId, partResult, raw_data.

Example of different registries stored in velassco_simuldata:



5.2.2 EDM Hadoop Plugin

The EDM Hadoop plugin can be implemented along the same pattern as the current implementation of the EDM web services. The main building blocks of this implementation are a web server front end and the EDMserver™. The front end to the server is implemented as servlets in a Tomcat web server. See the Figure 13:

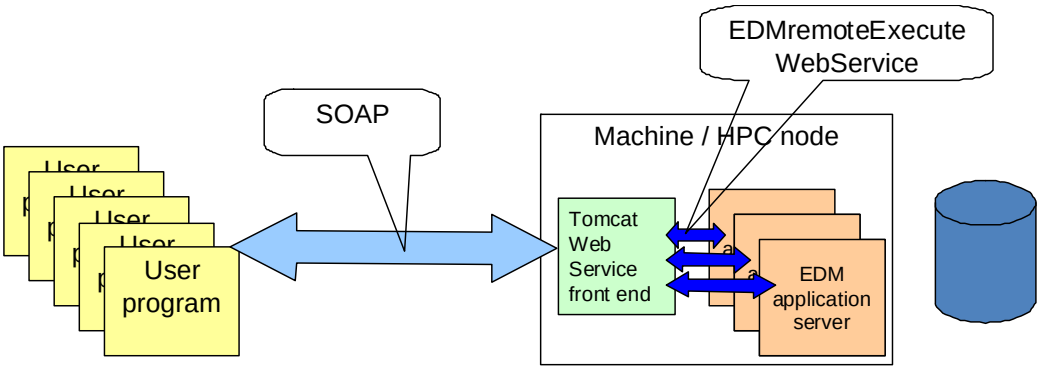


Figure 13 - Current EDM web service implementation

The servlets communicates with the *EDMserver*™ via two .dlls, one .dll that is the bridge between the Java code in the servlets and the general client module for communication with the *EDMserver*™, *EDMinterface*™.

The Hadoop EDM plugin will use the general Java interface to the *EDMserver*™. This interface is named EDOM3 and makes all server functionality available to a Java client program. See Figure 14 . The Hadoop EDM plugin will then consist of the following modules:

1. Java module that implements the Hadoop / *EDMserver*™ interface using EDOM3.
2. The EDOM3 Java layer.
3. JNI (Java Native Interface) based module to communicate between the EDOM3 Java layer and *EDMinterface*™.
4. *EDMinterface*™. This is the general interface for clients that will communicate with an *EDMserver*™.

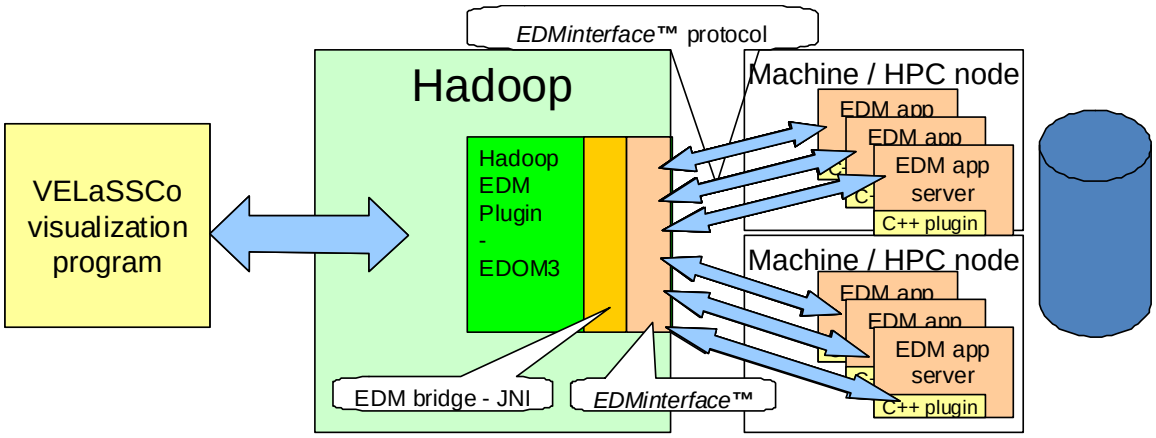


Figure 14 - VELaSSCo platform with Hadoop and EDM plugin

With the EDOM3 Java interface the Hadoop EDM plugin can:

- Create and delete databases
- Create and delete repositories and data models (data sets).
- Execute queries on selected data models
- Upload and download STEP (P21 and P28) files to/from data models.
- Retrieve database dictionary information.
- Install and compile Express-X programs

The domain that is simulated is divided into several sub-domains. The simulation of one sub-domain is executed on one HPC node. JOTNE proposes that the results of the simulation of one sub-domain are stored in one EDM data model (dataset). Each data model will contain the results from all time steps. All the data models from one simulation are contained in one EDM repository. This repository must also contain one data model that contains "global data" like type of simulation, name, description, dates, owner etc. and a list of the sub-domain data models.

Injection of new "result time-step slices" while displaying previous results can be implemented by having two EDM application servers pr. EDM data model (sub-domain). One serves the display queries and one adds new "result time-step slices".

The visualization query mechanism will be implemented as a C++ plugin to the EDM application server. The visualization engine executes database queries by executing methods in the server side C++ plugin. For example, a query that returns the necessary data for visualization of a cut plane may in C++ look like:

```
CutplaneData *VELaSSCoViewer::getCutPLaneData(.....)
{
  ...
}
```

Inside this method there can be a call to the VELaSSCo server that in the EDM way can look like:

```
edmiRemoteExecuteCppMethod(
  pluginName,          // string - .dll file name
  methodName,          // string
  targetDataset,       // specification of the datasets to be accessed
  mapSpecifications,   // data that is used to divide query into sub-queries
  Input parameters     // binary data (blob)
  Memory allocator(s) – for return values
  Return values        // binary data (blob)
);
```

The input data edmiRemoteExecuteCppMethod is the input parameters to VELaSSCoViewer::getCutPLaneData(.....) serialized to a binary data set (blob) ready for transfer to the server. Likewise is the return values from the query a blob that must be deserialized inside VELaSSCoViewer::getCutPLaneData(.....). If the input parameters and the return values are bigger than a certain limit, the data will be compressed before sending and decompressed at the receiving site.

In the VELaSSCo platform the edmiRemoteExecuteCppMethod call shall be started in all EDM application servers that shall take part in the execution. The execution of the edmiRemoteExecuteCppMethod on the client will call Hadoop in some way. And Hadoop shall then distribute the input parameters to all EDM application servers.

Pre-computed data sets can be stored in the EDM database as an EDM file attribute which is binary data stored in separated files in the database folder. By this mechanism it is very easy to access pre-computed data sets.

How the data shall be returned to the Visualization engine is an open question. Shall the Visualization engine know that the simulation domain is divided into sub-domains? And, if so, shall it receive the query result from each sub-domain separately?

The EDM Hadoop Plugin shall also be possible to plug into Flume. There the data to be ingested to the VELaSSCo database will be sent to the EDM server as input parameters to an ingestion method in the server side C++ plugin.

5.2.3 Events generated by the data ingestion

When a simulation program ingest data into the VELaSSCo platform several queries will be automatically triggered in order to already have pre-computed information when the user interacts with the simulation models stored in the platform.

This is the list of the queries that will eventually be triggered by the Data Ingestion module, from the Data Layer, depending on the type of the ingested data:

- Model information summary: with name, number of partitions, Bounding Box of the domain, mesh and result information (number of nodes, elements, time-steps and result names), and validation status initialized with default value.
- Discrete to continuum transformation, with default parameters.
- One, or more, simplified versions of the simulation model
- Pre-compute some frequent user queries detected by the data pattern detection/identification mechanism.

6. Query Manager

The Query Manager (QM) module of the VELaSSCo.QueryEngine.Layer is responsible of:

- receive and manage VELaSSCo queries
- ensure feed-back to the user

When receiving a VELaSSCo query from the visualization client the QM will evaluate the cost of executing this query. If the cost is small, them it will pass the query to the analytics module and receive the results of the query from the Real Time module and send the information back to the visualization client.

If the query is too heavy, then it will pass the query to the batch module and ask the analytics module for a simplified evaluation of the query, i.e. to evaluate the query over a simplified version of the simulated data. If the simplified version of the simulation model does not exist, then it will issue a coarsening query to generate a simplified version of the simulation model. The QM will receive result of the query over the simplified version and the progress of the batch queries and forward them to the user. At any moment the query may be stopped by the user, then the QM will stop the batch query evaluation but not the coarsening query if it has been generated, as the simplified version of the simulation model may be need for other queries. Figure 15 and Figure 16 display the workflow of both cases.

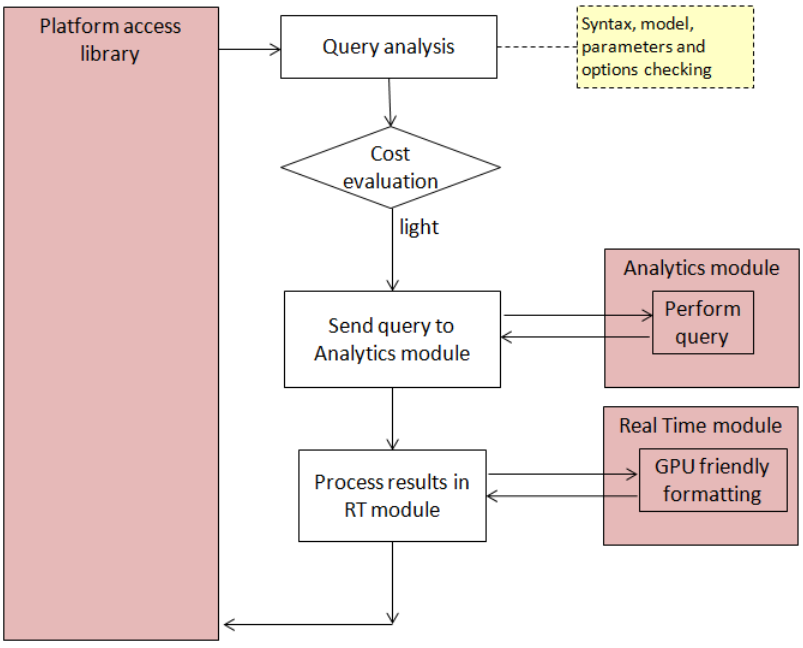


Figure 15. Steps to follow when processing light VELaSSCo queries

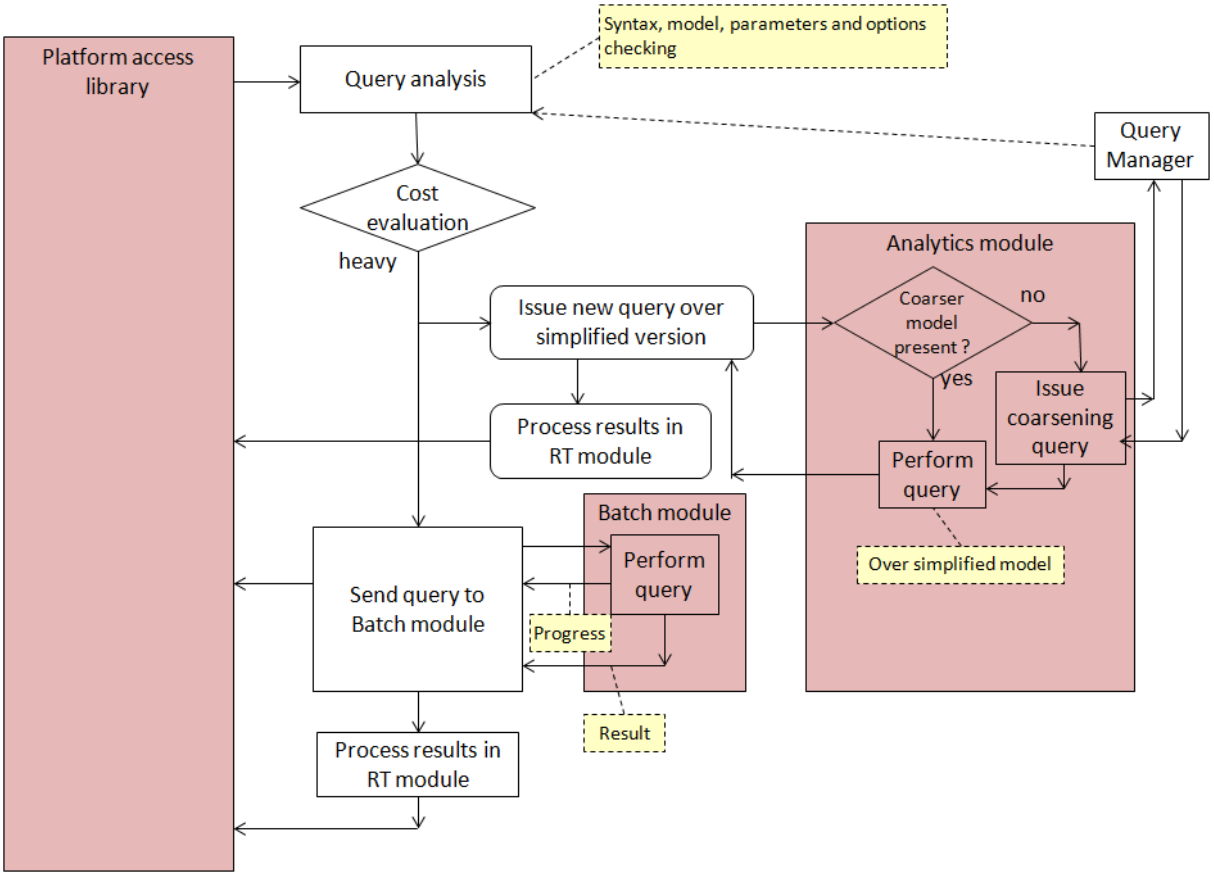


Figure 16. Steps to follow when processing heavy VELaSSCo queries

7. Data Analytics

The data analytics module of the VELaSSCo.QueryEngine.Layer will perform the data analytics over the stored simulation results. It will also provide a cost estimation of the data analytic query to help the Query Manager evaluate in which mode is the analytics to be evaluated: in Speed mode or in Batch mode. This module also contains the coarsening algorithms to build simplified versions of the simulation data in order to provide fast feedback to the user and the Discrete to continuum transformation needed by the DEM simulation data. These two lasts processes will be launched by the Data ingestion module of the data layer or by the query manager module depending on user parameters or the need to provide feed-back to the user.

The first VELaSSCo queries to be implemented will help determine the proper formats of communication between the different modules of the different layers and the libraries to access the platform. It will also help to further experiment with the platform and choose the proper tools, and their configuration for the correct and efficient execution of VELaSSCo queries and access and storage of the simulation data.

The list of the first queries is:

- Session connection:
  - Validation
  - List of models for model selection: with some of the already gathered information: with name, number of partitions, Bounding Box of the domain, mesh and result information (number of nodes, elements, time-steps and result names), validation status
- Model view:
  - Get already existing groups/layers (Surfaces, line meshes of DEM particles)
  - Extract skin of Volume meshes, and store it,
  - Store a thumbnail of the simulation model,
  - Change the validation status of the model
- Results view:
  - Get list of analysis, steps and result properties (name, type, location, …) present in the simulation model,
  - Given a list of nodes or elements, and a result identification (name, step, analysis) get the result values for those nodes or elements,
  - Given a point in space and a result identification (name, step, analysis) interpolate and get the result value,
  - Discrete to continuum transformation (DEM) according to default/provided parameters, as fast/light query and as batch/heavy query, to help evaluate cost of queries and implement batch module). Also store this transformation in the Data Layer.

These are common queries for both FEM and DEM type of simulation data.

For the 'Session connection' queries, a user identification will be needed to check permissions, and, eventually, maintain and store user-related model-independent data (like permissions, name …) or model related data, like thumbnails, used queries (like cut-planes definitions, iso-surfaces values, …)

To provide information in the model selection step, some information will be stored together with the simulation model. This information includes name, number of partitions (i.e. parts used in the distributed computing of the simulation problem), Bounding Box of the domain, mesh and result information (number of nodes, elements, time-steps and result names), validation status. This information should be calculated and stored when the data is ingested in the platform for a faster interaction with the user in the model selection step.

The 'Model view' and 'Result view' queries listed above for the 'first platform iteration' will define the specific data-queries to the HadoppAbstractFileSystem (HAFS) and both storage solutions (HDFS and EDM). These data queries will be used widely in almost all following VELaSSCo queries and thus should be implemented in the most efficient way, but avoiding too much specialization.

The simulation user may define his own names and descriptions for analysis, mesh groups, results. May be these names are also translated into other languages. So instead of implementing the data query HAFS.getDisplacementsForNodes(…), the data query footprint should be:

> *HAFS.getResultForNodes(ModelHND: reference to the user selected model, ResultName: String, ResultStep: String/Double, ResultAnalysis: String, listOfNodesId: listOfIntegers) returns ResultValues: listOfDoubles*
>
> *if result is a scalar, then ResultValues.size() == ListOfNodesId.size()*
>
> *if result is a vector, then ResultValues.size() == ListOfNodesId.size()*3 (x, y, z)*
>
> *ir result is a matrix, then ResultValues.size() == ListOfNodesId.size()*6 (Sxx, Syy, Szz, Sxy, Sxz, Syz)*

The same is valid for mesh/group names …

On a further iteration, the result of the 'skin extraction' query may be stored together with the model.

Thumbnail and validation status of the model may be stored together with the model and can be different depending on the user.

Following sections list the queries grouped functionalities for the first prototype and the final prototype, as described in the D1.3a deliverable.

The listed queries will provide a cost estimation method to help the QueryManager evaluate the cost of executing the query and perform the proper actions: execute it, or issue a simplified version of the query and send it to the batch module.

Depending on the experiments the details of the queries may vary to reduce latencies and bandwidth requirements and ensure the interactivity with the user.

7.1. FEM data analytics

Queries applied to FEM simulation data are:

1) **Mesh queries**:

- *getMeshGroupList(ModelId)*: returns a list of the mesh groups present in the selected simulation model with their properties: name, eventually colour, number and type of elements.
- *getMeshData(ModelId, MeshName):* returns a list of nodes and elements of the mesh. This query may be spitted eventually into several sub-queries like: getGlobalMeshCoordinates, getMeshCoordinates, getMeshElements, etc.
- *getBoundaryMesh(ModelId, MeshName):* returns the boundary (skin) of the volume mesh: a list of triangles and/or a list of quadrilaterals, depending if the volume mesh has tetrahedra, hexahedra, prisms or pyramids. If it's not present calculates the boundary mesh it together with the simulation model and returns it. Depending on the calculation cost, this skin may be stored together with the simulation model.
- *getNodeList(BoundingBox):* returns a list of nodes (id and coordinates) inside the bounding box.
- *getElementList(BoundingBox)*: returns a list of elements (ids) inside the bounding box.
- *doCutPlane/Line/Segment(cutcoefficients):* performs a cut on the simulation model using the plane, line or segment coefficients and returns a list of triangles, lines with their coordinates.
- *getBoundaryConditions():* returns a list of (String and element id's or node id's) of the placed conditions on the simulation model
- *deleteModel(ModelId):* delete a simulation model (original, simplified or discrete-to-continuum transformed)
- *deleteMesh(MeshName):* deletes a mesh of the simulation model

2) **Result queries:**

- *getAnalysisAndStepList(model):* returns, for the user selected model, the list of analysis and for each analysis a list of the steps present in the analysis.
- *getResultList(model, analysis, step):* returns a list of result information for the results present in the provided step. This information contains the result's name, result's type, components names and units, among others.

- *getResultsForNodes(model, analysis, step, result name, component name/all, list of nodes id's)*: returns a list of the results values for the provided nodes id's.
- *getResultsForElements(model, analysis, step, result name, component name/all, list of elements id's)*: returns a list of the results values for the provided nodes id's.
- *getResultForPoint(model, analysis, step, result name, component name/all, coordinates):* returns the interpolation of the result for the provided coordinates. Eventually a variant of this query may be implementer for a list of coordinates where the result is to be interpolated.
- *doCutPlane/Line/SegmentWithResult(cutcoefficients, analysis, step, result name, component name/all):* performs a cut on the simulation model using the plane, line or segment coefficients and returns a list of triangles, lines with their coordinates, and result values on the coordinates.

  When the user performs a cut plane and changes the result visualized, several doCutPlane/doCutPlaneWithResult queries may be launched with the same cut coefficients. This may be inefficient. Depending on the experiments or a more close analysis, a variant, or variants of these queries will be implemented:

  one with temporary objects:

  CutObjectId coi = doCutPlane(parameters)

  results = interpolateResults(coi, result name 1, …)

  results = interpolateResults(coi, result name 2, …)

  results = interpolateResults(coi, result name 3, …)

  deleteCutObject(coi);

  or returning more information, including nodes and elements of the original mesh which are crossed my the cut-plane, with their interpolation factors) to visualization client:

  InterpolationInformation localCutInfo= doCutPlane(parameters)

  tmp = getResultsForNodes(localCutInfo.originalNodes, result name 1, …)

  results = local_interpolate(localCutInfo, tmp);

  tmp = getResultsForNodes(localCutInfo.originalNodes, result name 2, …)

  results = local_interpolate(InterpolationInformation, tmp);

  tmp = getResultsForNodes(localCutInfo.originalNodes, result name 3, …)

  results = local_interpolate(localCutInfo, tmp);

  or returning more information to visualization client

- *getIsosurfaceMesh(model, result name, step, analysis, iso-surface value):* get the triangle/line mesh of all results values equal to the one provided of the original mesh. A variant of this query may be implemented for a list of values.
- *getIsosurfaceMeshWithResults(model, iso-surface-result (name, step, analysis), iso-surface value, result name, step analysis):* get the isosurface mesh for the provided parameters and the second result values interpolated for this iso-surface.

  The iso-surface query is similar to the Cut-query. They have the same problematic and will be handled in the same way.
- *getResultStatistic(model, result name, which statistic):* returns the selected statistic (minimum, maximum, average, standard deviation) of the selected result. if it does not exists, it calculates it and stores it together with the simulation model.
- *getStreamLines(model, vector result name, list of seed points):* returns a list of mesh lines which are the stream-lines of the selected vector field. A variant of this query will also interpolate a second result over these stream-lines.

  The stream-line query is similar to the CutLine-query. They have the same problematic regarding the interpolation of results, and will be handled in the same way.
- *getCoarserModel(ModelId, error tolerance):* returns an identifier for the simplified view of the model. If it does not exist it calculates it and stores the information for further results interpolation or it interpolates all results for the simplified view.

For the final prototype some of these result queries are to be implemented for results defined on an element basis (on the gauss points). This will require extrapolation from the gauss points to the vertices of the elements and a global smoothing of these values, i.e. an average of the values from the elements incident to a vertex.

7.2. DEM data analytics

The DEM analytics will be conducted by means of the **P**article **P**re/**P**ost **P**rocessor (**P4**). P4 is a data analytics toolbox developed by the University of Edinburgh. This toolbox allows the analysis of data from DEM simulations. P4 will be adapted to work with the Hadoop system of the platform in order to compute the analytics queries. Moreover, the analytics algorithms of the P4 will be developed to be able to work in parallel and as consequence speed-up the computations that will allow fast analytics of the large simulation data. Thus, P4 will be integrated in data engine layer of VELaSSCo platform.

The analytic queries to be performed by P4 can be split into two different groups depending on the type of data:

1) **Discrete data**

The analytics of the discrete data are related to the results of the particles (velocity, position, angular velocity, …) and contact forces generated by the DEM solver. The analytics for this kind of data can be grouped into:

- Extraction of results for all the particles, a group of particles or an individual particle for a single or a set of simulation time-steps.
- Statistics results: calculation of relevant statistic parameters (mean, median, mode, standard deviation, histograms) of the particle and contact results for all particles, a single particle or a group of particles for one time-step or along several time-steps.

2) **Continuum data**

The analytics for continuum data require a conversion of the discrete data performed by means of coarse-graining and binning techniques for spatial and temporal averaging over the particle and contact datasets. Thus, the discrete information is projected into a continuum field for analysis of bulk quantitates such as solid fraction, bulk velocity, kinetic/contact stress, strain, etc. Moreover, the time averaging allows reducing the noise in the results, recognizing the flow behavior patterns on time and optimization of the sampling frequency. Concerning this conversion from discrete to the continuum, the following analytics queries will be developed:

- A specific multi-resolution adjustment for multi-resolution time averaging of particle/contact quantitates.
- Selective access data based on particle material/phase and bulk quantities levels.
- Spatial integrals of particle/contact quantitate.

The results obtained from this transformation to the continuum are similar to the results from FEM simulations. Different analytics will be developed for this kind of results that are similar to the FEM analytics described in section 7.1.

Depending on the computational cost of the different analytics described, these will be performed by one of different blocks of the VELaSSCo engine layer: Real Time, Analytics or Batch (see section 3). Thus, the analytics involving short computational times will be performed through the Real Time and Analytics blocks to provide a short response time to the queries from the user. In contrast, the analytics that require intensive computation will be performed through the Batch block. For instance, these will be the case of discrete to continuum transformation of large data sets.

7.3. Visualization analytics/algorithms

The results of the queries of the Analytics module will be formatted for a more efficient utilization of the gpu resources on the visualization client side.

Some of the visualization functions are (names are temporal):

- *prepareVisualizationMesh*(list of elements): returns a gpu-friendly formatted list of elements (triangles, quadrilaterals, lines) from the given ones. Eventually this method will be spitted into three for the different types of elements, for a better performance.
- *prepareVisualizationMesh*(list of elements, list of coordinates): returns a gpu-friendly formatted list of elements (triangles, quadrilaterals, lines) and coordinates from the given ones. Eventually this method will be spitted into three for the different types of elements, for a better performance. It will, eventually, also calculate normal on the vertices of the mesh.
- *prepareVisualizationMeshResults*(list of elements, list of results): same as above but including results. A second version including coordinates of the mesh vertices may be implemented.
- *prepareVisualizationMeshVectors*(list of elements, list of results): returns a gpu-friendly formatted list of elements and coordinates with vector objects on the vertices of the mesh for more efficient visualization on the client side.
- *get/setVisualizationParameters():* gets and sets visualization parameters for a better interactivity and results visualization. This includes, for instance:
    - number of colors used in the contour fill visualization for a better result compression,
    - level of detail to control the coarsening process used by the analytics module
- *get/setVisualizationBoundingBox():* gets and sets the world window to control the level of detail and amount of graphical primitives send back to the visualization client. Also used when "traveling through the model" query, following the progressive display approach.
- *getBSplineRepresentation(list of lines):* given a list of connected lines, which can represent one or a set of stream-lines, returns a more compact and high-level representation of the provided lines.
- *getbSplineRepresentation(list of triangles):* given a list of connected triangles, which can represent one or a set of iso-surfaces, the query returns a more compact and high-level representation of the provided triangulated surfaces.

Due to the tight interaction between the QueryManager, Analytics, even Batch modules with the RealTime visualization module, some algorithms and queries may have been overlooked but which are required for a successful implementation of the platform.

For the final prototype some of these result queries are to be implemented for results defined over the elements (on the gauss points). This will require extrapolation from the gauss points to the vertices of the elements and a global smoothing of these values, i.e. an average of the values from the elements incident to a vertex.

The algorithms and queries involved in the animation of results and particles along the simulation will be analyzed on a later time, as it requires a fully functional version of the platform for a fine-tuning of the different components involved which work very tight together. This streaming query will demand the most from the visualization client down to the Database storage solution used in the platform.

8. Visualization Engines

As illustrated in Figure 14, the visualization client is separated from the database infrastructure, and communicates with Hadoop to gain access to the VELaSSCo platform in order to send queries and to receive results. The visualization client basically consists of three main components: a visualization engine, a VELaSSCo access library, and one or more rendering nodes.

The visualization engine serves as visualization and post-processing framework, which is responsible for generating the final images that are presented to the user. Depending on the specific use case, this can either involve rendering of data that has been pre-computed by the VELaSSCo platform (e.g., a triangle mesh representing an iso-surface), or post-processing simulation data on the client side (e.g., generating an iso-surface from a volume dataset). As visualization engines GiD (CIMNE) and iFX (Fraunhofer) are employed (see below).

The VELaSSCo access library provides a communication layer to the VELaSSCo platform. It uses a specific application programming interface (API) for sending queries and receiving results. Both GiD and iFX provide a plugin-mechanism to enable extensions. To connect the access library to the visualization engines, a plugin for each framework will be developed, where each plugin will be linked to the library. Keeping platform access in a separated library will allow to also target other frameworks besides GiD and iFX. The library will make use of Apache Thrift [10][9] to facilitate interaction with Hadoop.

The rendering nodes are dedicated machines that can be exploited by the visualization engine. This enables distributed parallel visualization and post-processing on the client side. Ideally, the rendering engines make use of a hybrid setup, where each node is fitted with multiple CPUs and GPUs.

8.1. GiD

GiD [11] is a general pre and post processing software, whose graphical user interface (menus, windows, dialog boxes, etc.) is written in Tcl/Tk and the kernel uses the C++ language. The 3D visualization is performed using the OpenGL graphics library. It run on MS Windows and Linux. GiD will translate user requests into calls to the PlatforAccess library which will communicate to the VELaSSCo platform and generate the proper VELaSSCo queries. Some queries will also be generated automatically without the user interaction.

The user identification and login mechanism provided in the PlatformAcces library should be exposed to the user so that he can access to the platform.

Also the model selection process will require some modification in the graphical user interface so that the user is able to select any model stored in the platform.

The validation status query should also be exposed in the graphical user interface.

The visualization client should also provide a way to inform the user about the state of the issued queries: if the query is being executed, if it is queued in the batch module, its progress in the execution, state of the visualized results (simplified, full-detailed), etc.

The monitoring module of the VELaSSCo platform will also provide information about the state and resources used by the VELaSSCo platform. The visualization client should also provide a mechanism to show this information to the user.

Depending on the Analytics or Visualization queries other parameters will be required from the visualization queries, as the implementation of the platform evolves.

8.2. iFX

iFX [12] from Fraunhofer is a post-processing and visualization framework that has been designed for the engineer's graphic workstation. The objective is not to be generic but to provide best-in-class solutions for specific customer needs with the following aims in mind: leveraging GPU hardware (GPU shader) and spatial acceleration structures, efficient visual analysis and goal orientation, and visual quality realized by topology-independent interpolation schemes. iFX shall be extended in VELaSSCo to exploit OpenGL 3.x up to 4.3 features.

8.3. Visualization Query mechanisms

Due to the chain of components involved when issuing a request to the VELaSSCo platform, a query has to go through several steps:

On the client side, a user interacts with a graphical user interface of one of the visualization engines. User actions (e.g., moving to a new part of the model, or requesting an iso-surface with a specific iso-value) are then translated by the plugin component into a query message that is sent to the access library. The library will then use Thrift to contact Hadoop in the VELaSSCo platform. This will trigger either retrieving simulation results (to be post-processed on the client), or the computation of geometric data (to be visualized on the client). The resulting data is then sent back the same way to the visualization engine, which makes use of its rendering nodes to display or process the data.

Depending on the use case, the result of a query can be received as a single dataset, but results can also be fetched in a streaming or progressive manner. In a streaming setting, the data is delivered continuously by the VELaSSCo platform. An example would be to send data depending on the current camera position. A similar situation would be progressive display, where the data stream adds more and more detail to an initial coarse representation. For example, large iso-surfaces could be encoded as a progressive level-of-detail triangle mesh.

Regarding to the architecture this functional block refers to the PlatformAccess library, which connects the visualization client with the VELaSSCo platform, as schematized in Figure 7.

The library is responsible for:

- start, maintain and end the connection to the VELaSSCo platform,
- user validation,
- model selection,

- forward queries issued from the visualization client,
- eventually uncompress the data received and pass them to the client to visualize,
- providing information about the level of execution of the queries
- provide tools to visualize high level representation of the data (bspline)
- eventually provide interaction metaphors for specialized queries.

The library will also trigger some events and such events should be handled by the visualization. Some these events are:

- actualizing the state of a query execution, for instance when it is being executed in batch mode,
- actualizing the mesh and results graphical information when more data is being received on a multi-resolution visualization, this may include refinement of the high level representation of the data (bspline).
- platform state actualization as provided by the monitoring module which controls the health state of the VELaSSCo platform.

As we gain knowledge while developing the platform other interactions may be needed between the platform and the visualization clients.

9. Prototype(s) definition(s)

In the VELaSSCo project, different versions of the platform are targeted. We have planned to distribute two versions: an open source version, and a close source version. To validate these platforms, different prototypes will be produced. Each of these prototypes is used to validate some part of the final platform. In the next subsections we will present some of them.

9.1. VELaSSCo platform

The validation phases will be decomposed in different step. Obviously development of a complex application needs time, and resources. Thus it is necessary to adopt an incremental conception of the platform. The development of this strategy will be enhanced with an extensible framework, to enable component development (each new feature can be seen as a specific component).

As stated before, we have planned to use the Hadoop ecosystem as a skeleton of our application. Existing components of Hadoop will be in charge of the extensibility of the platform. Hadoop is a kind of operating system specifically designed to deal with big data. With the newer version of Hadoop, it is possible to deploy complex computation model over this platform. We will this particularity to apply computation over the platform. Hadoop can also interact with external component to provide new features to this platform. For example Hbase brings a tabular storage system to Hadoop, Hive brings SQL support on top of Hadoop, etc.

As stated previously in the document, different components set will compose this platform:

- VQM (which interact directly with the use app),
- Analyze, and query engines,
- The storage tool set,
- External tools (monitoring, Flume,…)

8.1.1 VELaSSCo Query Manager

This component is one of the most important parts of this project. It will be in charge of communication between the user application and the VELaSSCo Platform. This component is used to decompose a query made from the user visualization tool into different sub queries.

Decomposition is performed among different sub-query engines: a real-time layer, a batch layer and an analytics layer. This manager will be developing to produce the most suitable query for the best query engine regarding needs of a user. This tool is used as a user gateway with the platform. VQM will be linked tightly with the storage and components of the platform. This tool will be in charge of management of necessary component on the platform, it will run the necessary plugin with the appropriate communication layer. For

example, to communicate with Hive or Hbase, VQM will communicate using Thrift. In the next section, we will present some of these components.

8.1.2 Analyze and queries engine

This layer is decomposed into three subs components: Real-time, batch and analytics. Batch layer is used to send data into chunk of information. The real-time plugin will produce information before the end of computation. All of these tools use existing dataset in the storage layer. The analytics part is in charge of complexes computation (producing new information).

For the real-time engine, we have planned to use the storm extension, which is considered to be the real-time engine for Hadoop. If requirements of our solution are more complex than storm capability, thus, we will develop an alternative real-time query engine.

For the batch layer, it is already exists different tool; we will use Hive and Hbase to access structured data (which have a tabular look and feel). For unstructured data, direct file access on the storage layer will be use. With this strategy, we have reduced the development cost; some experiments have already validated use of Hbase and hive for FEM or DEM data. Hive and Hbase have to be combined to provide the best access strategy on data.

The analytics part will be the most important part of this layer. This component is a new one, which will be in charge of starting computation over data. To reduce development over, we will use most of existing computation tool for the subpart of this layer. Execution over this component is decomposed into 3 steps:

Convert data into the adapted format if dataset is not suitable with the app.

Run the computation on this (new) data set, and provide feedback on the execution.

Convert output data on the desired format.

All conversions are executed depending on requirements provided by the visualization engine.

8.1.3 Storage tool set (EDM)

The basic modules of The VELaSSCo system are Data ingestion module, Visualization query mechanism and Visualization engine. Development of prototypes for these modules can be started today using EDM. JOTNE proposes that the development starts by implementing standalone programs for each module. The ingestion module is a C++ standalone program that opens an EDM database and stores FEM and/or DEM data in it. The visualization query mechanism can also be a C++ standalone program that opens and reads the EDM database produced by the ingestion module. Later one can extend this program with the visualization engine. Then we have the first VELaSSCo prototype.

The next step in the development of EDM based prototypes is to build the Data ingestion module and the Visualization engine as client programs that communicate with an *EDMserver*™. The Visualization query mechanism will be implemented as C++ plug-in to the *EDMserver*™.

8.1.4 External tools

This step is in charge of providing to user all information regarding the execution of the tools, availability of the system, charge, resources, etc. Flume agent is also a component of this part.

9.2. VELaSSCo platform prototype development plan

To produce the most suitable platform, different step will be necessary. We will produce a first iteration of the distribution with a minimum set of components. This minimum set of components will answer initial queries of the platform.

The next step will be in charge of improving simple queries and validate all components with simple queries.

And finally, last part of the project will deal with more complex queries, we will provide all tool needed by the sub part of the analytics.

10. Conclusions

In this deliverable, we present a summary of the efforts provided in task T2.2 related to the "Specification of the Big Data architecture".

Firstly, in the document has been described the core concept of VELaSSCo Architecture which is the Lambda Architecture Approach next to some relevant concepts associated to VELaSSCo project from a conceptual and technical perspective

Once the Lambda Architecture Approach has been presented, the deliverable provides a description of the VELaSSCo architecture enriched with several views of the system from the component diagram point of view, the main communication pipelines and a couple of architecture views based on commercial and open source tools respectively.

On the other hand the document provides detailed information about the HPC simulation infrastructure and the Big Data Framework to be developed in the project for supporting the VELaSSCo platform.

Subsequently, queries applied to FEM and DEM simulation data are presented next to more relevant visualization algorithms and functions.

For covering the UI layer or visualization layer in the VELaSSCo architecture the visualization engines GiD and IFX are described in detailed in the final sections of the documents as visualization and post-processing tools being responsible of generating the final images that are presented to the user.

Finally, a set of tentative prototypes definitions are presented in the document.

11. References

[1] N. Marz and J. Warren, Big Data. Principles and best practices of scalable realtime data systems, Manning Publications Co., Manning Early 2012, September 2013 (est.).

[2] The Apache HBase Reference Guide: http://hbase.apache.org/book/book.html

[3] Introducing ElephantDB: a distributed database specialized in exporting data from Hadoop: http://computerhelpkansascity.blogspot.com.es/2012/06/introducing-elephantdb-distributed_26.html

[4] Tomer Shiran , Introduction to Apache Drill: http://cdn.oreillystatic.com/en/assets/1/event/91/An%20Introduction%20to%20Apache%20Drill%20Presentation.pdf

[5] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC 2013, pages 5:1 to 5:16, New York, NY, USA, 2013. ACM.

[6] B. Lange and T. Nguyen, A Hadoop distribution for engineering simulation, 2014

[7] http://thrift.apache.org

[8] https://www.virtualbox.org

[9] https://www.docker.com

[10] Apache Thrift: https://thrift.apache.org

[11] GiD: http://www.gidhome.com

[12] iFX: http:// www.i-fx.net

[13] Apache Storm: https://storm.incubator.apache.org/