VELaSSCo

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

D3.2 – Pre-computed, or on-demand computed, transformations stored in HPC: Engine to create multi-resolution models & co. from simulation data, first version ready for first prototype

Version 1.2

Deliverable Information

Grant Agreement no	619439
Web Site	http://www.velassco.eu/
Related WP & Task:	WP3, T3.2
Due date	September , 2015
Dissemination Level	PU
Nature	P
Author/s	Miguel Pasenau
Contributors	Giuseppe Filipone, Alvaro Janda, Heidi Dahl, Ivan Martinez Rodriguez, Jochen Haenisch, Abel Coll

Approvals

	Name	Institution	Date	OK
Author	Miguel Pasenau	CIMNE		
Task Leader	Miguel Pasenau	CIMNE		
WP Leader	Miguel Pasenau	CIMNE		
Coordinator	Abel Coll	CIMNE		

Change Log

Version	Description of Change
Version 0.0	Content definition
Version 1.0	Draft including contributions from partners
Version 1.1	Review done by Abel
Version 1.2	Review done by Heidi

Table of Contents

Table of figures

1. Introduction

This report is related to the complex queries that transform the simulation data into new data to be used by the visualization's client.

These transformations may bring a better understanding of the simulation data, or may help in the user experience and the visualization process.

The list of operations that correspond to the transformations of the first prototype are already discussed in D1.3, D2.2, D2.5 and 3.1and include:

- CalculateBoundingBox: calculates the 3D axis aligned box that encloses the simulation model, defined by two vertices: minimum and maximum xyz coordinates;
- CalculateBoundaryMesh: calculates the skin of a 3D volume, or the edges of a surface;
- CalculateSimplifiedMesh: generates a simplified version of the 3D model;
- Discrete2ContinuumTransformation: creates a representation of the DEM data, using a static FEM mesh, by averaging the simulated result in time and space.

The results of these transformations may be stored in the platform, as the execution cost of these operations may be very high, and its output is foreseen to be used often by the visualization client.

As already mentioned in D3.1 a *VELaSSCo Query* (VQuery) (which results from a user interaction) is decomposed in several operations. As an example, when the final user wants to visualize a volume mesh, the visualization client will issue a GetBoundaryMesh VQuery to the platform. The platform will first look if the VQuery has already been calculated and if so, will retrieve the requested information and send it back to the visualization client. If the VQuery has not been pre-computed then it will perform the CalculateBoundaryMesh operation and both send the information back to the client and store it in the database (DB) for later use. A summarized workflow of this sequence can be seen in Figure 1.

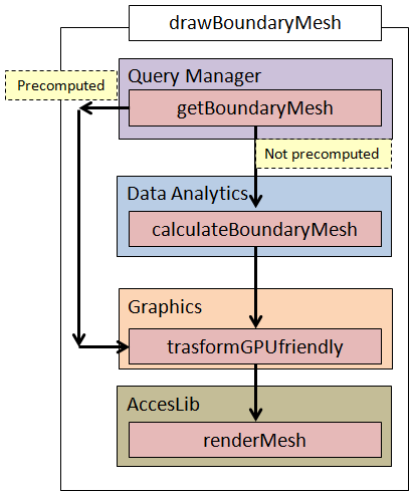

Figure 1. Workflow of the drawBoundaryMesh action issued by the user, and the corresponding getBoundaryMesh VQuery in the platform.

The transformation operations are grouped in the Analytics module of the VELaSSCo platform and are part of the complex queries and the Result Analysis Query family mentioned in D3.4, D3.1 and D2.2.

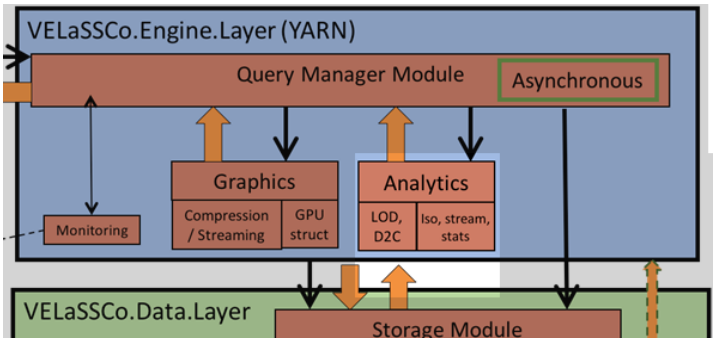

Figure 2. Analytics module and its relation with other VELaSSCo modules.

As already mentioned in D2.4 and D2.5 most of the analytics operations will performed using the MapReduce paradigm commonly used in Hadoop. A scheme of it is depicted in Figure 3.

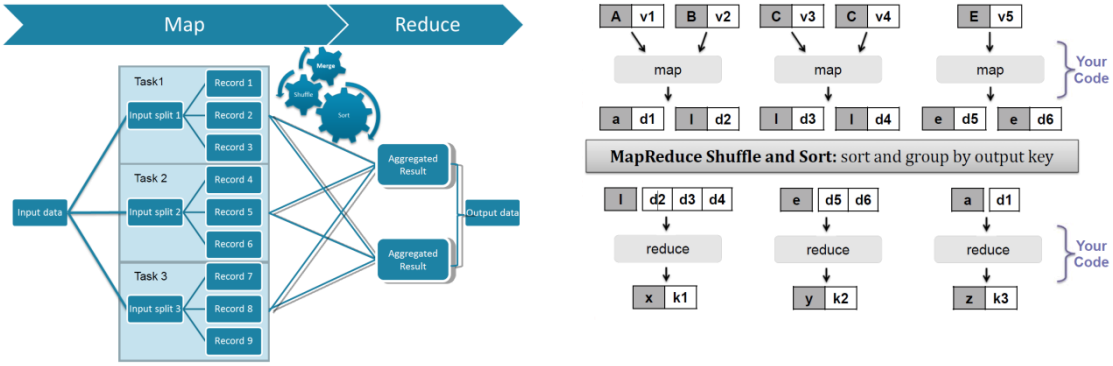

Figure 3. On the left a scheme of the Map Reduce process performed in the Hadoop framework is shown [1]. On the right, the programmable parts of the generic Map Reduce algorithm in Hadoop [2].

The Map Reduce algorithm splits the input data into several chunks. Then, the Map part of the algorithm is applied on each chunk and generates intermediate results as key-value pairs, which are then sorted and grouped by key. These are passed to the Reduce part of the algorithm that merges the information together and outputs the result.

As explained in D2.2, the VELaSSCo Platform has been built considering two scenarios: either the simulation data is stored in HBase tables, or in the EDM database. In the first scenario, the simulation data is stored in HBase tables in several rows (as described in D3.1), one per computing partition. The chunks of data used as input in the Map phase are selected rows/cells of these tables, and the result of the reduce part may be data that needs to be stored in tables too.

For instance, the CalculateBoundingBox as a Map Reduce Job will first select the coordinates' information of the rows corresponding to the user-selected simulation data. In the Map phase partial bounding boxes will be calculated for each input row, which results in several pairs of vertices, and in the reduce phase the partial bounding boxes are merged together to calculate the global bounding box defined by two vertices.

Another way to reduce the information transferred between the platform and the visualization client, besides the simplification and level-of-detail representation of the original simulation model, is to use a high level representation of the model such as using locally refined splines in the VELaSSCo platform. With this approach we can reduce the amount of transferred data by using high degree mathematical representation that approximates the original data.

As mentioned in D2.6 some of these operations may be triggered by the ingestion process in order to already pre-compute the information needed for the GetBoundaryMesh, GetSimplifiedMesh or GetBoundingBox VQueries. We will add more VQueries to this list using a mechanism to detect the most frequently used VQueries. The implementation of this mechanism in the ingestion process is scheduled for the final prototype but a brief explanation is included in this document.

2. Transformations

As mentioned previously, in the Map Reduce scheme there are three parts to be defined for each transformation operation:

- Input: Which data should be selected for the process, that is which column qualifiers from which rows should be passed to the map process
- Map: algorithm that extracts the partial information of the transformation
- Reduce: algorithm that merges together the partial results to generate the global results

Some Analytics transformations may require several MapReduce iterations or one of the two steps. Other analytics operations may be resolved using a Hive query, like the CalculateBoundingBox query that can be resolved using a "minimum, maximum" Hive query over the coordinates. The strategy to be used for each case will be determined after the release of the first prototype, depending on the test results concerning efficiency and performance.

2.1 CalculateBoundaryOfAMesh

Quick overview

Given a Model ID and mesh id, the operation calculates the boundary mesh. If the Mesh ID corresponds to a volume mesh, the operation extracts the surface skin:

- a mesh of triangles for a volume mesh of tetrahedra,
- a mesh of quadrilaterals for a volume mesh of hexahedra, and
- a mesh of both triangles and quadrilaterals for a mesh of prisms or pyramids.

If the mesh ID corresponds to a surface mesh, i.e. a mesh of triangles or quadrilaterals, the operation calculates the boundary edges of this surface mesh, i.e., a mesh of lines. The implementation for the first prototype will support only tetrahedra and triangles.

In the case of dynamic meshes, the boundary mesh is calculated for the mesh id at the time-step specified by the input parameter Time-step value.

The workflow of the operation CalculateBoundaryOfAMesh is depicted in Figure 4.

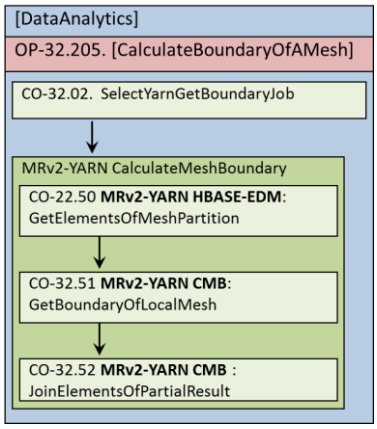

Figure 4. Components of the CalculateBoundaryOfAMesh operation.

Following the MapReduce scheme, the three stages for this operation are:

Input: GetElementsOfMeshPartition – the provided Model ID, mesh ID, analysisID and time-step will be used to select the proper rows in the HBase table for the input data in the MapReduce job.

Map: GetBoundaryOfLocalMesh – in this step the faces of the tetrahedra will be generated, identified by the id's of their vertices and the repeated triangles will be eliminated. The output of this step will be a mesh of unique triangles representing the skin of the local volume mesh, also identified by the id's of their vertices.

Reduce: JoinElementsOfPartialResult – in this step all the triangles (coming from the skin of the local volumes) will be merged together and the repeated ones are eliminated, also using the id's of their vertices. A new mesh will be generated with the unique triangles which represents the global skin mesh.

The same algorithms will be used to calculate the boundary edges of the triangle meshes. In the map phase, line elements will be generated from the triangles and will be identified using the id's of their vertices as well, and the repeated lines eliminated. In the reduce phase, all local boundary edges will be merged together and the repeated lines eliminated, generating a mesh of unique lines representing the boundary edges of the input triangle mesh.

CalculateBoundaryOfAMesh Job Execution and parameter handling

The following instruction is to run the CalculateBoundaryOfAMesh MapReduce job on YARN:

```
$ bin/hadoop jar $HOME/common/hadoop/share/hadoop/yarn/ CalculateBoundaryOfAMesh.jar
ModelID "MeshID" ["AnalysisID" TimeStepValue]
```

ModelID - Model ID (Hex String): the ID of the model selected by the user, identifying the simulated data for a specific model. It is a String that contains the hexadecimal representation of the 16 byte Model ID used in the VELaSSCo platform.

MeshID - Mesh id (String): is the name of the mesh for which we want to calculate a boundary mesh. The name should be provided between quotes.

In the case of dynamic meshes, i.e., a mesh is provided for each time-step of the analysis of the simulation, following parameters should also be provided:

AnalysisID - Analysis id (String): is the name of the analysis where the MeshID is present. The name should be provided between quotes.

TimeStepValue - value (double): is the value of the time-step of the analysis where the MeshID is present.

CalculateBoundaryOfAMesh Job Input and Output

The CalculateBoundaryOfAMesh Job will always produce a String as an output that includes information about the computation outcome. In case of Error the Job will fail and a String with the nature of the problem will be sent to the Query Manager.

On the first iteration the query will return a mesh with the list of coordinates and the list of elements to the query manager. Later on the resulting mesh will be stored in the "Simulations_VQuery_Results_data" and "Simulations_VQuery_Results_metadata" table with a composed rowkey that follows the combination ModelID + AnalysisName + TimeStep.

2.2 Discrete to Continuum transformation

Quick Overview

The discrete to continuum (D2C) transformation consists on spatial averaging methodology of DEM simulation data related to the particles and contacts. By means of this methodology, the new bulk properties are computed from the discrete DEM data and projected into a continuum field using a static mesh. The result is a mesh with some interpolated properties, similar to FEM simulation data, e.g.,

- Density field.
- Momentum field.
- Velocity field.
- Contact Stress field.
- Kinetic stress field.

Moreover, this VQuery includes the possibility to compute temporal averaging of the D2C results.

Since the bulk properties computed by the D2C transformation are associated to a static mesh, this new data can be visualized and analysed in a similar way to FEM simulation data by means of VELaSSCo queries.

The first version of the D2C transformation implementation is based on the Map Reduce paradigm. The parallelism is based on a single time-step, where each simulation time step is computed independently and in parallel by each map. For this reason, for this first algorithm version, the data for each time step must be stored in a single node.

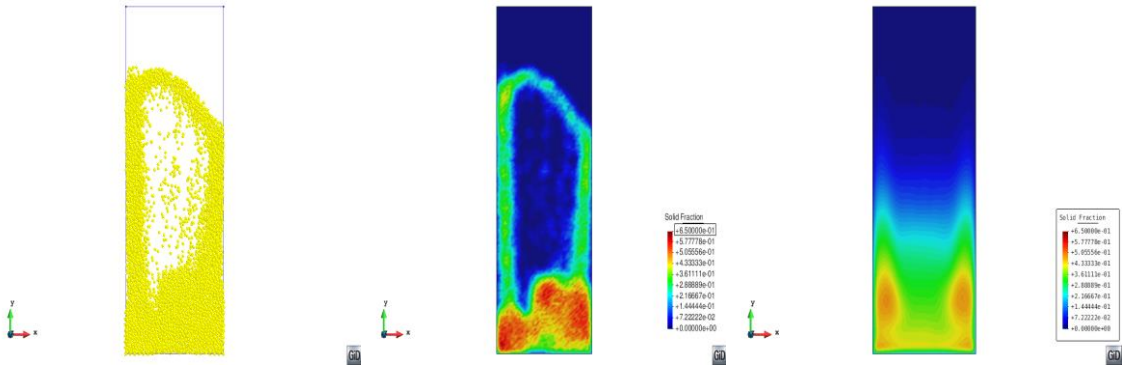

Figure 5. Example of the results obtained by means of the Discrete to Continuum transformation: Left, discrete data provided by DEM simulation solver. Center: D2C results for a single time-step. Right: D2C results including temporal averaging.

D2C Job Execution and parameters handling

The following instruction is to run the D2C MapReduce job on YARN:

```
$ bin/hadoop jar /localfs/home/velassco/common/hadoop/share/hadoop/yarn/D2C.jar p1 p2 p3
p4 p5 p5 p7 p8 p9 p10 p11 p12 p13
```

where `P = {p1..p12}` is the set of parameters required to run the D2C Job to be defined as follows:

p1 - Simulation id (Hex String): the simulation_ID of the DEM model which is present in the HBase table and taken into account to compute the D2C transformation. It is a String that contains the MD5 cryptographic hash function output of a given simulation.
Example for this parameter: fc49336494e0b1c72865a0e6617f4112.

p2 - Analysis Name (String): the analysis name for storing the results of the D2C analysis. It has to contain less than 9 characters. The same analysis name associated to the same D2C static mesh (identified as a simulation id in the HBase tables) is not allowed. At the end of the computation, the results of D2C will be stored in the HBase tables with the rowkey combination Simulation_ID' + Analysis_Name + Timestep + Partition where the Simulation_ID' corresponds to the Simulation_ID of the static mesh provided by parameter **p3**.
Example for this parameter: D2C_TEST

p3 - Static Mesh id (long): corresponds to a Simulation_ID of a VELaSSCo model that contains the data of a pre-generated static mesh which is already stored in the HBase table. It is a String that contains the MD5 cryptographic hash function output of a given static mesh. The results of the D2C will be associated to the vertices/nodes of this static mesh.
Example for this parameter: bdea3fde4c5f2a6ff51e84e9aac0247a.

p4 - Time-step options (String): this parameter allows the user to define different options for selecting the time-steps of the DEM model to be considered in the computation:

- "ALL": all time-steps of the DEM model are considered in the calculation.
- "SINGLE": only a single time-step is considered in the calculation.
- "INTERVAL": all the time-steps within the interval defined by the user are considered for the calculation.

Example for this parameter: "INTERVAL".

p5 - Time-steps (double or double,double): Depending on the **p4** value this parameter considers a double or two comma separated doubles. In any case this parameter has to be specified.

- If **p4** is equal to "ALL": any value put here is ignored.
- If **p4** equals to "SINGLE": only a single double value has to be provided. This value corresponds to the time-step for which the D2C calculation needs to be computed.
- If **p4** is equal to "INTERVAL": two doubles have to be provided separated by comma. The two values define an interval so that all time-steps of the DEM model (specified by parameter p1) within the interval are considered for the calculation. The first value has to be always lower or equal to the second one.

Example for this parameter: 2799000,99999999.

p6 - Coarse_graining_method (String): Two different methods can be selected:

- "Heaviside" : a Heaviside function is used to conduct the spatial averaging,
- "Gaussian": a Gaussian function is used to conduct the spatial averaging.

Example for this parameter: Gaussian

p7 - Coarse_graining_width_option (double): This parameter defines the length scale for the spatial averaging to be computed:

- In the case of "Heaviside" coarse graining method, the value of the parameter corresponds to the radius of the function.
- In the case of "Gaussian" coarse graining method, the value of the parameter corresponds to the standard deviation of the Gaussian function.

Example for this parameter: 0.00024

p8 - Coarse_graining_Cutoff_factor (double): the value of the parameter defines the maximum distance from the coarse-graining nodes (vertices of the static mesh) to the particle centers and contact point positions that will be considered in the calculation of the spatial averaging.

- If **p6** value is set to "Heaviside": this value will be ignored since the maximum distance is already defined by parameter **p7**. In any case this parameter needs to be specified.
- If **p6** value is set to "Gaussian": the maximum distance for spatial averaging is equal to **p7*p8**.

Example for this parameter: 3.0

p9 - Process_contacts (boolean): This parameter allows the user to choose if only the DEM particle data needs to the considered in the D2C calculation or if he/she also wants to include the DEM contact data:

- **p9** == *True*, DEM particle and contact data are considered in the calculation.
- **p9** == *False*, only DEM particle data are processed.

Example for this parameter: *True*

p10 - Do_temporal_averaging (boolean): this parameter allows the user to specify if temporal averaging of the D2C results needs to be computed. If the value is set to true the temporal averaging is computed.
Example for this parameter: *True*

p11 - Temporal_averaging_options (String): in case that the value of parameter **p10** is set to "true", the user can choose between two different temporal averaging options:

- "ALL": temporal averaging will be conducted by considering all the pre-filtered time-steps (**p4** and **p5** will filter the time-steps to be considered).
- "TEMPORALWINDOW ": the temporal averaging will be conducted for the pre-filtered time-steps within the width of the temporal window specified by parameter **p12**.

Example for this parameter: TEMPORALWINDOW

p12 – DeltaT (double): width of the temporal window for the temporal averaging. The results belonging to the pre-filtered time-steps inside the same temporal windows are averaged. This value has to be inserted in any case and if **p11** is equals to "ALL", it will be ignored.
Example for this parameter: 20000

p13 – Prefix value (String): this parameter is optional and has to be specified if and only if we want to run the Job in test mode. In particular, the parameter represents the prefix of the test tables to be used.
Example for this parameter: "Test_" in order to use the Test_Simulations_Data and Test_Simulations_Metadata tables.

Further optional parameters can be also included after **p12 or p13**, like the number of mappers or reducers to use for the computation. For example by adding "-D mapred.map.tasks=5" and "-D mapred.reduce.tasks=2" in the command line, the Job is processed by using 5 mappers and 2 reducers.

A specific example of a D2C MapReduce Job that can be run is:

```
$ ./bin/hadoop jar
/localfs/home/velassco/VELASSCO/trunk/modules/EngineLayer/D2C_of_a_model.jar
57f8f7237f3f024604c4f0b0f00851f0 D2C_test 3ce674c37cf7115ecddab84aec43bff0 INTERVAL
2799000,99999999 Gaussian 0.0024 3.0 true true TEMPORALWINDOW 20000 T_
```

This example runs the D2C query for a DEM model of VELaSSCo platform with Simulation_ID = 57f8f7237f3f024604c4f0b0f00851f0 by using the static mesh already present in the VELaSSCo platform with Simulation_ID' =

3ce674c37cf7115ecddab84aec43bff0. The spatial averaging will be conducted for both DEM particle and contact data by using a Gaussian function with standard deviation equal to 0.0024 and a maximum distance or cut-off equal to 3.0*0.0024. This calculation is conducted for all time-steps of the DEM model within the range [2799000, 99999999]. Moreover, for the time-steps within the specified interval, temporal averaging will be computed using a temporal window with a width equal to 20000.

An example of a running D2C job can be seen in Figure 6.

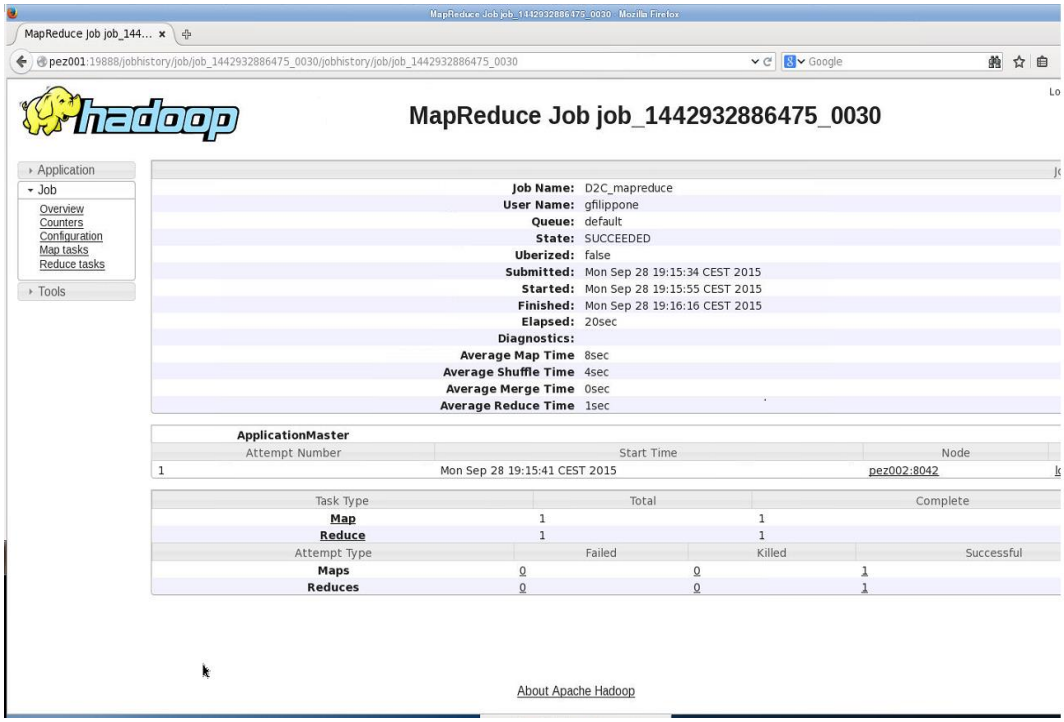

Figure 6. Monitoring the progress of the D2C map-reduce job.

D2C Job Input and Output

The D2C Job will always produce a String as an output that includes information about the computation outcome. In case of Error the Job will fail and a String with the nature of the problem will be sent to the Query Manager. If the Job succeeds, it stores the results of D2C calculation in the “Simulations_data” and “Simulations_metadata” tables with a composed rowkey that follows the combination Simulation_ID’ + Analysis_Name + timestep+partition. Note that the Simulation_ID’ is the input parameter “static mesh id” that corresponds to the Simulation_ID of the static mesh already stored in the VELaSSCo platform.

The Job will send back to the Query Manager a String containing a confirmation message of Success together with the Rowkey prefix (Simulation_ID’+Analysis_Name) where results have been stored.

For example, by considering the Job shown previously, if it successfully terminates, the results of the D2C will be stored with the row-keys

"bdea3fde4c5f2a6ff51e84e9aac0247a_test2800000", …, "bdea3fde4c5f2a6ff51e84e9aac0247a_test99989999" and the string message "Query completed successfully, Results are stored in the Data Layer! Model id: bdea3fde4c5f2a6ff51e84e9aac0247a Model Analysis:D2C_test " will be sent to the Query Manager.

If the Job fails (because, for example, the Simulation_ID does not exist) the query will send to the Query Manager the following message "ERROR #4: THE SPECIFIED SIMULATION ID DOES NOT EXIST".

What follows is the full list of the possible handled errors by this first algorithm version:

ERROR #1: THE TIME STEP " + Long.parseLong(args[4]) + "IS NOT PRESENT IN THE SIMULATION WITH ID " + Long.parseLong(args[0]));

ERROR #2: THE TIME STEP INTERVAL HAS NOT BEEN WELL SPECIFIED (TIMESTEP[0] GREATER THAN TIMESTEP[1])");

ERROR #3: THERE IS NO TIME STEP WITHIN THE SPECIFIED INTERVAL

ERROR #4: THE SPECIFIED SIMULATION ID DOES NOT EXIST

ERROR #5: THE SPECIFIED STATIC MESH ID DOES NOT EXIST

ERROR #6: IT IS NOT ALLOWED TO USE 'DEM' AS AN ANALYSIS NAME: NAME RESERVED TO SIMULATION MODELS

ERROR #7: IT IS NOT ALLOWED TO USE ' ' AS AN ANALYSIS NAME: NAME RESERVED TO STATIC MESHES

ERROR #8: ANALYSIS NAME ALREADY USED FOR THIS SIMULATION

ERROR #9: The number of parameters adopted is wrong

("Usage of D2C computation version 1.0:");

("args[0] = (Hex String) Simulation GUID //ID of the Simulation used for the post-processing computation");

("args[1] = (String) Analysis Name //Analysis of the Simulation used for the post-processing computation");

("args[2] = (Hex String) Static Mesh GUID //ID of the Static Mesh used to compute the final results. If the mesh is not present will be generated");

("args[3] = (String) Time-step options // key word defining 3 different cases: "ALL", "SINGLE", "INTERVAL"");

("args[4] = (List of Double) Time-steps // It depends on the args[3]: case ALL:{any value will be ignored}, case SINGLE:{only one double value}, case INTERVAL {two doubles in this form value1,value2}");

("args[5] = (String) Coarse_graining_method: Heaviside or Gaussian");

("args[6] = (Double) Coarse_graining_option_1: Width");

("args[7] = (Double) Coarse_graining_option_2: Cutoff factor (if method equals to Heaviside this values will be ignored)");

("args[8] = (Boolean) Process_contacts: if false only particle information is processed");

("args[9] = (Boolean) Do_temporal_averaging");

("args[10] = (Boolean) temporal_averaging_options //key word defining 2 different cases: "ALL", "TEMPORAL WINDOW"");

("args[11] = (Double) DeltaT: It dependes on the args[10]: case ALL:{any value will be ignored}, case TEMPORAL WINDOW:{only one double value}");

("args[12](Optional) = (String) This parameter is used for testing mode and is the prefix of the test tables used: e.g. \"Test_\"Simulations_Data and \"Test_\"Simulations_Metadata.")

2.3 CalculateSimplifiedMesh

Quick Overview

This operation is based on the MapReduce version of the vertex clustering method used by Vo et al. [3] and Pasenau [4,5], and is adapted for volume meshes and taking advantage on the fact that the simulation data has been already split by the simulation program into several computing domains.

Without going too deep into the specifics of the simplification method, the algorithm basically groups vertices of the original mesh into cells of a regular grid that encloses the model, accumulation information about the original mesh in the process. Then, for each group of vertices an optimal representative is calculated and used to substitute all of the original mesh grouped in that cell. All collapsed elements of the original mesh are discarded and only remains the elements of the simplified mesh.

The process requires two map-reduce jobs:

- **1st job**:
  - **Map**: for each element, calculates geometric information, calculates the id of the cells that the vertices of the element falls into, and outputs as many pairs as vertices the element have, each pair with the id of the cell and the calculated information.
  - **Reduce**: merges the calculated information of the pairs with same cell ID, computes the optimal representative for that cell, and outputs the new vertices for the simplified mesh.
- **2nd job**:
  - **Map**: for each element, calculates the id of the cells the vertices of the element falls into; if the new vertices ID's are unique then the triangle is passed to the reducer.
  - **Reduce**: emits the new simplified elements.

CalculateSimplfiedMesh Job Execution and parameters handling

The following instruction is to run the CalculateSimplfiedMesh MapReduce job on YARN:

```
$ bin/hadoop jar $HOME/common/hadoop/share/hadoop/yarn/ CalculateBoundaryOfAMesh.jar
GridSize ModelID "MeshID" ["AnalysisID" TimeStepValue]
```

GridSize – number of cells of the decimation grid (integer): the number of cells in one axis used to define the decimation grid used to group the vertices in cells, e.g. with 256 as input, the algorithm will use a regular decimation grid of 256^3 cells to enclose the model.

ModelID - Model id (Hex String): the ID of the model selected by the user and used in VELaSSCo to identify the simulated data for a specific model. It is a String that contains the hexadecimal representation of the 16 byte Model ID used in the VELaSSco platform.

MeshID - Mesh id (String): the name of the mesh from which we want to simplify. The name should be provided between quotes.

In the case of dynamic meshes, i.e., a mesh is provided for each time-step of the analysis of the simulation, following parameters should also be provided:

AnalysisID - Analysis id (String): the name of the analysis where the MeshID is present from which we want to simplify. The name should be provided between quotes.

TimeStepValue - value (double): the value of the time-step of the analysis where the MeshID is present for which we want to calculate a boundary mesh.

CalculateSimplfiedMesh Job Input and Output

The CalculateSimplfiedMesh Job will always produce a String as an output that includes information about the computation outcome. In case of Error the Job will fail and a String with the nature of the problem will be sent to the Query Manager.

On the first iteration the query will return a mesh with the list of coordinates and the list of elements to the query manager. Later on the resulting mesh will be stored in the "Simulations_VQuery_Results_data" and "Simulations_VQuery_Results_metadata" table with a composed rowkey that follows the combination ModelID + AnalysisName + TimeStep. Also interpolation information will be stored together with the simplified mesh in order to interpolate the results values for the simplified mesh from the original result values.

3. Locally refined splines in the VELaSSCo Platform

In continuum-based methods, a novel trend in Finite Element Analysis (FEA) is isogeometric analysis (IGA), where traditional FEA elements are replaced by spline based higher degree elements that provide a much more accurate and compact data representation of scalar and vector fields.

Analysed as a strategy for numerical simulations, IGA has some advantages and drawbacks compared with FEM, but in the frame of VELaSSCo we won't go into detail on these aspects. On one hand, it is interesting to include splines in the VELaSSCo platform, to explore (in the near future) possibilities to get data in the VELaSSCo platform coming from IGA solvers. On the other hand, VELaSSCo Platform can take advantage on splines technology in order to produce more smooth visualizations and provide a more efficient treatment of the simulation data considering memory consumption.

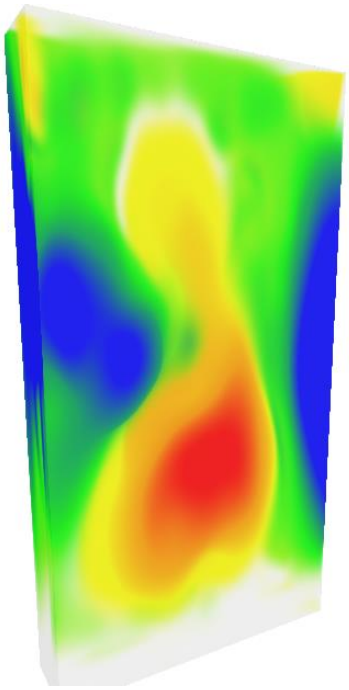

Figure 7: LRB-spline approximation of the fluidised bed use case.

In VELaSSCo, Locally Refined (LR)B-splines will be used as a new class of data analytics for FEA and DEM. For fields with an overall smooth behaviour, we expect a significant reduction in data volume. We will also compare spline-based discrete to continuous transformations to the traditional methods employed in DEM (for an early example, see the Figure 7).

Approximation and visualization of volumetric LRB-spline scalar and vector fields from FEA and DEM data will be complex queries implemented in the final prototype of the VELaSSCo platform. The approximations will be run as batch processes, and the results stored in the VELaSSCo infrastructure. The visualizations will be made available

through the visualization client (in the frame of VELaSSCo project: GiD or iFX frontends).

The specific features of the platform which may take advantage on the use of LRB-splines can be very wide, but it is not clear to know a priori which ones can improve the performance of the platform compared with the standard FEM data. Through discussion with the domain experts in the project, we have determined the core visualization functionalities that will be implemented:

1. Direct visualizations from volumetric LRB-spline fields, such as
    a. Transparent volumes (see Figure 7)
    b. Cut planes
    c. Iso-surfaces
2. Visualization of pre-computed
    a. Cut planes
    b. Streamlines
    c. Particle traces

All these cases include the visualization of the geometry involved (the plane result from the cut, for instance), and also the results attached to it (as a contour fill, for instance), considering the result can be scalar or a vector.

Time and resources permitting, additional functionalities may include:

3. Spatial averaging of volumetric LRB-spline fields
4. Direct visualization of fields such as stress and momentum
5. Direct visualization of streamlines
6. Approximation of 4D spline fields (3 spatial + 1 time dimension)
7. Dynamic time averaging from 4D LRB-splines

4. Triggered queries

As already mentioned some analytics operations are computed when the simulation data is ingested into the VELaSSCo platform, so that their results are ready for when the user connects to the platform. This trigger mechanism will be part of the final prototype.
A mechanism to detect the most used queries will also be part of the final prototype, adding more queries to be pre-computed.

4.1 Triggered queries from Flume/HBase

The idea behind triggering queries from Data Injection Module (specifically from Flume/Hbase) is to precompute some VQueries that always will be executed, and other which we know a priori will be executed very often.

When the Injection Process finishes, it will execute/pre-compute the VQueries so that the results of these VQueries are present when the user connects to the platform.

Currently the Data Injector uses a set of Flume Agents in charge of population into the a data model defined in Hbase that covers the DEM and FEM simulation formats. An Hbase Synchronizer is defined as part of the each Flume Agent, managing all the actions to be executed over Hbase, i.e., a “Put” action by each of the events coming from the agent or pipeline.

Considering this approach, the triggering mechanism could be based on the use of an ad-hoc Hbase Coprocessor. A Coprocessor is a framework that provides an easy way to run your custom code on a Region Server. We can establish an analogy with Triggers and Stored Procedures that are provided on traditional relational databases. An Observer Coprocessor is compared to triggers because like triggers they execute a custom code when certain event occurs (like Get or Put). Similarly, an Endpoint Coprocessor is compared to the stored procedures and it can be performed custom computation on data directly inside the region server.

Our triggering mechanism will be based on an Observer Coprocessor allowing our custom code to run just after the ‘Put’ operation. All methods providing this feature will start with the prefix ‘post.’ As we want to our code to be executed after the put operation then we should override following method of RegionObserver class:

```
public void postPut (final ObserverContext e, final Put put,
final WALEdit edit, final Durability durability) throws IOException {}
```

Trigger mechanisms will be included in the final version of the platform.

4.2 Detecting most used queries and data access patterns

The results of this deliverable will be pursued further in task 3.7, "Detection/identification of specific data patterns and other novel techniques". The

motivations for this task are the challenges that we will meet in applying the so far developed queries against big data sets.

For the closed source option of the VELaSSCo platform using the EDM database, Jotne will pursue the following paths for improving query performance:

1. General improvements of query / search performance in the Express Data Manager (EDM) database server. A two-step improvement will be tried out:
   1) Map the database files directly into the logical memory of the database server. By that one will use more hardware support when retrieving objects from the database which will improve performance.
   2) We will make the database objects access methods thread safe, which implies that EDM can utilize all the cores in a modern database server.
2. For some queries it can be worthwhile to execute them whenever the database is idle and even without a user induced event, and to store the query result for later retrieval. By that the server can deliver a pre-computed result quickly when a user actually requests the query. Finding boundary of a mesh is a typical example of a query where pre-computation is worthwhile.
3. For other queries the variation of the input parameters is so big that pre-calculation is not a good idea. For example finding the tetrahedron that contains a random point is such a query. In this case we will improve performance by building assisting data structures that support the search / computation. Such domain specific adaptations of queries will be derived from analyses of query specific data patterns and from the frequency of requests of such queries. Understanding data-access patterns for fast-data access requires knowledge of the domain, in the case of VELaSSCo, this is competence in FEM and DEM.

5. References

[1] From "MapReduce/Hadoop in a nutshell" http://wiki.opf-labs.org/download/attachments/29392905/FFFF-IntroPresAtScale-ONB.pdf

[2] From "MapReduce overview" http://courses.coreservlets.com/Course-Materials/pdf/hadoop/04-MapRed-1-OverviewAndInstall.pdf

[3] H.T. Vo, J. Bronson, B. Summa, J.L. Comba, J. Freire, B. Howe, V. Pascucci, and C.T. Silva, "Parallel Visualization on Large Clusters Using Mapreduce," Proc. IEEE Symp. Large-Scale Data Analysis and Visualization, pp. 81-88, Oct. 2011, DOI 10.1109/LDAV.2011.6092321.

[4] Miguel Adolfo Pasenau de Riera. "Detail-preserving mesh simplification". Master thesis. Department of Computer Science, Barcelona School of Informatics, Universitat Politècnica de Catalunya, UPC, Barcelona Tech, Spain, Sept. 2013. url: http://upcommons.upc.edu/pfc/handle/2099.1/20380.

[5] Miguel A. Pasenau and Carlos Andújar. Detail-preserving mesh simplification for scientific visualization. Tech. rep. Presented as conference paper. July 2014. url: http://www.wccm-eccm-ecfd2014.org/admin/files/fileabstract/a524.pdf.