



VELaSSCo

*Visual Analysis for **E**xtrêmement **L**arge-**S**cale
Scientific **C**omputing*

**D2.7 – Target engineering friendly and open standards
formatted data stored in the HPC cloud infrastructure**

Version #1.0

Deliverable Information

Grant Agreement no	619439
Web Site	http://www.velassco.eu/
Related WP & Task:	WP2, T 2.6
Due date	December 31 st , 2015
Dissemination Level	
Nature	
Author/s	
Contributors	

Approvals

	Name	Institution	Date	OK
Author		FRAUNHOFER/INRIA		
Task Leader	Frank Michel	FRAUNHOFER		
WP Leader	Bruno Raffin	INRIA		
Coordinator	Abel Coll	CIMNE		
Contributors	Iván Cores	INRIA		

Change Log

Version	Description of Change
0.1	Outline and initial content
0.2	Add main content + cleaning
1.0	Review and definitive version

Table of contents

1	Introduction.....	4
2	Hbase Data formats.....	6
2.1	VELaSSCo_Models.....	7
2.1.1	Simulation_Metadata.....	7
2.1.2	Simulation_Data.....	8
3	Conclusion.....	9
4	References.....	9
5	Annex I – Hbase table structure.....	10
5.1	VELaSSCo_Models.....	10
5.2	Simulations_Metadata: <i>metadata of Simulation models</i>	11
5.3	Simulations_Data: <i>meshes and results data of simulation models</i>	17
5.4	Simulations_VQuery_Results_Metadata: <i>metadata of the Vqueries</i>	21
5.5	Simulations_VQuery_Results_Data: <i>data of the Vqueries</i>	21
5.6	Standard Gauss Points definition.....	21

1 Introduction

D2.7 was expected (DOW) to contain refinements for the data conversion identified in D2.6. As we did not identify significant divergences from D2.6, we focus D.2.7 on the description of the actual table structures adopted for storing the data and metadata in the VELaSSCo parallel storage (DEM or FEM data sets). Previous descriptions made in D2.6 (section 5.2) and D3.1 (section 3.3.1) are outdated.

VELaSSCo's goal is to enable the high performance analysis of the data produced by parallel FEM and/or DEM simulations, relying on a Hadoop infrastructure. The way the data produced by these simulations is structured in the Hadoop infrastructure is critical to ensure that data queries can be efficiently implemented, taking advantage from the data parallel data processing capabilities offered by Hadoop. The data structure in Hadoop must take into consideration the way the data are organised by the simulation. Before presenting the structure of the VELaSSCo Hbase tables, we rapidly remind general key features related to numerical simulation data structures.

The parallel simulations like the ones considered in VELaSSCo often relies on a distributed memory model, following the MPI (Message Passing Interface) programming model, where the simulation data/space needs to be explicitly partitioned across the processes running on the cluster. This partitioning follow the simulation own internal optimization criteria, but with the usual goals of ensuring a even load balancing amongst processes while minimizing the needs for communications. Such partitioning often follows a principle of spatial coherency, i.e. data spatially close in the simulation space tend to be located in the same partition.

To simplify the writing of the results of the parallel simulation program and to reduce data communications, data at the boundary of several simulation partitions are often duplicated amongst these partitions. For instance, in a FEM simulation, a tetrahedron, and its vertices, located on a partition boundary will be duplicated on each neighbouring partition to avoid having partial definitions of tetrahedrons. Numerical simulations are iterative processes that progress in time with a given time-step. At a given frequency defined by the user, results, i.e. particles' positions and states for DEM simulations or elements' positions and state for FEM simulations, are exported.

In a traditional HPC simulation data, different time steps are often saved to different files, while the internal data partitions are often merged, suppressing the duplicated data, but also losing information about a potential data split that could convey a relevant data storage parallelism.

One fundamental aspect of map/reduce infrastructure is the structuring of data into tables of rows indexed by keys (first column) and having data attached in one or several extra columns. These tables are split into blocks. In Hadoop, tables are splits in physical "blocks", usually of 128MB, and logically into "splits". Boundaries of logical splits are not necessarily aligned with blocks. Blocks are managed by the low level HDFS file system, while Hbase exposes splits to the Hadoop infrastructure. Blocks are distributed on different cluster nodes and can be replicated to ensure data integrity

and to support fault tolerance mechanisms. The split size defines the granularity of data parallelism. One map instance processes full splits. Several map instances can process in parallel several splits, while one split cannot be processed concurrently by several maps instances. The actual acceleration provided by parallel splits processing depends on the physical location of the data. Notice that in Hadoop tables' rows are sorted according to the key index, the row-key.

Analysis of data from numerical simulations are often performed on a given timestep (iso-surface extraction, plane cut, streamlines...) or a series of time-steps (path-lines, results' statistics, discrete to continuum transformations, etc.). Thus, it is important to rely not only on the temporal parallelism offered by the various time-steps, but also by a parallelism internal to each time-step. For that purpose we leverage the partitioning performed when running the simulation, optimized for load balancing and reducing the needs for communications between partitions, into the Htables.

Because the simulation often runs at a significantly higher degree of parallelism than data analysis, the number of partitions available at the simulation level offers a potential parallelism that should, in most cases, enable to benefit from the resource aggregation provided by the Hadoop cluster (smaller in terms of nodes). Thus each partition for each time-step corresponds to a row. The indexing key is a built from the simulation id, the simulation analysis name, next the time-step value and eventually the partition number. This order is important as rows are ordered based on that index.

The goal is to have rows likely to be processed together stored closely one to each other to benefit from the caching effect of block loading during the hbase scan process. Because we keep the partitioning from the simulation, processing the data from one row becomes easier as a partition is usually self-contained.

On the opposite, this partitioning is simulation dependent, so queries must be aware of the partitioning scheme used. To mitigate this effect the data produced by a simulation are converted to the AP2019 standard format before to be injected into the EDM. For a detailed description of AP2019 please refer to Deliverable D2.6, where it was already described exhaustively. We detail in the following the Htable data structure adopted for VELaSSCo open source architecture.

Early versions have been exposed in deliverables D2.6 and D3.1. The one presented here is the one adopted for the current prototype. Notice that the result of popular queries, like mesh boundary extraction, can be stored as new datasets in the Hadoop cluster to save processing time and have a more reactive system. The scheme for storing these new data sets follows the same than the simulation provided data with small additions. We detail it here as well, but it may be subject to changes as this aspect of the VELaSSCo prototype is still under intensive development.

2 Hbase Data formats

HBase is a kind of Big Table, which enables access to data sets in a very efficient way. The access to this table is based on a key methodology. For each row a key is computed. Apache HBase¹, the open source, NoSQL standard for Apache Hadoop, is one of the two databases chosen to store engineering simulation data in a persistence layer. According to a Cloudera post², the main benefits on using Apache Hbase as NoSQL database are the following:

“Tightly integration with HDFS, HBase opens data stored in Hadoop to users across the company requiring real-time data access. As an integrated part of Hadoop, HBase also works closely with other popular, open standard tools such as Apache Kafka, Apache Flume, Apache Spark, and Impala.”

HBase can be used on top of multiple file systems, and in particular with the HDFS distributed file system from the Hadoop suite. Besides this, NoSQL capacities provide enough flexibility to adapt engineering simulation data to the most suitable data model, in order to optimize data access to retrieve simulation information.

HBase data model consists in following elements³:

- *Table*: design-time namespace, has many rows.
- *Row*: atomic key/value container, with one row key
- *Column Family*: divide columns into physical files
- *Column*: a key in the k/v container inside a row
- *Timestamp*: long milliseconds, sorted descending
- *Value*: a time-versioned value in the k/v container

The data types of these elements are:

- Row keys, column names, values: arbitrary bytes
- Table and column family names: printable characters
- Timestamps: long integers

In the VELaSSCo platform there are six global tables:

- *VELaSSCo_Models*: one with a listing of the simulation models and some properties and for each simulation model,
- *Simulations_Metadata*: a Metadata table with some specific metadata of the model,
- *Simulations_Data*: a Simulation Data table with the mesh(es) and results calculated by the simulation program,
- *Simulations_VQuery_Results_Metadata*: as *Simulations_Metadata* this table will store the metadata of the output of the VQueries,
- *Simulations_VQuery_Results_Data*: as *Simulations_Data* this table will store the data of the output of the VQueries, and
- *VELaSSCo_Users*: to manage permissions to execute / access the queries.

Simulations_Vquery_... stores the output of the VQueries, like the *GetBoundaryOfMesh*, *GetSimplifiedModel* or *SplineVolume* representation, or

temporary information for Isosurfaces or Cutplanes, that can be later reused, like cut extended information needed to interpolate results for the cut.

In the case of the Discrete2Continuum transformation, which creates a FEM mesh with new analyses, time-steps and results, the Vquery will inject the data in tables: the Simulations_Metadata and Simulations_Data.

Eventually, data about the input parameters used to run the Vqueries could be stored in the CF:Q of Simulations_VQuery_Results_Metadata.

For a better operation of the flume agents when injecting the simulation data into the VELaSSCo platform, instead of storing the metadata and simulation results in independent tables for each simulation model, its metadata and data will be stored in two global tables: Simulations_Metadata and Simulations_Data. To access the data of a simulation, its SimulationId will also be used as key.

Modelling data schema must take into account main functionalities required by the platform, as well as optimizing access to stored simulation data. Next sub-sections will describe the main three tables: VELaSSCo_Models, Simulations_Metadata and Simulations_data:

2.1 VELaSSCo_Models

This is the global table, which contains general information about the simulation processed and stored. Data is stored in a unique Column Family called “Properties”, which also contains specific qualifiers for simulation name, user, file path, etc. The main role of this table is offering a general overview of engineering simulation data stored, types, path, names, etc. to provide input parameters to build complex queries.

2.1.1 Simulation_Metadata

This table contains the data description (metadata) of the simulation meshes and results present in the Simulations_data table for all models in the VELaSSCo platform. The main information stored here related to types of particles, types of results, etc. Data is divided into three column families; M, represents Mesh data, G represents GaussPoints and R refers to results. The *SimulationID* will be used to access the metadata of the simulation model with id *SimulationID*.

For instance, a FEM simulation model may have different meshes with triangles, tetrahedrons and lines with names, unit and color properties. The meshes are grouped by name and element type (all triangles with same name together, etc.) and this table will contain the name, color, element type and units and the Simulations_Data will contain the list of triangles with ids and connectivities. The same for a velocity vector, the Simulations_Metadata table will contain the name of the result, the type, in this case a vector, the number of components, eventually the component names, and other properties and the Simulations_Data will contain the list of result values with their id's.

For a single SimulationModel there will be several entries in this table, one for each AnalysisName and Step inside this analysis, for instance to represent that a fluid is only

present at some time-steps of the simulation. The entries will contain the mesh description, gauss points description and results description for that step of that analysis for simulation problems with dynamic meshes, like particle simulations. For simulations problems with static meshes, like a fluid flow simulation without refinement, their mesh and gauss points description will be on an extra entry with an empty AnalysisName, and a step value of 0. The entries corresponding to each step of analyses will have empty mesh and gauss points descriptions, but result definitions will be present.

Traditional FEM simulation models have a single Coordinate set and several Meshes, representing several parts (layers) of the model.

For DEM simulations, to represent the particles and contacts as described in the P4 format, there will be more than one Coordinate set:

- *Coordinates-p3p* with number of particles (as appearing in P3P simulation files).
- *Coordinates-p3c* with number of particle2particle contacts (as appearing in P3C files).
- *Coordinates-p3w* with number of particles2wall contacts (as appearing in P3W files).
- *Coordinates-w* with number of nodes of wall meshes (considering all the wall meshes together).

Also for DEM simulations, as described in the P4 format, there will be always at least 4 meshes:

- Mesh-p3p (for particles, maybe additional meshes needs to be considered once the complex shaped particles are included, for instance a mesh for spherical particles and another one for complex shaped particles? It will depend on how visualization of complex shaped particle is implemented ... For the analytics all particles could be in the same mesh)
- Mesh-p3c (for p2p contacts with element type “quadratic line”).
- Mesh-p3w (for p2w contacts with element type “line”).
- Mesh-walls (for the mesh of the walls. They are like the FEM meshes).

2.1.2 Simulation_Data

Finally, this is the most important table, where simulation result data is stored. Similarly, information is divided into two column families: M for Mesh and R for Results. It contains different types of results like, scalar, vectors, etc. Qualifiers are defined with a unique name in order to identify specific information that could be retrieved for queries.

This table contains the data of coordinates, meshes and results for all simulation models. Following the recommendations for HBase schema design, the table will have only two column families with several column names (qualifiers). The data will be stored in binary form, to avoid binary-to-string conversions when storing and string-to-

binary conversions when querying the data. Each vertex/coordinates, element and result value will be stored separately:

- the first letter indicates if it's a 'c' coordinate, 'm' mesh or 'r' result
- followed by 6 digits that represents the id's of the coordinate set, mesh set or result set respectively,
- In the case of column qualifiers related to meshes 'm' and results 'r', 2 characters.
- followed by 1 special character “_”
- followed by 8 bytes which are the 64-bit-integer of the Id of the vertex, mesh element or result value respectively.

This data model is currently being assessed in terms of functionality and performance, so minor changes could be adopted in the development of the VELaSSCo platform.

A detailed description of these tables can be found in this document Appendix.

3 Conclusion

The HTable structure described in this document reflects the structure actually in use in the current prototype implementation (December 2015). Data computed from queries and that are expected to be saved in the Htable (pre-computed queries) follow the same data structure. However pre-computed queries are still in their early implementation stage. As the implementation and the experiments progress we may expect some changes.

4 References

- 1: <http://hbase.apache.org>
- 2: <http://vision.cloudera.com/open-standards-in-apache-hadoop-apache-hbase/>
- 3: “HBase Schema Design”, HBaseCon 2012, by Ian Varley.

5 Annex I – Hbase table structure.

Next sub-sections shows all the internal parameters and columns in each HBase table. VELaSSCo_Users table is not depicted in this section because it is only used to manage permissions to execute/access the queries.

5.1 VELaSSCo_Models.

The main role of this table is offering a general overview of engineering simulation data stored, types, path, names, etc. to provide input parameters to build complex queries.

This table contains a list of the simulation models present in the VELaSSCo platform with:

- **Key:** *SimulationId*: 16 bytes (known as GUID in Session VQueries)
- **ColumnFamily:** *Properties*, single column family with following columns:
 - *nm (name)*: String can be repeated by other user, ...
 - *fp (FullPath)*: full path of the simulation data, so that it can be uniquely identified as used and known by the simulation users, this is the simulation model path in the cluster where the simulation program being executed becomes its source information, or where the simulation results comes from, so that the user can identify the model's origin and locate any needed information; and not the final path inside the VELaSSCo platform where the HDFS files lie)
 - *bb (BoundingBox)*: 6 double precision numbers
 - *vs (ValidationStatus)*: string
 - *np (Number of partitions)*: integer
 - *uName (uUserName)*: string (contains the access permissions, one of *no_access / read_only / read&write&delete* for user *UserName*)
 - *ot (Other data)*: Byte Array (optional)
 - Thumbnails are stored as files in HDFS, so they are not in this table.
 - Eventually another column(s) with the names of the tables that store the temporary data like multi-resolution models, skins, pre-computed queries.

Access permissions:

Access permissions are stored in the global simulation table for each simulation, i.e. row. There will be one column name (qualifier) per user starting with letter “u”, following with the user name and its value is the access permission to that model which can be: *no_access*, *read_only* or *read&write&delete*.

Special user name ALL is used to set the default access permissions for all users, when no specific permission is present.

The column uALL = "" should always be present to specify the default access to the model for all users.

Example:

Key	CF: Properties
0x0001	<i>nm</i> = "3Dcylbody", <i>fp</i> = "/user/KratosUserA/simulation/3Dcylbody", <i>bb</i> = {0 0 0 1 1 1}, <i>vs</i> = "Not validated", <i>np</i> = 1, <i>umiguel</i> = "read&write&delete", <i>ubenoit</i> = "read&write&delete", <i>uALL</i> = "read_only", <i>ot</i> = NULL
0x0002	<i>nm</i> = "Car", <i>fp</i> = "/user/KratosUserB/simulation/Car", <i>bb</i> = {-10 -5 -3 10 5 3}, <i>vs</i> = "Not valid, motive", <i>np</i> = 96", <i>ualvaro</i> = "read&write&delete", <i>uALL</i> = "read_only", <i>ot</i> = NULL
0x0003	<i>nm</i> = "FluidizedBed", <i>fp</i> = "/user/DemUserC/simul/FluidizedBed", <i>bb</i> = {-2 -2 -2 6 10 22}, <i>vs</i> = "Invalid, missing", <i>ujochen</i> = "read&write&delete", <i>uALL</i> = "no_access", <i>np</i> = 1

5.2 Simulations_Metadata: metadata of Simulation models

The table containing the Simulations_Metadata will have only three column families with several column names (qualifiers):

- **Key:** SimulationID + AnalysysName + Step (16 bytes + integer (4bytes = length) + String + double 8 bytes)
- **ColumnFamily:** *M* (stands for Mesh) with following columns:
 - *un* (units): String with the name of the units of the geometry
 - *C* coordinate set definitions:
 - *CCCCCCC##*: the full length name of the column qualifiers is a String of 9 bytes:
 - *CCCCCC*, 6 digits, is the number of coordinate set
 - *##*, 2 characters, specific for the column qualifiers types of the coordinates set.
 - *c000001nm* (Coordinates 000001 name): String, name of this coordinate set, for DEM will be p3p.
 - *c000001nc* (Coordinates 000001 number of coordinates): 64-bit integer
 - *c000002nm* (Coordinates 000002 name): String, name of this coordinate set, for DEM will be p3c.
 - *c000002nc* (Coordinates 000002 number of coordinates): 64-bit integer
 - ...
 - *CCCCCCCnm* (Coordinates CCCCCCC name): String, name of this coordinate set, for DEM will be p3p, p3c, p3w or w.
 - *CCCCCCCnc* (Coordinates CCCCCCC number of coordinates): 64-bit integer
 - *M* mesh definitions:
 - *mMMMMMMM##*: the full length name of the column qualifiers is a String of 9 bytes:
 - *MMMMMMM*, 6 digits, is the number of mesh set

- *##*, 2 characters, specific for the column qualifiers types of the mesh set.
 - *m000001nm* (*Mesh 000001 name*): String, name of the mesh 000001
 - *m000001cn* (*Mesh 000001 coordinates column name*): String, indicating the name (prefix) of the coordinates this mesh uses
 - *m000001et* (*Mesh 000001 element type*): String, one of {Point, Line, Triangle, Quadrilateral, Tetrahedron, Hexahedron, Prism, Pyramid, Circle, Sphere, ComplexShape, quadratic line for p2p contacts of DEM simulations} (or may be SphereCluster, as they are represented as cluster of Spheres...) (for the initial platform we only need to support **lines, quadratic lines, triangles, tetrahedrons and spheres**)
 - *m000001ne* (*Mesh 000001 number of elements*): 64-bit integer, number of elements of Mesh 000001
 - *m000001nn* (*Mesh 000001 number of nodes per element*): integer, indicates the number of nodes of the element, for instance tetrahedrons of 4 or 10 nodes
 - *m000001cl* (*Mesh 000001 color*): String (optional), with the colour of the mesh in hexadecimal format: #rrggbb or #rrggbbaa.
 - *mMMMMMMnm* (*Mesh MMMMMM name*): String, name of the mesh M
 - *mMMMMMMcn* (*Mesh MMMMMM coordinates column name*): String, indicating the name (prefix) of the coordinates this mesh uses
 - *mMMMMMMet* (*Mesh MMMMMM element type*): String, one of { Point, Line, Triangle, Quadrilateral, Tetrahedron, Hexahedron, Prism, Pyramid, Circle, Sphere, ComplexShape, quadratic line for p2p contacts of DEM simulations} (or may be SphereCluster, as they are represented as cluster of Spheres...)
 - *mMMMMMMne* (*Mesh MMMMMM number of elements*): 64-bit integer, number of elements of Mesh 1
 - *mMMMMMMnn* (*Mesh MMMMMM number of nodes per element*): integer, indicates the number of nodes of the element, to represent quadratic elements, for instance triangles with 3 or 6 nodes or tetrahedrons with 4 or 10 nodes
 - *mMMMMMMcl* (*Mesh MMMMMM color*): String (optional), with the colour of the mesh in hexadecimal format: #rrggbb or #rrggbbaa
- **ColumnFamily: G** (*stands for Gauss Points*) with following columns:
 - G gauss point definitions:
 - *gGGGGGG##*: the full length name of the column qualifiers is a String of 9 bytes:
 - *GGGGGG*, 6 digits, is the number of gausspoint set

- *##, 2 characters, specific for the column qualifiers types of the gausspoint set.*
- - *g000001nm (name):* String with the name of the gauss points
 - *g000001et (element type):* String with the element type for this gauss points definition
 - *g000001ng (number of gauss points):* integer, number of integration points, for instance 1, 3 or 6 for triangles
 - *g000001ni (nodes included):* Boolean, for gauss points on lines, if the ends are included in the definition or not
 - *g000001ic (internal coordinates):* Boolean, indicating if the internal definition is used or the definition provides its own set of natural coordinates
 - *g000001nc (natural coordinates):* double [g000001ng][3], natural coordinates of the definition, or empty if g000001ic == true
 - *g000001mn (mesh name):* String indicating the mesh name to which this definition refers, or empty if the gauss points are defined for all elements of type g000001et
 - ...
 - *gGGGGGGnm (name):* String with the name of the gauss points
 - *gGGGGGGet (element type):* String with the element type for this gauss points definition
 - *gGGGGGGng (number of gauss points):* integer, number of integration points, for instance 1, 3 or 6 for triangles
 - *gGGGGGGni (nodes included):* Boolean, for gauss points on lines, if the ends are included in the definition or not
 - *gGGGGGGic (internal coordinates):* Boolean, indicating if the internal definition is used or the definition provides its own set of natural coordinates
 - *gGGGGGGnc (natural coordinates):* double [gGGGGGGng][3], natural coordinates of the definition, or empty if gGGGGGGic == true
 - *gGGGGGGmn (mesh name):* String indicating the mesh name to which this definition refers, or empty if the gauss points are defined for all elements of type gGGGGGGet
 - This column family is not needed for the initial version of the platform, But as far as I know it should be present though, as it will be cumbersome to add it afterwards. For the initial version of the platform, this CF can remain empty for FEM simulations. For DEM simulations, the following data needs to be added so that it is possible to distinguish between p2p contacts and p2w contacts:
 - Column qualifier = *g000001nm* *value = gp_p2p*
 - Column qualifier = *g000001et* *value = line (or Line)*

- Column qualifier = *g000001ng* *value = 1*
- Column qualifier = *g000001ni* *value = false*
- Column qualifier = *g000001ic* *value = false*
- Column qualifier = *g000001nc* *value = (empty)*
- Column qualifier = *g000001mn* *value = p2p contacts*

- Column qualifier = *g000002nm* *value = gp_p2w*
- Column qualifier = *g000002et* *value = line (or Line)*
- Column qualifier = *g000002ng* *value = 1*
- Column qualifier = *g000002ni* *value = false*
- Column qualifier = *g000002ic* *value = false*
- Column qualifier = *g000002nc* *value = (empty)*
- Column qualifier = *g000002mn* *value = p2w contacts*

- Moreover there will be standard gauss points definitions and the meshes can use them (definitions at the end of the document):
 - GP_LINE_1,
 - GP_TRIANGLE_1, GP_TRIANGLE_3, GP_TRIANGLE_6,
 - GP_TETRAHEDRA_1, GP_TETRAHEDRA_4, GP_TETRAHEDRA_10,
 - GP_SPHERE_1
 - Other standard definitions will be added when the corresponding element types are added to the platform.
 -
- **ColumnFamily: R** (*stands for Result*) with following columns:
 - R result definitions:
 - *rRRRRRR##*: the full length name of the column qualifiers is a String of 9 bytes:
 - *RRRRRR*, 6 digits, is the number of result set
 - *##*, 2 characters, specific for the column qualifiers types of the result set.
 - *r000001nm* (*Result 000001 name*): String with the name of the result
 - *r000001rt* (*Result 000001 type*): String, one of {Scalar, Vector, Matrix, ?PlainDeformationMatrix?, ...}
 - *r000001nc* (*Result 000001 number of components*): integer, for scalars = 1, vectors = 2, 3 or 4, Matrices = 3, 4 or 6.
 - *r000001cn* (*Result 000001 component names*): String[*r000001nc*] (optional)
 - *r000001lc* (*Result 000001 location*): String, one of {OnNodes, OnGaussPoints}
 - *r000001gp* (*Result 000001 gauss point name*): String, with the gauss point name if *r000001lc* == OnGaussPoints

- *r000001co* (Result 000001 coordinates name): String (optional), referring to the coordinate column name if *r000001lc* == OnNodes and there are more than one coordinate set.
- *r000001un* (Result 000001 units name): String with the name of the units for this result
- ...
- *rRRRRRRnm* (Result RRRRRR name): String with the name of the result
- *rRRRRRRrt* (Result RRRRRR type): String, one of {Scalar, Vector, Matrix, ?PlainDeformationMatrix?, ...}
- *rRRRRRRnc* (Result RRRRRR number of components): integer, for scalars = 1, vectors = 2, 3 or 4, Matrices = 3, 4 or 6.
- *rRRRRRRcn* (Result RRRRRR component names): String[*rRRRRRRnc*] (optional)
- *rRRRRRRlc* (Result RRRRRR location): String, one of {OnNodes, OnGaussPoints}
- *rRRRRRRgp* (Result RRRRRR gauss point name): String, with the gauss point name if *rRRRRRRlc* == OnGaussPoints
- *rRRRRRRco* (Result RRRRRR coordinates name): String (optional), referring to the coordinate column name if *rRRRRRRlc* == OnNodes and there are more than one coordinate set.
- *rRRRRRRun* (Result 1 units name): String with the name of the units for this result

Example of FEM simulation with static mesh **Simulation_Metadata (3Dcylbody rows)**:

Key: <i>SimulationId</i> + <i>Analysis</i> + <i>Step</i>	CF: M	CF: G	CF: R
0x0001 +0+ "" + 0.0	un = "m", c000001nm = "coordinates", c000001nc = 339185, m000001nm = "exterior volume", m000001cn = "c000001", m000001et = "tetrahedral", m000001ne = 717793, m000001nn = 4, m000001cl = "#00ff00", m000002nm = "down 9", m000002cn = "c000001", m000002et = "triangle", m000002ne = 21720, m000002cl = "#4d4d80"	(empty)	(empty)
0x0001 +6+ "RANSOL" + 91.5	(empty)	(empty)	r000001nm = "Pressure", r000001rt = "scalar", r000001nc = 1, r000001lc = "OnNodes", r000001un = "Pa", r000002nm = "Velocity", r000002rt = "vector", r000002nc = 3, r000002cn = "v-x, v-

			y,v-z"; r000002lc = "OnNodes", r000002un = "m/s", r000003nm = "Vorticity", r000003rt = "scalar"; r000003nc = 1, r000003lc = "OnNodes", r000003un = "1/s"
...	(empty)	(empty)	...
0x0001 + 6+ "RANSOL" + 200.0	(empty)	(empty)	...

Example of particle simulation with dynamic mesh **Simulation_Metadata (FluidizedBed rows):**

Key: SimulationId + Analysis + Step	CF: M	CF: G	CF: R
0x0003 + 3+ "DEM" + 0.0	un = "", c000001nm = "p3p", c000001nc = 11880, c000002nm = "p3c", c000002nc = 3036, c000003nm = "p3w", c000003nc = 321, m000001nm = "particles", m000001cn = "c000001", m000001et = "sphere", m000001ne = 11880, m000001nn = 1, m000001cl = "#f0ff0f", m000002nm = "p2p contacts", m000002cn = "c000002", m000002et = "quadratic line", m000002ne = 3036, m000002cl = "#818181" m000003nm = "p2w contacts", m000003cn = "c000003", m000003et = "line", m000003ne = 321, m000003cl = "#0000ff"	g000001nm = "gp_p2p", g000001et = "line", g000001ng = 1, g000001ni = "false", g000001ic = "false", g000001nc = (empty), g000001mn = "p2p contacts" g000002nm = "gp_p2w", g000002et = "line", g000002ng = 1, g000002ni = "false", g000002ic = "false", g000002nc = (empty), g000002mn = "p2w contacts"	r000001nm = "Volume", r000001rt = "scalar", r000001nc = 1, r000001lc = "OnNodes", r000001co = "c000001", r1un = "", r000002nm = "Mass"; r000002rt = "scalar"; r000002nc = 1, r000002lc = "OnNodes", r000002co = "c000001", r2un = "", r000003nm = "Velocity", r000003rt = "vector", r000003nc = 3, r000003cn = "V-x,V-y,V-z"; r000003lc = "OnNodes", r000003co = "c000001", r000003un = "", r000004nm = "Force p2p", r000004rt = "vector"; r000004nc = 3, r000004cn = "Fx p2p,Fy p2p,Fz p2p", r000004lc = "OnGaussPoints", r000004gp = "gp_p2p", r000004un = "", r000005nm = "Force p2w", r000005rt = "vector"; r000005nc = 3, r000005cn = "Fx p2w,Fy p2w,Fz p2w", r000005lc = "OnGaussPoints", r000005gp = "gp_p2w", r000005un = ""
0x0003 + 3 + "DEM" + 0.xx
0x0003 + 3 + "DEM" + 87.0

5.3 Simulations_Data: meshes and results data of simulation models

This table contains the data of coordinates, meshes and results for all simulation models. Following the recommendations for HBase schema design, at the beginning of this document, the table will have only two column families with several column names (qualifiers). The data will be stored in binary form, to avoid binary-to-string conversions when storing and string-to-binary conversions when querying the data. Each vertex/coordinates, element and result value will be stored separately:

- the first letter indicates if it's a 'c' coordinate, 'm' mesh or 'r' result
- followed by 6 digits that represents the id's of the coordinate set, mesh set or result set respectively,
- In the case of column qualifiers related to meshes 'm' and results 'r', 2 characters.
- followed by 1 special character “_”
- followed by 8 bytes which are the 64-bit-integer of the Id of the vertex, mesh element or result value respectively.

The row key and the two column families are:

- **Key:** SimultionID + AnalysisName + Step + PartitionId (16 bytes + integer (4bytes = length) + String + double (8 bytes) + integer (4 bytes))
- **ColumnFamily:** *M (stands for Mesh)* with following columns (key-value pairs):
 - *C* coordinate set data: each vertex will be stored separately as:
 - *cCCCCCC_id*: double[3] (the full length name of the column qualifiers is a String of 8 bytes + 8 bytes for the id)
 - *CCCCC, 6 digits*, is the number of coordinate set referenced by the Mesh in metadata.mMcn ;
 - *Id, int64 (8 bytes)*, is the id of the vertex stored in binary form ;
 - *double[3]*: coordinates of the vertex
 - *c000001_1* = (34.55921297, 19.83166827,86.05163597)
 - *c000001_2* = (89.54424860, 92.98155843,91.49390204)
 - ...
 - *c000001_87654321* = (16.213, 56.607, 50.641)
 - ...
 - *cCCCCCC_82345* = (17.8073115, 63.9179244, 57.193386)
 - *cCCCCCC_88888888* = (82.87357, 82.66503, 66.52144)
 - *M* mesh data: each element will be stored separately as:
 - *mMMMMMM##_id*: double[number of nodes per element] (the full length name of the column qualifiers is a String of 10 bytes + 8 bytes for the id)
 - *MMMMMM, 6 digits*, is the number of mesh set.
 - *## is 2 characters to distinguish between the different types of column qualifiers related to mesh.*
 - *Id, int64 (8 bytes)*, is the id of the element stored in binary form

- normal FEM elements:
 - **triangles:**
 - m000001cn_1 = n1n2n3 (triangle using nodes/vertices n1, n2, n3 (int64[3]), and
 - m000001gr_1 = element 1 belongs to group GG (int64)
 - **tetras:**
 - m020406cn_8765434 = n1n2n3n4 and
 - m020406gr_8765434 = GG
 - **hexas:**
 - m844444cn_212312 = n1n2n3n4n5n6n7n8 and
 - m844444gr_212312 = GG
 - ... **quadratic** elements ...
- Particles (DEM elements):
 - **Spheres:**
 - m000001cn_1 = n1 for sphere with id 1, using vertex n1 (int64)
 - m000001rd_1 = RR for sphere with id 1, using radius RR (double)
 - m000001gr_1 = GG for sphere with id 1, belonging to group GG (int64).
- Contacts (DEM elements):
 - **Quadratic lines (p2p contacts):**
 - m000002cn_1 = n1n2n3 (quadratic line using nodes/vertices n1, n2, n3 (int64[3])). The n1 and n2 corresponds to nodes/vertices of the coordinates set “p3p” (c000001_id) and n3 corresponds to nodes/vertex of the coordinates set “p3c” (c000002_id).
 - m000002gr_1 = element 1 belongs to group GG (int64). This column qualifier can be empty or omitted for the first prototype.
 - **Lines (p2w contacts):**
 - m000003cn_1 = n1n2 (line using nodes/vertices n1, n2 (int64[2])). The n1 corresponds to nodes/vertices of the coordinates set “p3p” (c000001_id) and n2 corresponds to nodes/vertex of the coordinates set “p3w” (c000003_id).
 - m000003gr_1 = element 1 belongs to group GG (int64). The value corresponds to the ID provided for each contact below the column “WALL” of p3w file.
 - **Circles:**

- m000002cn_1234 = n1 for circle with id 1234, using vertex n1 as centre (int64),
 - m000002rd_1234 = RR for circle with id 1234, using radius RR (double),
 - m000002no_1234 = NxNyNz for circle with id 1234, using normal NxNyNz (double[3]),
 - m000002gr_1234 = GG for circle with id 1234, belonging to group GG (int64).
- - specific for ComplexShape / SphereCluster
 - **ColumnFamily: R** (*stands for Result*) with following columns:
 - R result data with one column qualifier per result value
 - rRRRRRRvl_id: double[rRRRRRRnc] (the full length name of the column qualifiers is a String of 10 bytes + 8 bytes for the id)
 - - RRRRRR, 6 digits, is the number of the result set defined in the metadata table;
 - "vl" is 2 characters.
 - Id, int64 (8 bytes), is the id of the value (may reference to a vertex or a element stored in binary form) ;
 - double[rRRRRRRnc]: values of the result for that vertex/element:
 - 1 value for nodal scalar
 - 2, 3 or 4 values for nodal vectors
 - 3, 4 or 6 values for nodal matrices
 - If the result is defined for the element (gauss points), the above values are repeated for all gauss points of the element (one triangle with 6 gp will have one entry rRRRRRRvl_1 with 6 scalars values, or 4 * 6 = 24 vector values)
 - **Scalar:**
 - r000001vl_1 = (34.55921297)
 - r000001vl_2 = (89.54424860)
 - **Vector:**
 - rRRRRRRvl_82345 = (17.8073115, 63.9179244, 57.193386)
 - rRRRRRRvl_87654321 = (16.213, 56.607, 50.641)

Example of FEM simulation with static mesh **Simulations_Data (3Dcylbody rows):**

Key: SimulationId + Analysis + Step + PartitionID	CF: M	CF: R
0x0001 + 0 + "" + 0.0 + 0	c000001_0 = (0.0, 0.0, 0.0), c000001_339185 = (1.0, 1.0, 1.0),	(empty)

	m000001cn_0 = (1, 2, 3, 4), m000001gr_0 = (1), m000001cn_717793 = (339184, 339183, 339182, 339181), m000001gr_717793 = (2), m000002cn_717794 = (1, 2, 3) m000002cn_739513 = (7,66, 8)	
0x0001 + 6 + "RANSOL" + 91.5 + 0	(empty)	r000001vl_1 = (4), r000001vl_339184 = (324), r000002vl_1 = (1.1, 2.2, 3.3), r000002vl_339184 = (10.0, 10.0, 9.1), ...
0x0001 + 6 + "RANSON" + 200.0	(empty)	...

Example of particle simulation with dynamic mesh **Simulations_Data (FluidizedBed rows)**:

Key: SimulationId + Analysis + Step + PartitionId	CF: M	CF: R
0x0003 + 3 + "DEM"+0.0+0	c000001_1=(0.0005, -0.0055, 0.0005), ..., c000001_11880 = (0.00237071 0.00278272 0.00786869), c000002_1=(0.0005, -0.0055, 0.0005), ..., c000002_3036 = (0.00317887, 0.0452111, 0.00928326), c000003_1=(0.0005, -0.0055, 0.0005), ..., c000003_463 = (0.00317887, 0.0452111, 0.00928326),m000001cn_1={1}, m000001rd_1={1.0472e-06}, m000001gr_1={3}, ..., m000001cn_11880={11880}, m000001rd_11880={9.04779e-07}, m000001gr_1={1}, m000002cn_1={181, 241, 1}, m000002gr_1={}, ..., m000002cn_3036={8004, 10478, 3036}, m000002gr_1={}, m000003cn_1={1, 1}, m000003gr_1={}, ..., m000003cn_463={11685, 463}, m000003gr_1={0}	r000001vl_1 = (5.23599e-10), r000001vl_2 = (5.23599e-10), ... r000001vl_11880 = (9.04779e-10), r000002vl_1 = (1.0472e-06), r000002vl_2 = (5.23599e-10), ... r000002vl_11880 = (9.04779e-07), r000003vl_1 = (0.0,0.0,0.0), r000003vl_2 = (0.0,0.0,0.0), ... r000003vl_11880 = (0.0263565, -0.0355245, -0.00972661), r000004vl_1 = (-8.67362e-17, 0.0, 0.0), r000004vl_2 = (-1.73472e-16, 0.0, 0.0), ... , r000004vl_3036 = (-4.56173e-05, - 8.20138e-06, 1.15113e-05), r000005vl_1 = (-9.03448e-320, - 1.23605e-314, -1.23471e-314), r000005vl_2 = (-9.03448e-320, - 1.23605e-314, -1.23471e-314), ... , r000005vl_463 = (1.63278e-05, - 3.22285e-05, -0.000361286)
...
0x0003 + 3 + "DEM"+99999+0

5.4 Simulations_VQuery_Results_Metadata: *metadata of the Vqueries*

This table stores the name and parameters of the VQueries whose data is stored in the Simulations_Vquery_Results_Data table.

Simulations_VQuery_Results_Metadata:

- Rowkey: (SimID + An + Step + QID)
- CF: M, G, R (as in Simulations_Metadata table)
- CF: Q with
 - *vg*: “Vquery-name”, *qp*: “Vquery-parameters”, *ul*: “user list who performed the query”
- QID = md5(QueryName + QueryParameters)

5.5 Simulations_VQuery_Results_Data: data of the Vqueries

This table stores the mesh and results data of the performed VQueries and other ‘temporary’ data like interpolation information for isosurfaces or cutplanes. This table will also store the output of the Spline Volume representation.

Is to be seen if the output of the Discrete2Continuum transformation data is stored in this table or in the Simulations_Metadata and Simulations_Data tables.

Simulations_VQuery_Results_Data:

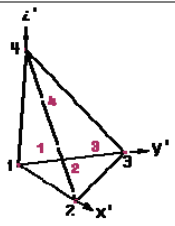
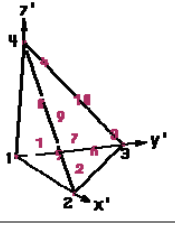
- Rowkey: (SimID + An + Step + QID + PartID)
- CF: M, R (as in Simulations_Data table)
- CF: Q with
 - *qr*: “Other results that cannot be stored in CF:M or CF:R columns ”
- QID = md5(QueryName + QueryParameters)

5.6 Standard Gauss Points definition

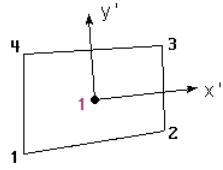
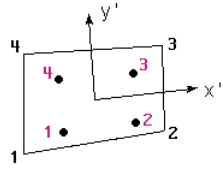
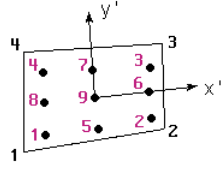
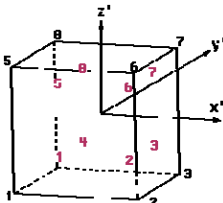
All element types supported by the platform and visualization clients will have a single gauss point at the center of the element, to represent cell values.

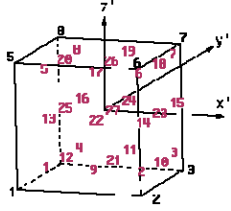
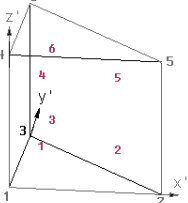
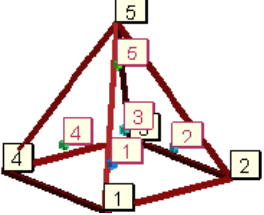
The standard Gauss Points to be internally defined and understood by the VELaSSCo platform and the visualization clients are:

For the initial version of the platform:		
Name	Location in element	Internal coordinates
GP_LINE_1	centre of line	
GP_TRIANGLE_1		$a=1/3$ (a, a)
GP_TRIANGLE_3		$a 1/2$ (a, 0) (a, a) (0, a)

GP_TRIANGLE_6		$a=0.09157621$ $b=0.81684757$ $c=0.44594849$ $d=0.10810301$ $(a, a) (b, a) (a, b)$ $(c, d) (c, c) (d, c)$
GP_TETRAHEDRA_1	Centre of element	
GP_TETRAHEDRA_4		$a=(5+3*\sqrt{5})/20=0.585410196624968$ $b=(5-\sqrt{5})/20=0.138196601125010$ $(b, b, b) (a, b, b) (b, a, b) (b, b, a)$
GP_TETRAHEDRA_10		$a=0.108103018168070$ $b=0.445948490915965$ $c=0.816847572980459$ $(a, a, a) (c, a, a) (a, c, a) (a, a, c)$ $(b, a, a) (b, b, a) (a, b, a)$ $(a, a, b) (b, a, b) (a, b, b)$
GP_SPHERE_1	centre of sphere	

Other standard gauss point definitions, eventually for the final version of the platform:

Name	Location in element	Internal coordinates
GP_QUADRILATERAL_1		$(0, 0)$
GP_QUADRILATERAL_4		$a=0.57735027$ $(-a, -a) (a, -a)$ $(a, a) (-a, a)$
GP_QUADRILATERAL_9		$a=0.77459667$ $(-a, -a) (a, -a) (a, a)$ $(-a, a) (0, -a) (a, 0)$ $(0, a) (-a, 0) (0, 0)$
GP_HEXAHEDRA_1	centre of the element	
GP_HEXAHEDRA_8		$a=0.577350269189626$ $(-a, -a, -a) (a, -a, -a) (a, a, -a) (-a, a, -a)$ $(-a, -a, a) (a, -a, a) (a, a, a) (-a, a, a)$

<p>GP_HEXAHEDRA_27</p>		<p>$a = 0.774596669241483$ $(-a, -a, -a) (a, -a, -a) (a, a, -a) (-a, a, -a)$ $(-a, -a, a) (a, -a, a) (a, a, a) (-a, a, a)$ $(0, -a, -a) (a, 0, -a) (0, a, -a) (-a, 0, -a)$ $(-a, -a, 0) (a, -a, 0) (a, a, 0) (-a, a, 0)$ $(0, -a, a) (a, 0, a) (0, a, a) (-a, 0, a)$ $(0, 0, -a)$ $(0, -a, 0) (a, 0, 0) (0, a, 0) (-a, 0, 0)$ $(0, 0, a)$ $(0, 0, 0)$</p>
<p>GP_PRISM_1</p>	<p>centre of the element</p>	
<p>GP_PRISM_6</p>		<p>$a=1/6=0.1666666666666666$ $b=4/6=0.6666666666666666$ $c=1/2-1/(2\sqrt{3})=0.211324865405187$ $d=1/2+1/(2\sqrt{3})=0.788675134594812$ $(a, a, c) (b, a, c) (a, b, c)$ $(a, a, d) (b, a, d) (a, b, d)$</p>
<p>GP_PIRAMID_1</p>	<p>centre of the element</p>	
<p>GP_PIRAMID_5</p>		<p>$a=8.0*\sqrt{2.0/15.0}/5.0$ $=0.584237394672177$ $b=-2/3=-0.6666666666666666$ $c=2/5=0.4$ $(-a, -a, b)$ $(a, -a, b)$ (a, a, b) $(-a, a, b)$ $(0.0, 0.0, c)$</p>
<p>GP_CIRCLE_1</p>	<p>centre of the element</p>	