H2020-EINFRA-2017

EINFRA-21-2017 - Platform-driven e-infrastructure innovation

DARE [777413] "Delivering Agile Research Excellence on European e-Infrastructures"

D4.6 Data Consolidation and Linking Toolkit II

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Deliverable	D4.6 Data Consolidation and Linking Toolkit II
Work package	WP4: Big Data Processing and Analytics
Tasks involved	T4.3
Type	DEM: Demonstrator, pilot, prototype
Dissemination Level	PU = Public
Due Date	31/12/2020
Submission Date	30/12/2020
Status	Draft
Editor(s)	Antonis Lempesis (ATHENA)
Contributor(s)	Antonis Lempesis (ATHENA)
Reviewer(s)	
Document description	The report accompanies the software deliverable for the Data Consolidation and Linking components integrated in the DARE platform. It provides a short description on the different components, their role within the DARE platform and their current status of maturity. It also provides links to relevant code and documentation.

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
1	24/12/2020	Initial version	A. Lempesis (ATHENA)
2	30/12/2020	Final	A. Lempesis (ATHENA)

Executive Summary

The document summarizes the activities carried out under T4.3, Data Consolidation and Linking Toolkit, of the project. During the first period of the project, the task laid the foundation for enabling the DARE platform to handle heterogeneous and disparate voluminous data sources. Following the general architectural principles and integration paradigm adopted by DARE, the relevant components were packaged, exposed and integrated in the DARE platform. Furthermore, the semantic representation required for enabling the exploitation of linked data within the platform has been based on established standards in order to facilitate future extensions and the reach of the platform in terms of easily incorporating different data sources.

Table of Contents

List of Terms and Abbreviations

Abbreviation	Definition
UDF	User Defined Function
SQL	Structured Query Language

1 Introduction

1.1 Purpose and Scope

The present document accompanies the software components that comprise deliverable D4.6, Data Consolidation & Linking Toolkit II, of the project.
It provides an overview of the design of the DARE Data Catalogue, part of the DARE Knowledge Base, and provides information on its technical realization at the current stage of the project.
Additionally, it summarizes the work performed on the Exareme component during the second period of the project and reports on the design and API of the semantic registries used by the DARE platform. Since this is the second version of the deliverable (D4.5 being the first version), it was decided that the contents of D4.5 will remain and the progress during the second period of the project will be added. In the first version of the report emphasis was given on the initial design and initial implementation rather than performance. In the second version, we describe new developments in terms of functionality but also address performance issues that arised.

1.2 Approach and relationship with other Work Packages and Deliverables

Data Consolidation and Linking components are used to support the two use cases of the project: EPOS and IS-ENES/Climate4Impact. As such, it is directly related to WP6 and WP7 and more specifically D6.1 and D7.1. Moreover, in the case of Exareme, a general-purpose tool designed to be directly invoked by dispel4py, it is related to D4.3. Additionally, the design of the Data catalogue semantic registries is based on work related to the DARE Semantic registries on WP2, while the implementation of the relevant APIs relates to work on the SemaGrow component, part of D4.1, Big Data Analytics Toolkit.

1.3 Methodology and Structure of the Deliverable

The structure of this deliverable is as follows. In the first section a brief description of the design of the DARE Data Catalogue is presented. Additionally, the section provides a high-level description of Exareme. It also presents its components, the modifications and enhancements to it in order to support the two use cases and, finally the Exareme method that was used and tested in the two use cases.
The second section describes the means of integration of the Data Consolidation and Linking components within the DARE platform.

2 DARE Data Consolidation and Linking Toolkit

2.1 DARE Data Catalogue

2.1.1 Catalogue Design

The DARE catalogue constitutes the functional implementation of the Dare Knowledge Base, focusing at the present stage on the interfacing with the enactment system. It is logically divided in three catalogues, handling Processing Elements (PEs), Components, and Data respectively, corresponding to three concepts in the dare:context.
The Data Catalogue in particular acts as a metadata registry for the datasets used and produced by the users. Conceptually, we can distinguish between descriptive and functional characteristics of the datasets.
The former includes information on the owner and creator of the dataset (at the personal and organizational levels), define relevant topics and themes and provide links to other assets, e.g., other datasets or publications. Such information is expected to be used for discovery and linking.

The latter are more closely associated with the operation of the platform as they specify access rights, the type and location of the file(s) included in the dataset, as well as temporal and spatial information for the equipment and/or process that produced the dataset.
The data catalogue uses concepts defined by the DCAT W3C Recommendation[1]. Hence, datasets described within the catalogue are conceptualized as instances of the dare:Dataset class, a subclass of dcat:Dataset. Additional properties for associating a dataset with the overall operation of the DARE platform (i.e. the dataset's creator/contributor and the processes within the platform that led to the creation of the dataset) are included in the catalogue's schema. Taking into account the work on EPOS-DCAT, further concepts for scientific equipment (dare:Equipment) and facilities (dare:Facility) are defined, as generalizations of epos:Equipment and epos:Facility concepts.

2.2 Exareme

Exareme is a distributed and federated processing engine that provides a declarative language to define dataflows. Exareme's algorithms are written in SQL with several Python extensions. Exareme's functionality is based on two different types of nodes:

- **Master.** Exareme's master node is providing the REST API, receives requests, submits jobs to the workers and returns the responses to the clients.

- **Worker.** Exareme's worker is the execution engine which receives jobs from the master node, processes the data and returns the results.

2.2.1 Modifications to Exareme

In order to support the IS-ENES/Climate4Impact use case a number of significant changes have been made to the Exareme and new functionality has been added.

Automated Support of federated (local/global) execution.

In its initial state, Exareme was strictly a distributed execution engine. This meant that the master node was responsible for accessing the data to be processed and distributing it to the worker nodes. However, in the Climate4Impact use case the data is already stored in multiple data nodes. To better support the use case, Exareme was modified to separately process the datasets in a local step (in the best case, one worker node per data node but more than one data nodes can be assigned to a worker node, if the computing resources are not enough) and then receive the results in its master node and merge/aggregate them in a final global step.

In essence, Exareme has evolved to also function as a federated query and processing engine, able to process different datasets with potentially different code in each worker node and merge the results as a final step in the master node. Of course, for more complex use cases, nothing prohibits Exareme from working in a "hybrid" mode where the results of the federated execution are further processed in a distributed manner, by taking advantage of the worker nodes.

Support of Python scripts.

Exareme's processing processing power as a distributed query engine was supported by a large number of user defined functions (UDF) that allowed its users to manipulate and analyze the data using pure SQL. However, in order to support the Climate4Impact use case, Exareme was modified to support the direct execution of Python scripts. This was deemed necessary because the code

[1] https://www.w3.org/TR/vocab-dcat/

from ENES was already developed (in Python) and it would require too much effort to rewrite using the Exareme UDFs and SQL.

Deployment

Exareme is deployed and runs as a service (as opposed to other processing elements that are executed on demand and are shut down when the particular computations have been completed). It receives queries and returns the results through a RESTful API on the master node. The worker nodes are statically deployed (their number and location is determined when the DARE platform is deployed) but this is planned to change in the future.

During the second period of the project the development on Exareme continued by adding extra features and addressing performance issues. More specifically, the following features were added:

Iterative local/global execution

In the first version of the federated execution mode of Exareme, only one round local and then global execution was available. This means that processing was performed in the worker nodes with locally located data ("local" step), the results were returned to the master node and the final step of the computation took place there ("global" step). This mode of operation was enough for some cases but there are algorithms that require that the local/global mode should be repeated. This feature was implemented and now Exareme is able to perform iterations of local/global computations, which terminate when a user defined condition is met (e.g. a fixed amount of iterations has been performed or the result has converged).

Predefined Algorithms

A large number of algorithms has been implemented and are available out of the box for all Exareme users. Some of them are Machine Learning oriented but the majority is general purpose. An indicative list of the implemented algorithms includes:

- k-fold cross validation
- CART
- Hold-out validation
- Kaplan-Meier Estimator
- Logistic Regression
- Naive Bayes Training
- Pearson Correlation
- Iterative Dichotomiser 3 (ID3)
- Descriptive Statistics
- Calibration Belt
- ANOVA

For a complete list of the implemented algorithms, see https://gitlab.com/project-dare/ExaSpark/-/tree/master/Exareme-Docker/src/mip-algorithms.

Kubernetes

A major shortcoming of Exareme during the first phase of the project was its inability to be deployed using Kubernetes. Instead, Exareme could only be installed as a standalone service or using Docker swarm, an orchestration platform that is competing and incompatible to Kubernetes, that is used by the rest of the DARE platform. In the second period of the project, this shortcoming was

addressed: Exareme can now be deployed using Kubernetes, thus making the DARE platform more uniform and much easier to install and maintain.

Optimizations

Apart from the new features that were implemented in the second period of the project, one very significant performance enhancement was introduced. In previous versions of Exareme, the worker nodes were launched whenever a new task was to be executed. This resulted in long startup times that, especially for short lived tasks, could amount up to 90% of the total runtime. In the latest version of Exareme, the worker nodes are launched as services that incur the startup cost only once and then wait for jobs to execute. This change not only greatly improves the performance of Exareme (by also allowing for the caching of tasks and their results), but also enables the creation of stateful UDFs or Python scripts.

The Exareme code is hosted in the DARE GitLab server and can be accessed in https://gitlab.com/project-dare/ExaSpark

2.3 Deployment Testbed

2.3.1 IS-ENES/Climate4Impact Use Case

For the Climate4Impact use case the need existed for efficient processing of large amounts of data, coming from multiple data nodes. In this case, the federated processing capabilities of Exareme are suitable for the implementation of the use case.
Exareme has been tested with Python code and sample NetCDF datasets. Specifically, the Python code that we ran with Exareme is based on Python's library icclim[2] which processed NetCDF files. The test took place on 3 NetCDF files which were stored across three Exareme worker nodes. Python's script processed locally the datasets and the results were returned to the Exareme's master node. Exareme's compression did not enhance the execution since NetCDF is already a compressed representation format.

2.3.2 EPOS Use Case

For the EPOS use case, the distributed or federated functionality of Exareme was not needed. Instead, since in this use case very large files must be copied over the network from their storage to the DARE platform, it was decided to evaluate whether Exareme's compression algorithm could be used to reduce the size of the data.
Exareme's compression for tabular data was tested with a sample MSEED file. The data were saved in 100HZ, which is the most commonly used rate. The duration of the time series was about 105 minutes. The tests showed that there was no compression benefit since MSEED files are already very efficiently compressed.
As a result, it was decided that Exareme will not be used in the EPOS use case, since it would not offer any real benefits.

[2] https://icclim.readthedocs.io/en/latest/

3 Integration in DARE Platform

3.1.1 Data Catalogue API

The Data Catalogue define an LDP-compliant API for submitting and retrieving information to the underlying graph database.

The triple store used is a Virtuoso database, packaged and deployed in accordance to the BDI paradigm, and including additional libraries for using the Jena API in order to handle RDF data.

At the moment, the API exposes two RESTful calls:

listdatasets: It returns a list of the datasets accessible to the user corresponding to the user token provided as input to the call.

submitdataset: records information on a new dataset imported to the platform. The information includes (and is given as parameter to the REST call):

- The user submitting the dataset (and specifically her user token).
- A name and description for the dataset
- The component that produced the dataset
- The process that generated the dataset
- The workflow that was executed for producing the dataset
- The date of submission
- The path on the distributed file system accessible to the platform where the dataset can be found

The implementation of the API can be found at: https://gitlab.com/project-dare/data-catalogue

3.1.2 Exareme communication with other components

dispel4py/Exareme communication. Disple4py submits a job using Exareme's RESTful API. The parameters include the Python's script that will run and the selected datasets. Exareme's response contains a complex JSON file that includes one result for each dataset.

Exareme Processing. Exareme's master receives the request and runs the algorithm in parallel across all the local/worker nodes. Then the local nodes return their result to the master. Exareme master merges the local results. Possible further global aggregation is a second step of a dispel4py workflow. This design allows Exareme to return all the local results with information about the local datasets so that provenance is available through all the intermediate steps of the federated algorithm.

Provenance Support. When Exareme executes a local step, it returns the local result with several information required to support provenance (i.e., the data node it ran, version of data etc.). These intermediate local results are expressed in JSON. Then, Exareme's master node combines these multiple local JSON results in one bigger JSON. This JSON file is returned to dispel4py.