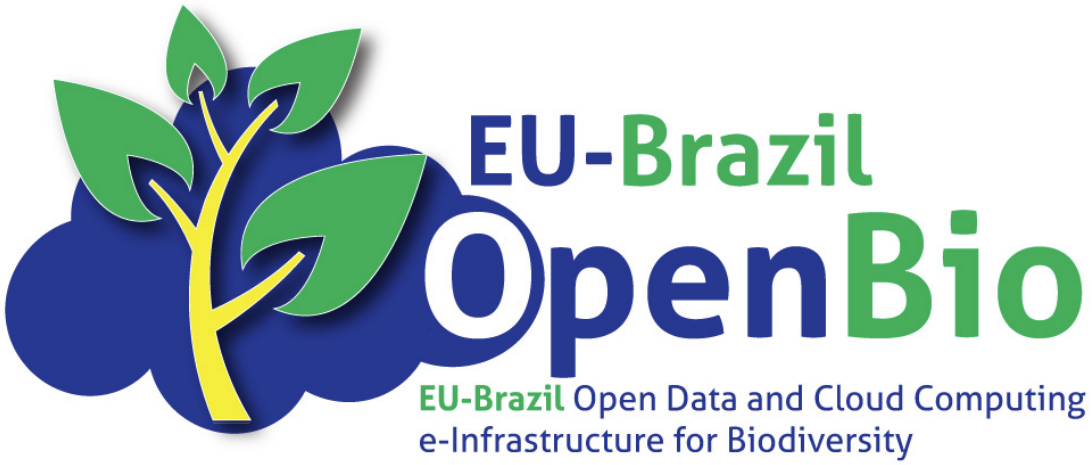

D4.4 Infrastructure Enhancement and Optimisation Final Report

Contract number: FP7-288754 / CNPq 590052/2011-0

Start Date of Project: 1 June 2011

Duration of Project: 28 months

Work Package: WP4 – Infrastructure Set-up, Integration and Testing

Due Date: M28 – 30/09/2013

Submission Date: 18/10/2013

Partner Responsible for the Deliverable: BSC (EU) / UFF (BR)

Dissemination Level: PU

Nature: Report

Author(s): Daniele Lezzi (BSC), Erik Torres (UPVLC), Francisco Quevedo (Species2000), Ignacio Blanquer (UPVLC), Renato De Giovanni (CRIA), Jose Fernando (CESAR), Vinicius Garcia (CESAR), Rodrigo Assad (CESAR), Cristina Boeres (UFF), Fernanda Oliveira Passos (UFF), Rafael Amaral (UFF), Vinod Rebello (UFF), Leonardo Candela (CNR).

Reviewer(s): Renato De Giovanni (CRIA)

Disclaimer

EUBrazilOpenBio - Open Data and Cloud Computing e-Infrastructure for Biodiversity (2011-2013) is a Small or medium-scale focused research project (STREP) funded by the European Commission under the Cooperation Programme, Framework Programme Seven (FP7) Objective FP7-ICT-2011-EU-Brazil Research and Development cooperation, and the National Council for Scientific and Technological Development of Brazil (CNPq) of the Brazilian Ministry of Science and Technology (MCT) under the corresponding matching Brazilian Call for proposals MCT/CNPq 066/2010.

This document contains information on core activities, findings, and outcomes of EUBrazilOpenBio project, and in some instances, distinguished experts forming part of the project's Strategic Advisory Board. Any references to content in both website content and documents should clearly indicate the authors, source, organization and date of publication.

The document has been produced with the co-funding of the European Commission and the National Council for Scientific and Technological Development of Brazil. The content of this publication is the sole responsibility of the EUBrazilOpenBio Consortium and its experts and cannot be considered to reflect the views of the European Commission nor the National Council for Scientific and Technological Development of Brazil.

Table of Contents

Executive Summary

The EUBrazilOpenBio project aims to design a novel Data and Cloud Computing e-Infrastructure by integrating existing EU and Brazilian technologies and resources to support the processing demands of two biodiversity-related problems that are both data intensive and computationally intensive. This document is the last report on 'Infrastructure Enhancement and Optimization' and aims to describe the integration of technologies and resources that compose the production level infrastructure. In particular, the report focuses on the advancements and improvements made in the development of the components needed to implement the infrastructure and on their usage for the final version of the use cases.

The first report [4] discussed the initial technological planning (described in [1]) behind the definition of an appropriate configuration of both the technology and the infrastructure to serve the needs of the two use cases. The results of a detailed performance analysis of the original proposal have been used to help identify the set of enhancements and optimisations to improve the first prototype of the infrastructure that included data and computational technologies (described in [2][3]5]) as gCube, openModeller, EasyGrid AMS, COMPSs, Usto.re, VENUS-C and HTCondor.

The second report [9] detailed how the identified building components have been put together in order to implement the use cases and how they benefitted from the adoption of these components. The final structure of the EUBrazilOpenBio e-Infrastructure was presented with an exhaustive analysis of the optimization activities performed to implement the use cases.

1 Introduction

The EUBrazilOpenBio project aims to build an e-Infrastructure by leveraging primarily on resources (textual publications and datasets, maps, taxonomies, tools, services, computing and storage capabilities) provided by European and Brazilian e-Infrastructures sustained by existing projects and initiatives. Interoperation extends through all of the infrastructures namely: hardware & computing facilities (Cloud and Grid computing, Internet), portals and platforms as well as the scientific data knowledge infrastructures. The development of the e-Infrastructure has been driven by the requirements of two biodiversity use cases: Use Case I – Integration between Regional & Global Taxonomies, and Use Case II – Data usability and the use of ecological niche modelling (ENM) but the architecture of the infrastructure is generic and modular thus allowing the implementation of more applications that require the services provided by EUBrazilOpenBio.

This deliverable reports on the final results of the application of the cloud programming models and management paradigms identified in the context of the use cases on the EUBrazilOpenBio production e-infrastructure. The objective in the implementation of this production version was to enhance the previous e-infrastructure in order to integrate the developed services and components in a unified environment and to further enhance the implementations to take advantage of the integrated data resources and the variety of distributed computing and storage resources available.

The report discusses the basic components integrated in the EUBrazilOpenBio infrastructure and the enhancements and optimizations to enable the execution of the use cases on the available resources. The access to the infrastructure is enabled in the form of Virtual Research Environments, based on gCube, that interact with the data and computational resources (Usto.re, VENUS-C Cloud, HTCondor and legacy openModeller servers) through use case specific interfaces. A specific section is dedicated to the application of programming models and execution paradigms for the optimized execution of the use cases on the available resources and to the extensions to access external infrastructures through the adoption federation models.

2 EUBrazilOpenBio e-Infrastructure

This section describes the final production version of the EUBrazilOpenBio e-Infrastructure that offers applications as services to the biodiversity scientific community. In particular, the functionalities and capabilities provided by the infrastructure have been designed according to the requirements of the two project use cases.

As depicted in Figure 1, the EUBrazilOpenBio hybrid infrastructure includes (i) data storage facilities based on gCube and Usto.re, (ii) computing services that allow the execution of the use cases optimized through the adoption of programming models and application managers such as COMPSs and EasyGrid AMS, (iii) a variety of computational infrastructure accessed through the integration of management middleware, currently VENUS-C and HTCondor, and (iv) external data sources, for example, GBIF and CoL.

The core services of the architecture mainly rely on gCube facilities and D4Science.org support. The gCube core services (namely Information System, Workspace and Storage Service, Species Discovery Service and Result and Visualization Service), deployed on a number of gCube nodes, allow the enactment of the use cases by the provision of Virtual Research Environments (VREs).

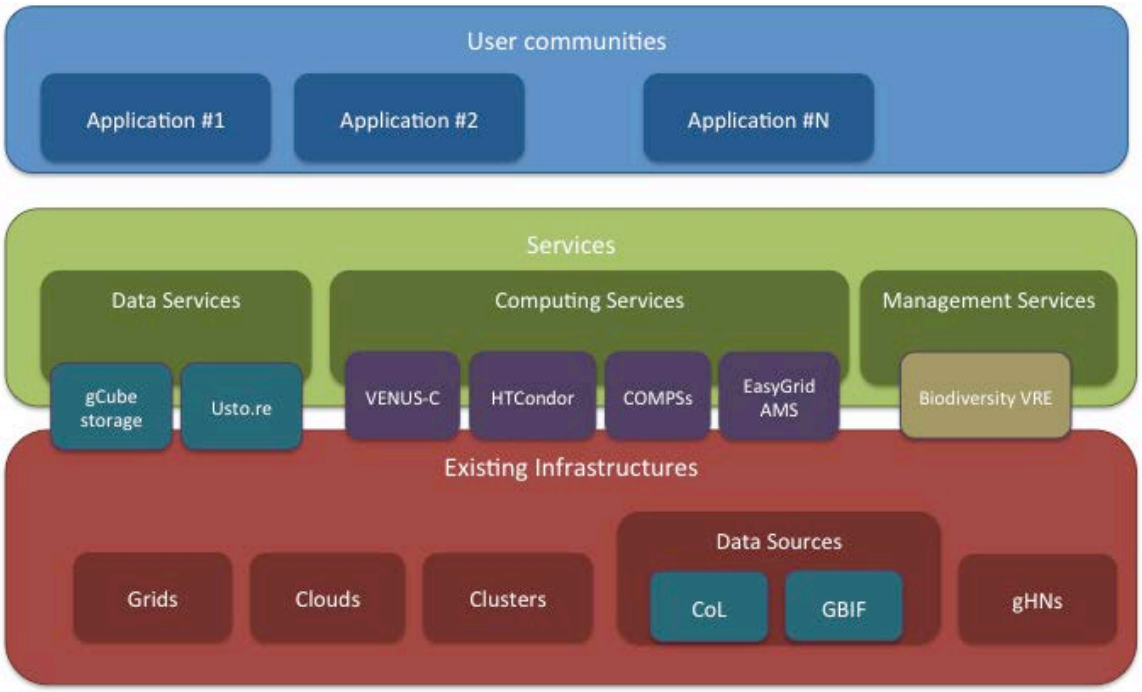

Figure 1 - Architecture of EUBrazilOpenBio e-Infrastructure

An up to date picture of the status of the infrastructure can be acquired by the EUBrazilOpenBio Gateway available at https://portal.eubrazilopenbio.d4science.org. In particular, the monitoring tool makes it possible to browse the list of "resources" contributing to the EUBrazilOpenBio infrastructure.

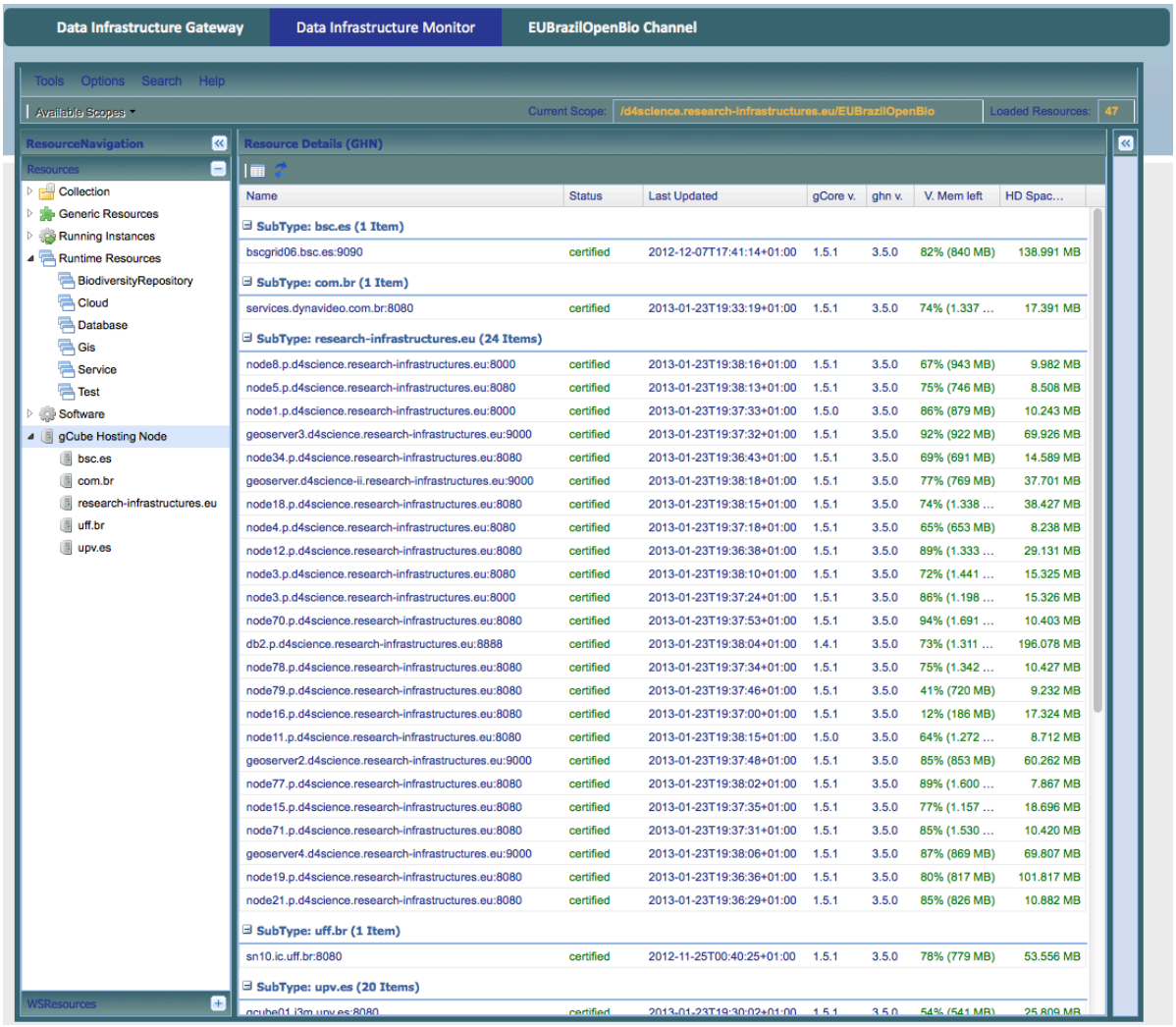

Figure 2. EUBrazilOpenBio Gateway: Infrastructure Monitoring Facility Screenshot

2.1 The SpeciesLab Virtual Research Environment

Virtual Research Environments are "systems" aiming to provide their users with web-based working environments that offer the entire spectrum of facilities (including services, data and computational facilities) needed to accomplish a given task by dynamically relying on the underlying infrastructure.

In the context of EUBrazilOpenBio, the SpeciesLab Virtual Research Environment has been created in order to provide its users with an integrated environment supporting the two target use cases, i.e. the cross-mapping of taxonomies and the production of species distributions models.

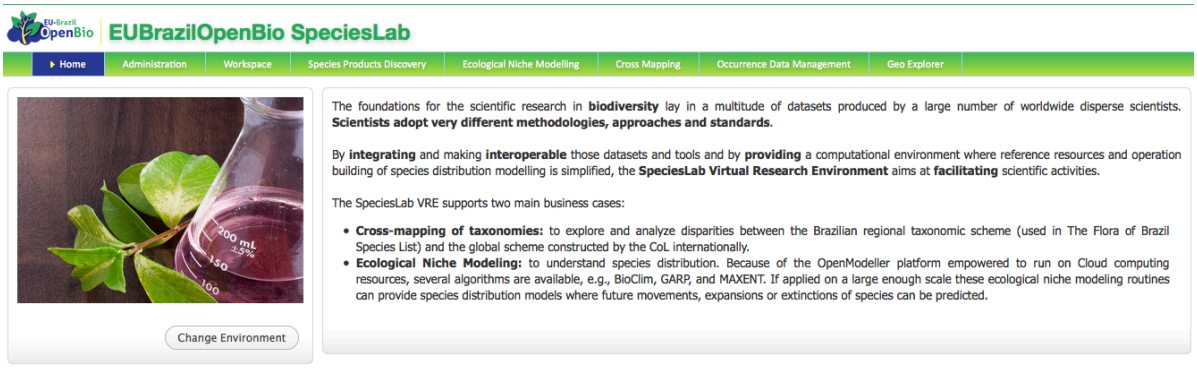

Figure 3. The SpeciesLab VRE Homepage

The main facilities this VRE offers are:

- ***Species Products Discovery***: to enable users to discover and manage species products (occurrence data and taxa names) from a number of heterogeneous providers in a seamless way.

  For discovery, the portlet supports the specification of search criteria based on species scientific name or common name as well as on the type of product the user is interested in, i.e. occurrence points or taxa names. In addition to that, the user can specify *(i)* specific data sources to be searched, *(ii)* a specific geographical area (via a bounding box) and *(iii)* a specific time interval. Results can be organized by classification, data provider, data source, and rank.

  For management, the portlet supports diverse facilities depending on the type of product to be managed. Regarding taxa names, the portlet makes it possible to have a detailed description of each selected name including the classification, to save the discovered objects into the workspace to use them in other contexts (e.g. taxa names comparison), to produce entire checklists of part of a classification by starting from a given taxa name. Regarding occurrence points, the portlet makes it possible to have a detailed description of the selected occurrence point datasets, to dynamically visualize the selected occurrence points on a map, to store the selected data in the workspace as to use them in future activities (e.g. niche modelling).

  Once discovered, objects can be stored into the workspace for future use;

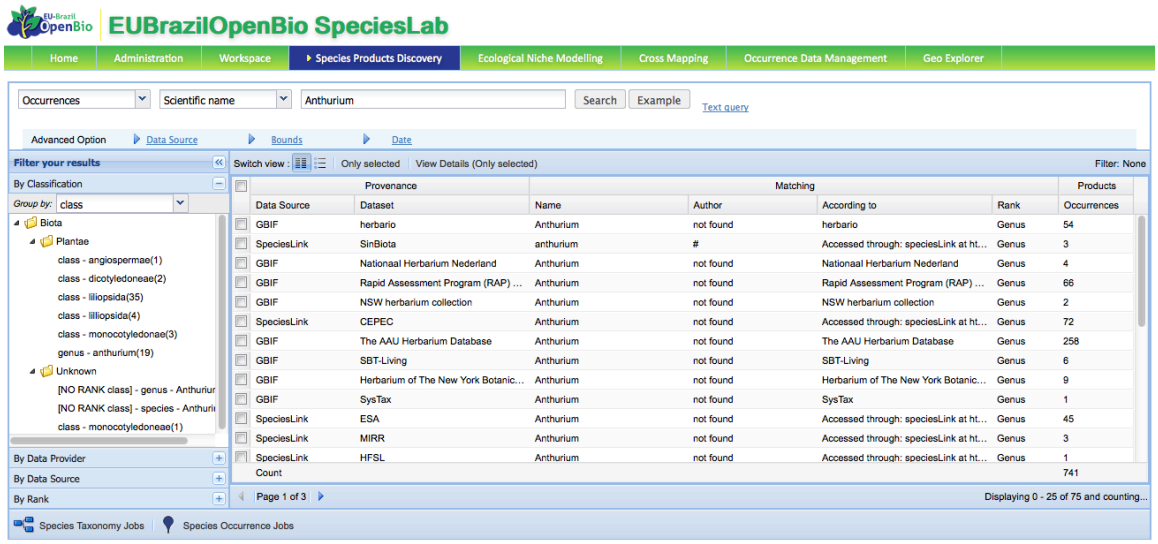

Figure 4. Species Products Discovery Facility Screenshot

- ***Ecological Niche Modelling:*** to enable users to define and manage ecological niche modelling tasks. These tasks consist of complex and computationally intensive model creation, testing, and projection activities. Users are provided with a feature-rich environment allowing them to characterise such tasks by specifying the data to be exploited (occurrence records and environmental parameters), the algorithms to use and the parameters to be considered during the testing phase. Tasks can be monitored after being submitted. Moreover, the results of each task can be easily accessed, visualised and saved into the user workspace for future use;

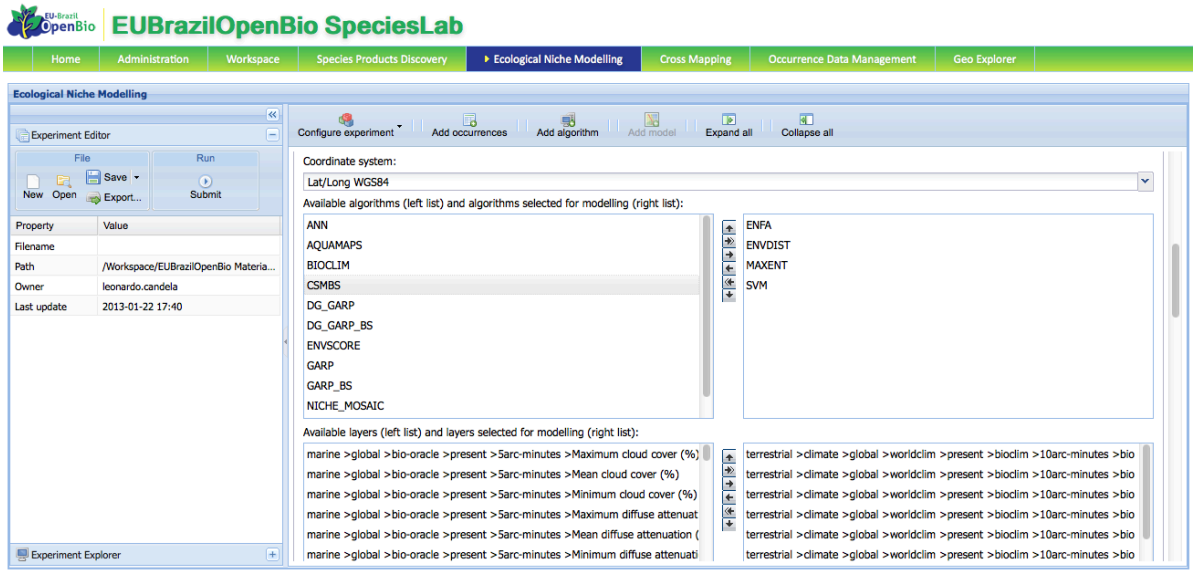

Figure 5. Ecological Niche Modelling Facility Screenshot

- ***Cross-mapping:*** to enable users to compare different checklists. Checklists can be either produced by relying on the Species Products Discovery or owned by the user. Moreover, they are expected to be in the Darwin Core Archive format. Users are provided with a feature rich environment enabling them to import checklists to be compared, to inspect the content of every checklist, to compare (a.k.a. cross-map) two checklists by using diverse comparisons mechanisms, to visualise the results of a comparison and to save it for future use;

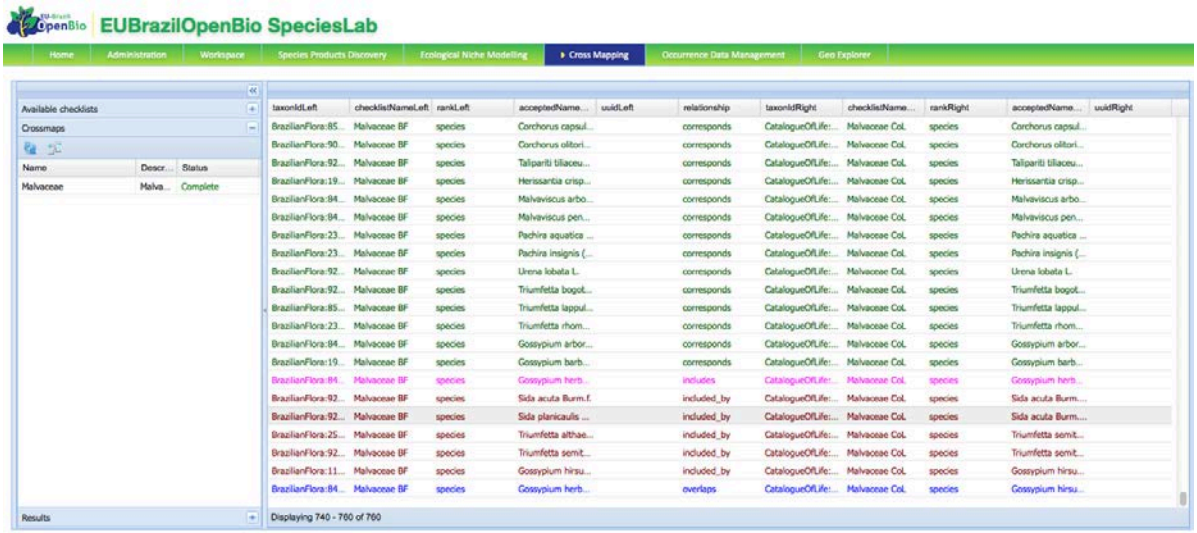

Figure 6. Cross-mapping Facilities Screenshot

- ***Workspace***: to enable every user to store and organise information objects for future use. In addition to that, the user is allowed to collaborate with other users by sharing objects and messages;

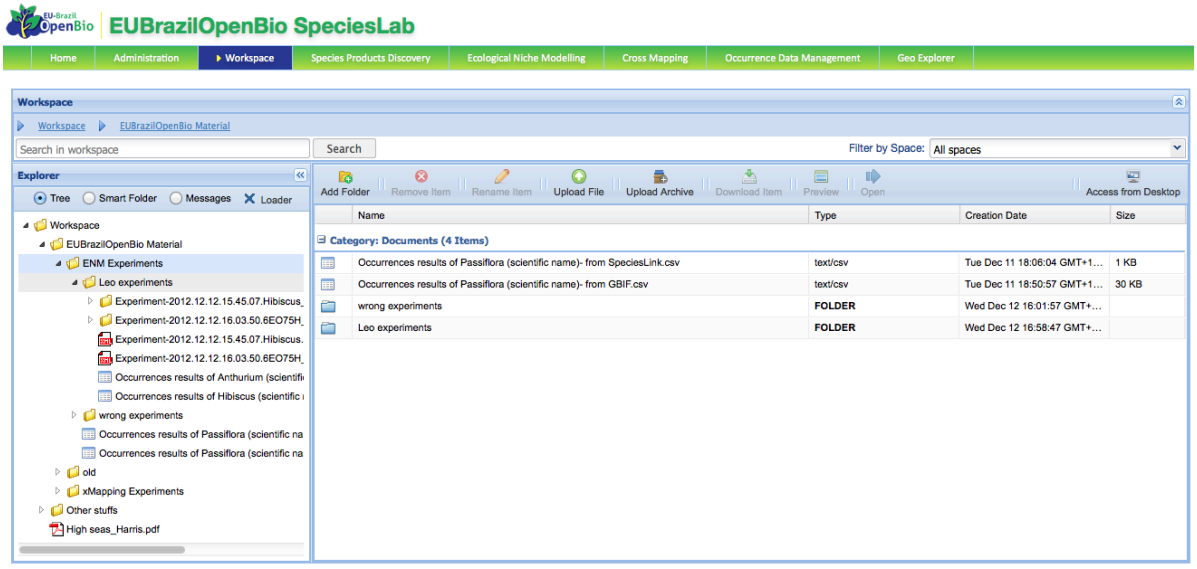

Figure 7. Workspace Facilities Screenshot

- ***Maps Visualisation***: to enable users to search, browse and visualise GIS layers available in the infrastructure independently of the repository that physically stores them. Among the available layers, there are maps produced via the Ecological Niche Modelling facilities, when the creators are willing to share them with the other VRE members;

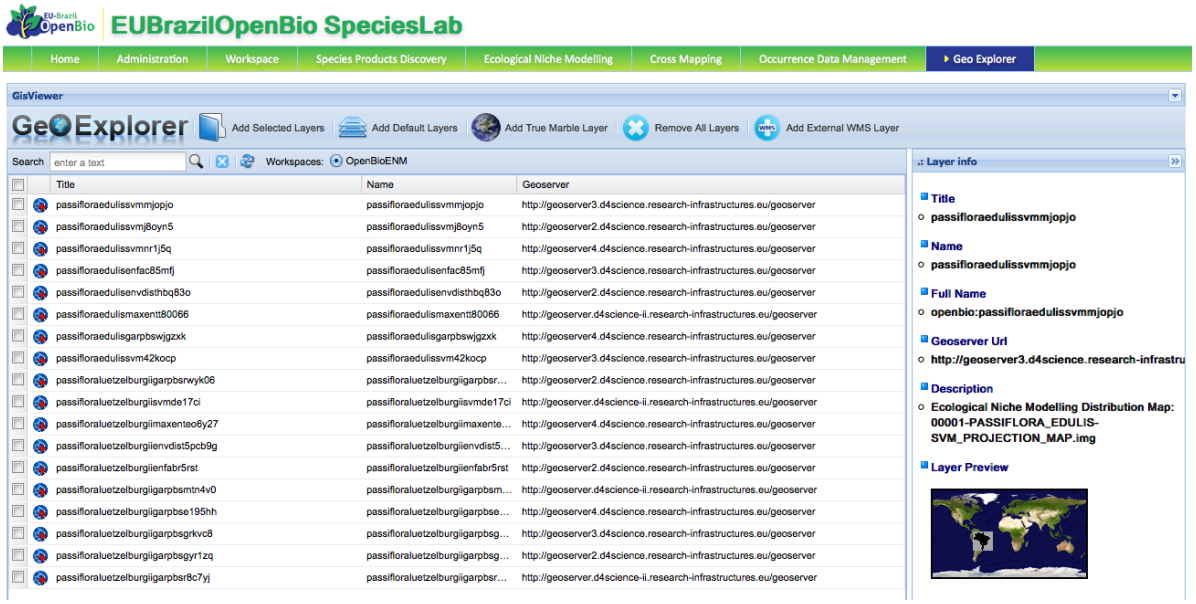

Figure 8. Maps Visualisation Facilities Screenshot

- ***VRE Management***: to enable authorised users (i.e. VRE Managers) to manage other users using or willing to access the VRE. VRE Managers can *(i)* authorise users in accessing the VRE, *(ii)* assign or withdraw roles to users, *(iii)* remove users, and *(iv)* send a communication to the current users.

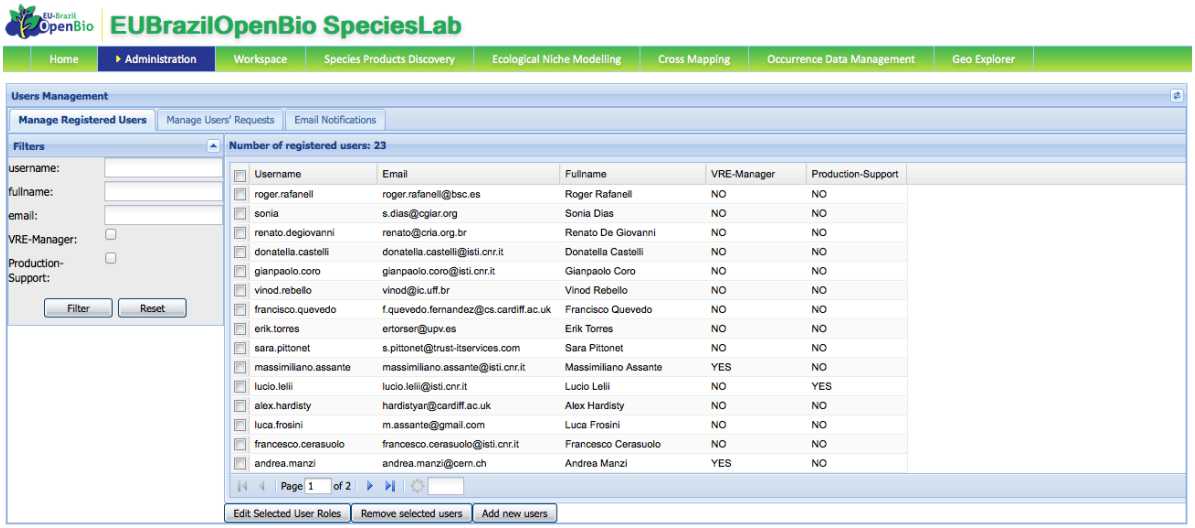

Figure 9. VRE Management Facilities Screenshot

2.2 The EUBrazilOpenBio File-oriented Data Storage Infrastructure

EUBrazilOpenBio caters for a file-oriented data storage facility by relying on both gCube storage facilities and Usto.re.

gCube storage facility is conceived to offer - via a POSIX-like interface - access to distributed storage systems supporting the access and storage of unstructured byte streams. In particular, it is conceived to support the organisation and operations normally associated with local file systems whilst offering scalable and fault-tolerant remote storage. In order to do that, it federates diverse storage back-end including MongoDB. This facility manifests in a thin software library that acts as a mediator with the storage backend instances currently existing in the infrastructure and offers a unifying view over them. Such a software library is expected to be used by clients to download, upload, remove, add, and list files. Files have owners and owners may define access rights to files, allowing private, public, or group-based access. Through the use of metadata, the library allows hierarchical organizations of the data against the potentially flat storage provided by the service's back-ends.

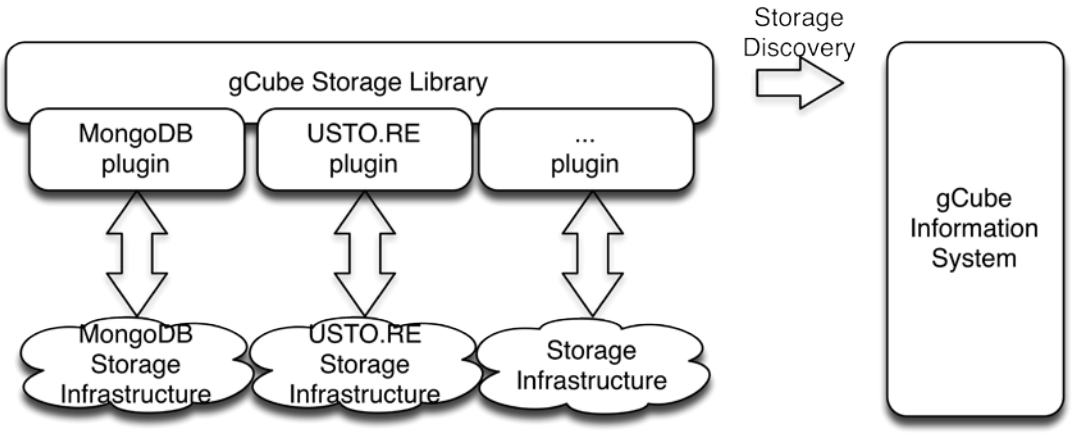

Figure 10 gCube Storage Library scheme

The Usto.re Cloud Storage Platform aims to provide an alternative storage solution for the EUBrazilOpenBio e-Infrastructure. It comprises features that focus on both user interactions through a seamless agent and web interface, as well as on resource integration with the e-Infrastructure components through an Application Programming Interface (API).

The main goal of the Usto.re is to provide important benefits of a Cloud Data Storage System to EUBrazilOpenBio e-infrastructure, as follows:

- **Scalability**: Using idle hardware resources of machines (hosts) connected to a network in a rational way.
- **Availability**: promoting a reliable system to deal with common variations in availability of hosts.
- **Security**: the adoption of appropriate security policies, techniques and strategies to provide privacy and user authentication.
- **Sustainability**: Operation even with small low cost resources or devices.
- **Ubiquity**: data access from web browsers, tablets, smartphones, through Application Program Interface, and software clients.

To promote the integration between gCube and Usto.re platform, a Representation State Transfer (REST) strategy was initially adopted based on principles used by industry leaders such as Amazon AWS, Google and Twitter. The REST API includes: user access, backup and restoration of files, file directory synchronization, user file management among other functionalities.

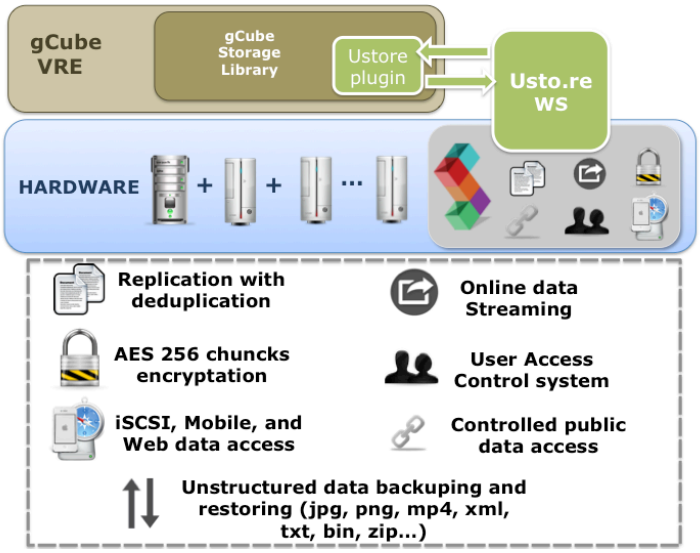

Figure 11 – Usto.re REST Web Service

In order to promote the needed communication between gCube and Usto.re REST API (Usto.re WS), a plugin was implemented following the interface standards (see Table 1) of gCube Storage Management Library.

Table 1 – gCube Storage Library implemented Interface

Relative PATH	Description
put(boolean replace)	Puts a local file in a remote directory. If replace is true then the remote file, if exist, will be replaced. If the remote directory does not exist it will be

	automatically created.
get()	Downloads a file from a remote path to a local directory.
remove()	Removes a remote file.
removeDir()	Removes a remote directory and all files present. This operation is recursive.
showDir()	Shows the content of a directory
lock()	Locks a remote file to prevent multiple accesses. Return an id string to unlock the file. Remember that every lock has a TTL associated (Default: 180000 ms).
unlock(String lockKey)	If the file is locked, unlocks the remote file.
getTTL()	Return the TimeToLive associated with a remote file if the file is currently locked.
setTTL(int milliseconds)	Reset the TimeToLive of a remote file that is currently locked.

After that, a testbed environment was provided for CNR, UFF and CESAR in order to perform the needed integration tests between Usto.re and gCube.

Nowadays, Usto.re Team is performing the needed changes to provide an Usto.re production version composed by gCube system requirements. In turn, that version will pass for new integration tests and active to production use.

2.3 Computing Services

The EUBrazilOpenBio infrastructure offers a number of computing services including COMPSs (cf. Sec. 2.3.1), EasyGrid AMS (cf. Sec. 2.3.2), VENUS-C (cf. Sec. 2.3.3), HTCondor (cf. Sec. 2.3.4), and gCube Hosting Node (cf. Sec. 2.3.5).

2.3.1 COMPSs Framework

COMPSs is a programming framework, composed of a programming model and an execution runtime which supports it, whose main objective is to ease the development of applications for distributed environments. On the one hand, the COMPSs programming model aims to keep the programmers unaware of the execution environment and parallelization details. They are only required to create a sequential application and specify which methods of the application code will be executed remotely. This selection is done by providing an annotated interface where these methods are declared with some metadata about them and their parameters. On the other hand, the COMPSs runtime is in charge of optimizing the performance of the application by exploiting its inherent concurrency. The runtime intercepts any call to a selected method creating a representative task and finding the data dependencies with all the previous ones that must be considered along the application run. The task is added to a task dependency graph as a new node and such dependencies are represented by edges of the graph. Tasks with no dependencies enter the scheduling step and are assigned to available resources. This decision is made according to a scheduling algorithm that takes into account data locality, task constraints and the workload of each node. According to this decision the input data for the scheduled task are transferred to the selected host and the task is remotely

submitted. Once a task finishes, the task dependency graph is updated, possibly resulting in new dependency-free tasks that can be scheduled.

Through the monitoring of the workload of the application, the runtime determines the excess/lack of resources and turns to cloud providers enforcing a dynamic management of the resource pool. In order to make COMPSs interoperable with different providers, a common interface is used, which implements the specific cloud provider API. Currently, there exist connectors for Amazon EC2, Azure and for providers that implement the Open Cloud Computing Interface (OCCI) and the Open Virtualization Format (OVF) specifications for resource management.

2.3.2 EasyGrid AMS

The EasyGrid middleware is an Application Management System (AMS) that provides several mechanisms to manage the execution of applications in distributed systems. As these computing systems become larger in scale they become ever more heterogeneous, exhibit more variable and constant changes in behaviour, and are more likely to be shared by competing applications. Designing applications to extract good performance from these kinds of platforms can be extremely complex. The goal of the EasyGrid AMS is to free programmers from this arduous effort and the need to modify or tune the parallel application to each type of system on which the application should execute. Specifically, the EasyGrid AMS is able to oversee the efficient execution of tasks of a MPI application (with or without precedence constraints) by configuring and statically scheduling tasks, prior to the execution, based on the characteristics of the application and the predicted availability/performance of the target resources. During the execution itself, the EasyGrid AMS astutely employs dynamic scheduling and fault tolerance mechanisms to constantly adjust the execution and reduce the chance of failure. Furthermore, the application granularity may be reconfigured at runtime to further improve performance, often without stopping the execution. By incorporating the AMS into the application, it becomes self-managing or autonomic [8], exhibiting features such as self-optimisation, self-healing and self-configuration. This can be especially beneficial in dynamic, shared, heterogeneous computing environments, like clouds, for example.

The execution of the MPI application is controlled by additional management processes. The design of each management process is based on a subsumption architecture which consists of layers of tasks achieving behaviours. Each layer adds a new level of competence with higher levels subsuming the roles of lower levels when they wish to take control. The process management behavioural layer is responsible for the dynamic creation of MPI processes and the routing of messages between processes. The dynamic scheduling and fault tolerance layers utilize status information provided periodically by the monitoring layer, to decide if, and when, it is necessary to activate a re-scheduling mechanism and to detect and treat process and resource failures.

2.3.3 VENUS-C Middleware

The Programming Model Enactment Service (PMES) is a bridge to the VENUS-C platform, allowing the execution of applications programmed through the COMPSs programming model. A client is used to contact an OGF BES compliant Web Service in order to submit the execution of a COMPSs application. This request is expressed through a JSDL document containing the application name, the input parameters and data references.

The implementation of the COMPSs BES includes a Job Manager and a Job Dispatcher that actually execute the application and manage its life cycle. The submission request is received by that Job Manager that creates an enactment service job object assigning it a Universally Unique Identifier (UUID); the job is created and queued in the Job Dispatcher which is responsible for dealing with execution. The Job Dispatcher maintains a user resizable pool of threads that is responsible for picking jobs from a Job Dispatcher queue filled by the Job Manager. First, a virtual machine is

requested to the Cloud Provider in order to deploy the COMPSs runtime that schedule the tasks on the remote nodes and the application, downloading its package from the Cloud storage. Once the runtime machine is deployed the COMPSs application is remotely started. In its turn, the COMPSs runtime will schedule the tasks on a set of machines created on demand. This phase includes the staging of input files from remote locations as specified in the execution request.

2.3.4 HTCondor

Condor, recently renamed HTCondor, is a workload management system for compute-intensive jobs on clusters and wide-area distributed systems of either dedicated or shared resources. These resources form an HTCondor *pool*, comprised of a central manager and an arbitrary number of machines. The role of HTCondor is to match waiting job requests with available resources (machines). All HTCondor services send periodic updates to the central manager, who acts as a centralised repository of information about the state of the pool. Using this state information, the central manager assesses the current state of the pool and tries to match pending requests with the appropriate resources. The ClassAd mechanism provides a flexible framework for matching resource requests (jobs) with resource offers (machines). Jobs can easily state both job requirements and job preferences. Likewise, machines can specify requirements and preferences about the jobs they are willing to run. Each resource has an owner, the one who sets the policy for the use of the machine, including when HTCondor requests will be accepted. Job submission to HTCondor requires a *job submission specification* to define which program to run and resource requirements to run it.

2.3.5 gCube Hosting Node

By relying on gCube technology, in particular on the gCube Hosting Node component, the EUBrazilOpenBio infrastructure delivers a distributed computing platform catering for the deployment of applications including Web Services. In essence, a gCube Hosting Node is a unit of any gCube-based infrastructure that can be used to host a service and offers a number of facilities for interfacing the hosted service with the rest of the infrastructure.

By relying on gHNs, gCube offers facilities for dynamically (un-)deploying application instances. These application instances are automatically added to the infrastructure and their status can be monitored by using the gCube Information System, either in a programmatic way or via a graphical user interface (cf. Figure 2). At the time of writing this report, the EUBrazilOpenBio infrastructure offers more than 45 gHNs. An up to date picture can be acquired via the monitoring tool[1].

This facility makes it possible to create a scenario that promotes service-oriented horizontal scalability, i.e. new service instances are dynamically created and the load can be distributed among a larger number of instances.

[1] https://portal.eubrazilopenbio.d4science.org/web/guest/monitor

3 Enhancements and Optimisations

This section details the enhancement and optimisation activities performed in order to deploy the project use cases on the production e-Infrastructure through the adoption of distributed computing technologies, programming models and management paradigms.

As depicted in Figure 12, the EUBrazilOpenBio e-Infrastructure provides the resources to allow scientists to access the Use Case I and II tools through a Web interface. Through a specific GUI in the biodiversity VRE, Use Case I demonstrates, given two taxonomic checklists in the Darwin Core Archive (dwc-a) format, how a scientist can import those checklists and run a cross-map experiment between them, obtaining the relationships that exist between the taxa in one checklist with the taxa in the other. Through its GUI, Use Case II allows scientists to choose a set of species, develop ecological niche models based on occurrence data (using the different algorithms available), test them and project the models given different environmental conditions. In addition, services exist to allow scientists to use their own data, and save or share the results through their own workspace within the VRE. A couple of data discovery services can also be used to obtain data from respected biodiversity repositories and databases.

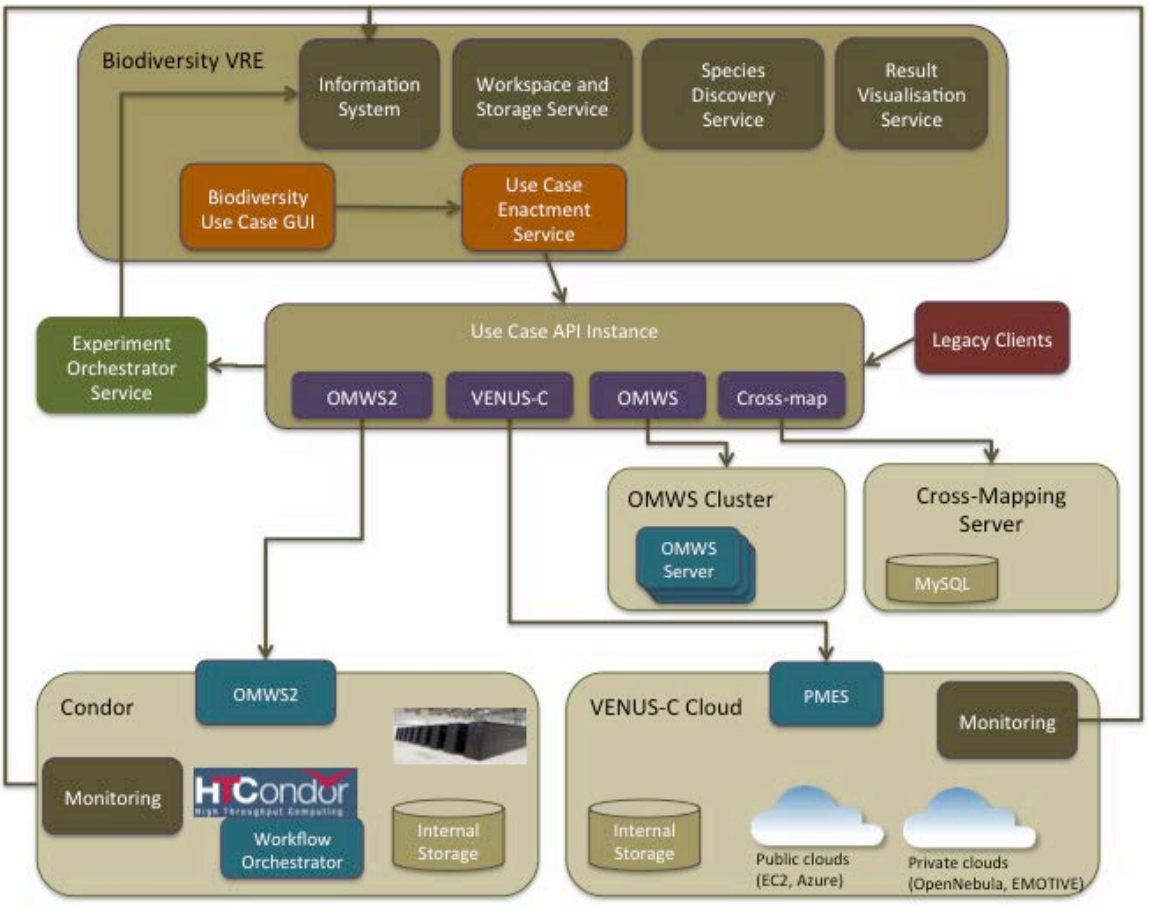

Figure 12 – Implementation of the use cases on the EUBrazilOpenBio Infrastructure

For each Use Case, a specific Use Case Web Service is responsible for implementing an API that allows both the EUBrazilOpenBio VRE and existing Use Case Clients (GUIs or other software tools)

to interact with the infrastructure's back-end computing resources transparently. The service is integrated with an Experiment Orchestrator Service (EOS) that provides meta-scheduler functionalities to select the best resource to execute an application request. Such a decision depends on multiple factors such as the load of each infrastructure, the mean execution time of the application on each infrastructure, the infrastructure availability, the availability of input layers in the local repository, the availability of openModeller algorithms and execution time forecasting. A Job Resources Optimizer (JRO) tries to optimize the resources usage on the chosen resource provider by analysing the application topology and trying to guess which are the computational needs to obtain the best performance. The EOS service relies on the JRO for setting up the number of needed computing units, and its specifications in case of being virtual resources. The EOS retrieves the information from the Information Service, filled with data provided by each computational backend.

Access to the computing resources is managed by specific plugins. Three plugins have been provided: one for the VENUS-C middleware allows the connection to the Programming Model Enactment Service used to schedule COMPSs applications to virtual instances provided by private and public clouds; a OMWS2 plugin allows the execution of jobs across a pool of working nodes managed by HTCondor; and a specific plugin to allow compatibility with existing OMWS servers.

3.1 Use Case I implementation

3.1.1 Initial architecture: i4Life project

At the beginning of the project, the cross-mapper tool developed inside the i4Life project consisted of 2 main modules:

- A perl module in charge of importing checklists in dwc-a format into a relational database.
- A set of php pages, which apart from being the GUI of the application, they also contained the business logic of the application.

Both modules interacted with a MySQL database that was used as a persistent storage. That database was used for storing the data (checklist and cross-map) as well as the state of the "service" (e.g., indicating which checklists have been imported and which cross-maps have been executed, etc.)

The 4 main limitations of that approach were:

a) It didn't offer a service in a way that other applications could call programmatically, so it couldn't be run inside of an external workflow or called by a different GUI. Instead, the cross-mapper tool must be executed through its web site by clicking buttons in different pages.
b) Even if a web service interface has been created for programmatic access to the cross-mapping tool, the languages in which it was developed (perl and php) didn't facilitate the deployment of the service in the gCube infrastructure and/or future improvements like the use of COMPSs.
c) The main computational workload resided in the SQL queries that were executed in the DB engine.
d) Finally, the state of the service was kept in the database. This peculiarity implied that although the system could have multiples instances of the program in different machines, unless they were connected to the same database (with the potential concurrency problems), the user had to call always to the same instance to see, for example, the previously imported checklists.

To address the first 2 issues, it was decided to migrate the code to java. During that process, a deep study of the legacy code was carried out, detecting potential bugs and bottlenecks. The result of it has helped to improve the old code as well as making the new code faster and more robust. Also during the migration, new features were added to provide extra functionalities that didn't exist in the legacy application.

3.1.2 Current architecture

The current architecture of the UC-I consists basically of 2 independent components. On one hand, we have a SOAP web service which exposes a set of methods that allows a client to run cross-map experiments. And on the other hand, we have a GUI interface that interacts with the cross-map service alongside other services and tools provided by infrastructure.

Internally, the cross-map service was developed using a top-down approach and it is formed by a set of java projects, each one covering one aspect of the application. For example, there is a service interface project where the public interface of the service is defined (using a WSDL file). There is also another java project that contains the business logic and which could be called from the command line without the need of the web-service façade. The façade is created by another java project, using the Tuscany SCA framework, to easily expose the business logic as a web service or any other protocol. Finally, there are other several java projects for covering internal aspects like accessing the database using DAOs, etc.

The other part of the system, the GUI, was developed jointly by CNR and Cardiff, and it is basically a GWT project that uses the GXT widget library. It has been adapted to be deployed as a liferay portlet inside the gCube portal and internally it interacts with the cross-map service and with other systems from gCube, such as the gCube information system and the gCube workspace.

Deployment configuration:

The deployment artefacts of the 2 components previously described are war files. Each war file is deployed in the infrastructure as follows:

- The GUI is deployed as a portlet inside EUBrazilOpenBio portal. The administrator of the portal will assign it into a Virtual Organization (VO) and inside the VO into a Virtual Research Environment (VRE). It is through this VRE where the user can interact with the portlet. The portlet internally queries the Information System to get the URI of the external cross-map web service it has to access and also interacts with the workspace in order to read and write files from it.
- The cross-map web service, which at the moment is deployed in only one external machine, is registered into the Information System as an external resource. The instance of the cross-map web service is running on a server in Cardiff called *litchi1*. The cross-map service, as its predecessor did in the i4life project, connects with a MySQL database in which the data and the state of the service are persistent.

This basic deployment configuration is depicted in Figure 13

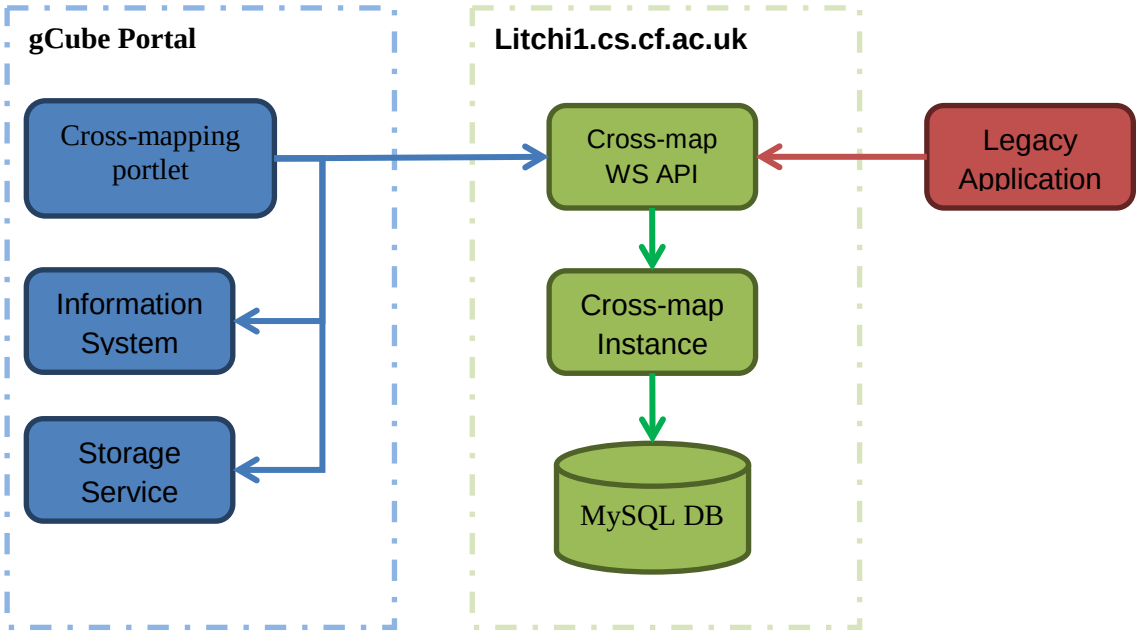

Figure 13: Current architecture deployment for UCI

The main benefit of this simple architecture is that it can be used as a Proof of Concept to show that it is possible the re-use external services already implemented by third-parties to give them an added value:

- The web service itself probably will not require any modification or perhaps minor ones and the main effort will be in the creation of a GUI. Task that is relatively simple following the specifications about how to create a gCube portlet.
- Once the portlet has been created, it will not only serve as a simple client of the web service, but it can also interact with other gCube services, like the workspace. This allows users to utilise files that have been created by other applications. For example, for the cross-mapping, users can use as input the dwc-a files that have been obtained through the Species Product Discovery.
- Finally because the portlet is deployed inside a portal. The user can run from the same web browser a more complex experiment in which the external web service is only a part of it.

The final architecture for the cross-mapper does not yet take advantage of the computational and storage resources offered by the EUBrazilOpenBio infrastructure.

The possibility of enhancing the performance of this use case by means of using distributed computing has been explored, however after an initial evaluation it was considered that such an implementation within the context and the time frame of this project would incur overheads that would outweigh the benefits.

3.2 Use Case II implementation

This use case addresses computational issues when Ecological Niche Modelling needs to be used with a large number of species by means of complex modelling strategies involving several algorithms and high-resolution environmental data. The use case is based on the requirements of the

Brazilian Virtual Herbarium (BVH) of Flora and Fungi, which has its own system capable of interacting with an openModeller Web Service (OMWS) to carry out a standard niche modelling procedure for plant species that are native to Brazil. Species that can be modelled by BVH come from the List of Species of the Brazilian Flora, which currently contains ~40.000 entries. The corresponding occurrence points are retrieved from speciesLink - a network that integrates data from distributed biological collections, currently serving almost 4 million records considering only plants. For species with at least 20 occurrence points available, the standard modelling procedure involves generating individual models with five different techniques: Ecological-Niche Factor Analysis, GARP Best Subsets, Mahalanobis distance, Maxent and One-class Support Vector Machines. Besides generating the models, a 10-fold cross-validation is performed to assess model quality and a final step is needed to merge the individual models into a single *consensus* model, which is then projected into the present environmental conditions for Brazil in high-resolution.

As described in the performance profiling section of the previous report (D4.2), the entire process is computing-intensive, especially when dealing with a high number of species. This section reports the developments for the implementation of the use case in the EUBrazilOpenBio infrastructure.

The implementation of the use case includes the development of an API to support the execution of complex experiments through the extension of the existing interface and several optimizations through the implementation of OM workflows in COMPSs executed in cloud resources managed by the VENUS-C middleware, as well as enhancements with HTCondor and a further optimization of openModeller core through EasyGrid AMS.

3.2.1 OMWS2

The original openModeller XML Schemes are comprised of an XML schema[2] defining all elements, attributes, structure and data types that are used to serialize openModeller objects, and a WSDL file defining the operations of the OMWS API[3]. Each operation defined by this scheme supports the execution of one simple action with openModeller, for example, to create, test or project a model. When a user wants to perform several actions with the same dataset, it is necessary to submit each operation to the service separately. For example, to create five models with the same species occurrence dataset using five different modelling algorithms, five different requests are needed (one per algorithm). The same occurs for experiments that create models for different species using the same modelling algorithm. When there are dependencies between the operations, for example, creating a model and then projecting it into an environmental scenario, the client is responsible for sending the initial request, monitoring and retrieving the results of the operation that creates the model and also for including the serialized model as an input parameter in the projection operation that follows.

Another characteristic of OMWS is that it doesn't provide support for user sessions. Instead, every operation returns a ticket that the user has to store and to use with any other subsequent operation that depends on the results of the previous operation, such as monitoring and retrieving the results of that operation.

In order to better handle a high number of niche modelling tasks, which is beyond the common use of niche modelling applications, a new version of the openModeller XML Scheme was created to allow the execution of multi-staging, multi-parametric experiments (Tables 1 and 2).

[2] The openModeller XML Scheme: http://openmodeller.cria.org.br/xml/1.0/openModeller.xsd
[3] The openModeller WSDL: http://openmodeller.cria.org.br/ws/1.0/openModeller.wsdl

Element	Scope
AvailableAlgorithms	openModeller
Algorithms	openModeller
ModelParameters	openModeller
ModelEnvelope	openModeller
TestParameters	openModeller
TestResultEnvelope	openModeller
SerializedModel	openModeller
ProjectionParameters	openModeller
ProjectionEnvelope	openModeller
ExperimentParameters	openModeller 2.0
AbstractJob (CreateModelJob, TestModelJob and ProjecModelJob)	openModeller 2.0
ExperimentTickets	openModeller 2.0
ResultSet	openModeller 2.0

Table 2 - Extended openModeller 2.0 Formats

Table 1 shows the general types defined in the extended openModeller format[4]. One of the objectives of the new types is to support experiment pipelines involving point sampling, model creation, model testing and/or model projection tasks, in addition to the legacy unique-staging experiments.

Any of these experiments will produce only one request, reducing the number of interactions with the server, simplifying client logic, and allowing server implementations to use parallelization techniques. The second advantage over traditional OMWS is that these types support the following multi-parametric requests per experiment:

1. Several species occurrences datasets
2. Several algorithm definitions
3. Any combination of several species and several algorithms

OMWS uses the openModeller XML Scheme to define a Web service interface to openModeller[5]. The existing openModeller Server[6] provides an implementation of OMWS that can be configured to use HTCondor, distributing the jobs across a pool of working nodes.

[4] The new openModeller XML Scheme: http://openmodeller.cria.org.br/xml/2.0/openModeller.xsd
[5] The original openModeller Web Service: http://openmodeller.sf.net/web_service.html
[6] The original openModeller Server: http://openmodeller.sf.net/om_server.html

Operation	Scope
ping	OMWS
getAlgorithms	OMWS
getLayers	OMWS
createModel	OMWS
getModel	OMWS
testModel	OMWS
getTestResult	OMWS
projectModel	OMWS
getProgress	OMWS
getLog	OMWS
getLayerAsAttachement	OMWS
getLayerAsUrl	OMWS
getProjectionMetadata	OMWS
runExperiment	OMWS2
getResults	OMWS2
cancel	OMWS2

Table 3 – OMWS2 OMWS+ interface.

Table 2 shows the operations defined in the new openModeller Web Service interface (2.0)[7]. To maximize the usage of the new OMWS service instance provided by EUBrazilOpenBio, it was decided that the instance should fulfil the following requirements:

1. Provide legacy ecological niche modelling operations (i.e., the same as those defined by OMWS 1.0).
2. Provide new operations that support multi-staging, multi-parametric experiments.

Operation	Parameters	Result
RunExperiment	ExperimentParameters	ExperimentTickets (one for each job)
getProgress	Tickets (comma-separated list of strings)	Progresses (comma-separated list of progresses)
getResults	Tickets (comma-separated list of strings)	ResultSet
cancel	Tickets (comma-separated list of strings)	Cancelled tickets (comma-separated list of strings)

Table 4 – OMWS 2.0 operations

[7] The new openModeller WSDL: http://openmodeller.cria.org.br/ws/2.0/openModeller.wsdl

Table 3 shows the extended methods with the corresponding inputs and outputs.

The type “ExperimentParameters” consists of a section containing basic data needed by the jobs, such as environmental layers, occurrence points, algorithm parameters and models, followed by another section specifying each job (CreateModelJob, TestModelJob or ProjectModelJob). The corresponding “ResultSet” type can accommodate any sequence of model creation, testing or projection result types. The existing "getProgress" operation was extended to accept a comma-separated list of tickets, returning a comma-separated list of progresses. Jobs can also be cancelled now through the “cancel” operation.

Although most operations in OMWS 2.0 are the same ones that can be found in OMWS 1.0, the amount of changes justified a new namespace for both the XML Schema and the WSDL, and therefore OMWS 2.0 is not backward compatible with the original OMWS interface. Legacy methods, such as “createModel”, “testModel” or “projectModel” can still be used in the same way that they are used through OMWS, but under a new namespace. This way, legacy clients are kept aware of the differences between the two versions but can migrate to the new protocol with little modification. At the same time, the OMWS server provided by EUBrazilOpenBio is compatible with both the 1.0 and the 2.0 protocol versions, allowing legacy and new clients to seamlessly interact with it.

The new version of the protocol[8] will become official and soon be released as part of openModeller 1.4, which will also include a standard server implementation[9] in C++ and a simple command-line client in Perl.

3.2.2 openModeller Enhancements with COMPSs in the cloud

The proposed extension of the openModeller Web Service API provides a way to automatically convert multi-stage & multi-parameter experiments into a set of single legacy operations supported by the openModeller command-line suite. This is implemented through COMPSs, which orchestrates the execution after automatically generating the execution graph (Figure 8).

As depicted in Figure 11, the COMPSs Job Manager (implemented in the VENUS-C PMES), receives the execution requests from the Orchestrator that dispatches user’s requests received from the OMWS2 interface to support multi-staging and multi-parametric experiments through COMPSs and openModeller.

The ENM workflow is composed by the following operations:

- **Convert:** converts a multi-stage and multi-parameter request into a set of single operation calls.
- **Model:** models the species distribution based on a chosen algorithm, a set of occurrence points and a set of environmental layers.
- **Test:** tests the model against a set of reference occurrence points.
- **Project:** projects the species model into an environmental scenario, generating a potential distribution map.
- **Translate:** formats and colours the projected map into a viewable image.

When a *RunExperiment* document is received, the *Convert* method splits the experiment request into a set of single-operation requests. Each request handles a single species-algorithm pair.

The *Model* operations start computing each request, thus blocking the following executions of the *Test* and *Project* operations, which depend on it.

[8] The new openModeller Web Service: http://openmodeller.sf.net/web_service_2.html
[9] The new openModeller Server: http://openmodeller.sf.net/om_server_2.html

The *Translate* operation is also queued, and starts when the projected map is available, colouring it with a provided palette and eventually converting it to a desired image format as depicted in Figure 15. An example resulting execution graph managed by COMPSs is depicted in Figure 14.

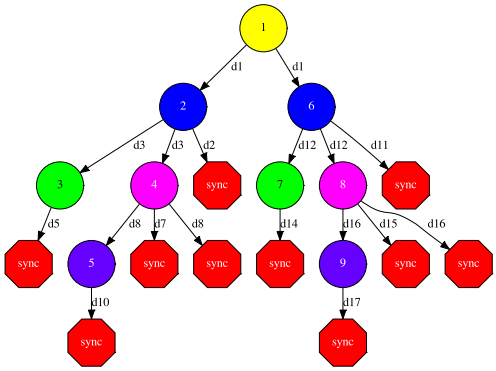

Figure 14 - ENM COMPSs workflow with 1 species and 2 algorithms

The simplified snippet shown in Figure 15 describes the ENM COMPSs workflow code which consists of a regular sequential code, Figure 15(a), where each previously described method is invoked depending on the operation request types, and on a Java annotated interface which provides to the COMPSs runtime library the method signatures that must be orchestrated and executed remotely, Figure 15(b).

```
//For each pair species-algorithm request
for (String rqstPath : speciesRqst){
        SerializedModelType serModel = null;
        ...
        if(modelRqst.exists())
        serModel = MImpl.om model(modelRqst, modelOut, ...);

        if(testRqst.exists()){
        MImpl.om test(modelRqst, serModel, testRes, ...);
        if(modelRqst.exists())
        MImpl.om test(testRqst, serModel, testRes, ...);
        else
        ...
}
if(projRqst.exists()){
        if(modelRqst.exists()){
        MImpl.om proj(projRqst, serModel, stats, map, ...);
        MImpl.proj clrTrl(map, palette, output, format, ...);
        }
        else{
        ...
        }
    }
}
```

(a)

```
public interface ModellerItf {

@Method(declaringClass = "modeller.MImpl")
SerializedModelType om_model(
        @Parameter(type = Type.FILE, direction = Direction.IN)
        String modelRqst,
        @Parameter(type = Type.FILE, direction = Direction.OUT)
        String serModel,
        @Parameter(type = Type.STRING direction = Direction.IN)
        String binaryArgs,
        @Parameter(type = Type.BOOLEAN direction = Direction.IN)
```

```
        boolean debugFlag
);

@Method(declaringClass = "modeller.MImpl")
void om_test(
        @Parameter(type = Type.FILE, direction = Direction.IN)
        String testRqst,
        @Parameter(type = Type.OBJECT, direction = Direction.IN)
        SerializedModelType serModel,
        @Parameter(type = Type.FILE, direction = Direction.OUT)
        String result,
        @Parameter(type = Type.STRING, direction = Direction.IN)
        String binaryArgs,
        @Parameter(type = Type.BOOLEAN, direction = Direction.IN)
        boolean debugFlag
);
…
}
```

(b)

Figure 15 - ENM workflow main code (a) and its associated COMPSs interface (b)

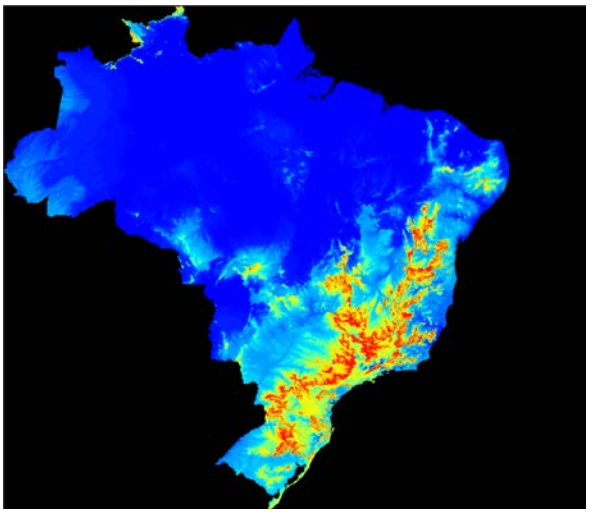

Figure 16 - Potential distribution of *Passiflora amethystina* obtained with the ENFA algorithm

The extensions implemented to enable the execution of complex openModeller workflows are backwards compatible with the original OMWS specification, allowing legacy clients to be fully supported in the new implementation and, therefore, still able of submitting experiments to the execution resources without using the graphical user interface developed by the project. In this case the execution on cloud resources has been optimized through the adoption of pre-deployed virtual instances where to run single openModeller operations. This feature is available by the COMPSs-PMES service that can be configured to boot a configurable number of VMs on the provider where the service is deployed; this solution allows serving these kinds of requests in a reasonable time avoiding the overhead of VM creation; if the number of requests exceeds the available resources, the service is still able to dynamically deploy new instances in order to cope with the burst of load. This version of the ENM service has been provided to the BioVeL project for the evaluation of the execution of workflows in the EGI Federated Cloud[10].

[10] http://www.eubrazilopenbio.eu/Content/News.aspx?id=bfe1f163-fb53-4a22-a67e-6f1a718b2389

3.2.2.1 Evaluation

The BSC testbed previously described has been used with a total of 10 quad-core virtual instances with 2GB of memory and 1GB of disk space running a Debian Squeeze Linux distribution. The aim of these tests is to validate the ENM-COMPSs workflow implementation, evaluate the advantage of the elasticity features of the COMPSs runtime, and compare the execution on a dynamically provided pool of virtual resources with a run on a pre-deployed and static virtual environment (Grid-like scenario).

Eight species of the genus *Passiflora*, each one with more than 20 occurrence points, were used to test the application. Models were generated using the following high resolution environmental layers from WorldClim: mean diurnal temperature range, maximum temperature of warmest month, minimum temperature of coldest month, precipitation of wettest quarter, precipitation of driest quarter, precipitation of warmest quarter, precipitation of coldest quarter and altitude. A simplified standard procedure consisting of model creation followed by an internal model test (confusion matrix and ROC curve calculation with the same input points) and a native model projection (with the same environmental layers) followed by a final image transformation was used for each species with a set of three algorithms used by BVH (SVM, ENVDIST and ENFA) called with a different set of parameters. The Brazilian territory served as a mask in all operations. This scenario composes a total of 46 simultaneous single operation requests.

Figure 17 depicts the evolution on the number of virtual machines used along the execution, highlighting how the COMPSs runtime is sensitive to the tasks load, adapting the number of current resources to the demand. After the initial scale-up phase, the load remains constant until the end of the process is reached, thus starting to free resources progressively. This is not the case of the static execution scenario (overlapped on the figure), which despite being 18.4% faster than the dynamic approach, is more expensive from a cost point of view because of the continuous usage of idle resources.

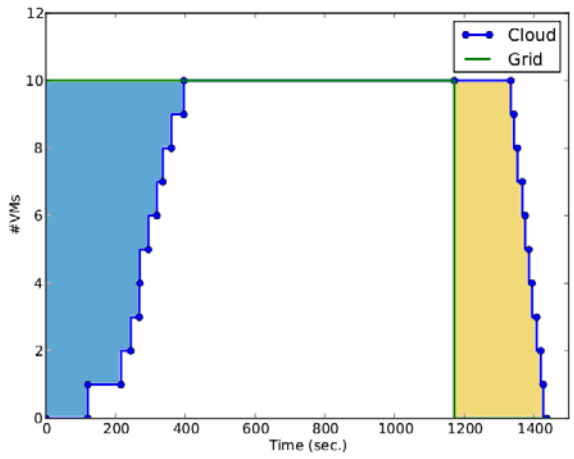

Figure 17 - Elasticity graph

Table 4 compiles the execution time and speedup of running the presented experiment on both configurations limiting the maximum number of virtual machines that the system could use. As it can be observed, the speedup is moderate because the application does not offer a high degree of parallelism (Figure 16). Despite this, COMPSs reaches good performance running on an on-demand provided environment (with an average performance loss around 9.6%), with a mean serving time of the Cloud middleware of about 120 seconds per VM. However, the speedup is not dramatically penalized, and the resources management is generally improved by reducing the overall execution costs.

#VMs	#Cores	Cloud		Grid	
		Time	Speedup	Time	Speedup
1	4	02:00:21	1.00	01:46:9	1.00
2	8	01:00:47	1.98	00:53:23	1.97
4	16	00:33:52	3.55	00:31:06	3.38
8	32	00:25:16	4.76	00:18:03	5.82
10	40	00:23:57	5.02	00:19:32	5.38

Table 5 - Execution times of grid and cloud executions

3.2.3 openModeller Enhancements with HTCondor

Virtualization is a popular approach to increase resource utilisation, achieving computational resource (memory, processing and storage) scalability while at the same time attempt to reduce power consumption. On the other hand, the use of virtualization can incur performance overheads. As well as highlighting the ability of the EUBrazilOpenBio architecture to incorporate HTCondor managed systems into the infrastructure, a number of architectural options and configurations were investigated to meet some design objectives: This OM service can be deployed dynamically; it supports elasticity for on demand computing by being implemented to make use of cloud resources; can cope with resource heterogeneity using both physical and virtual machines; and the ideal data placement was identified.

The proposed architecture to realise the said objectives and support the operations of UCII is shown in Figure 18. Two configurations of physical machines were used, on which HTCondor was installed, to evaluate the infrastructure composed of virtual and real machines capable of executing openModeller, in accordance with the submitted requests.

1. Host Type 1 (HT1)
   - Dual processors: Intel Xeon CPU 3.06GHz with Cache size (L2): 512 KB
   - Memory 2,5 GB DDR1 (266 MHz)
   - Disks: 10K RPM SCSI
2. Host Type 2 (HT2)
   - Dual hexacore processors: Intel Xeon CPU X5650 2.66GHz with Cache size (L3 per processor): 12288 KB and Cache L2: 256 KB per core
   - Memory 24 GB DDR3 (1333 MHz)
   - Disks: SATA II 7200 RPM

Machines of Host Type 2, with virtualization support in hardware, were installed with CentOS 6.2 and KVM (Kernel-based Virtual Machine) to provide full virtualization for VMs running unmodified CentOS Linux images. The data stores are used to hold the environmental layers.

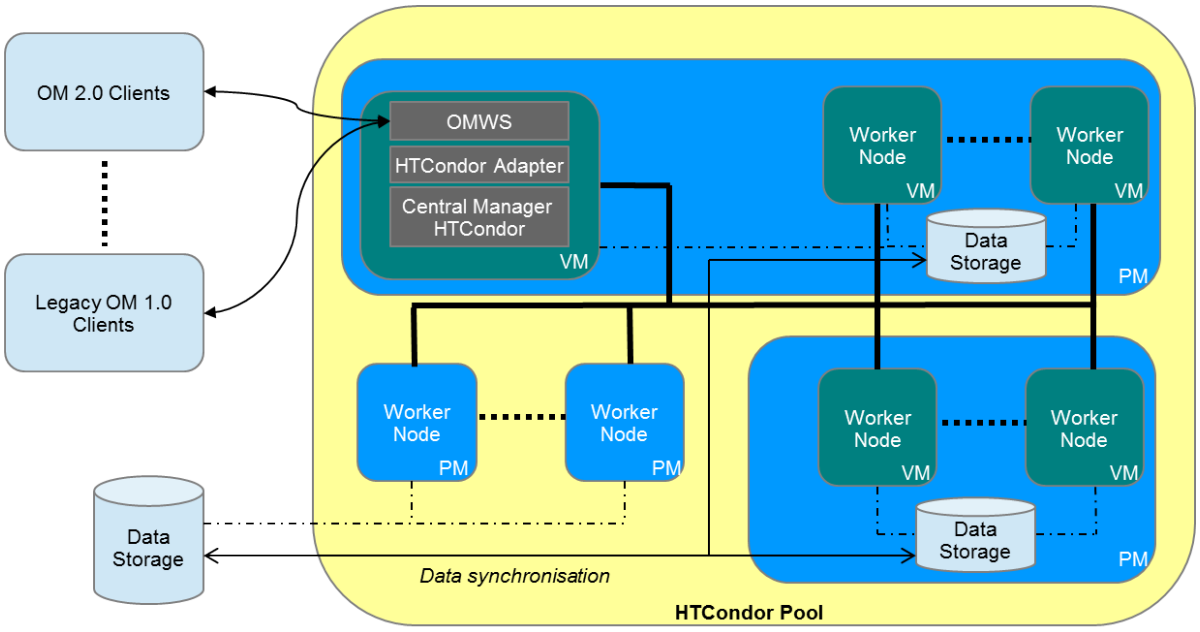

Figure 18 - HTCondor Pool Configuration

The HTCondor pool is comprised of a *central manager* and an arbitrary number of workers. This pool can be composed by a set of virtual machines, which are initialized or shutdown on demand autonomically, as well as a set of physical machines. Conceptually, the pool is a collection of resources (machines) and resource requests (jobs). The role of HTCondor is to perform the match-making between the waiting requests with available resources. Workers may join and leave the pool at any time but, while a member, send periodic updates to the central manager, which is the centralised repository for information about the state of the pool.

The central manager is implemented in a VM and hosts the openModeller Web Service (OMWS) server. Currently, OM clients can interact with the server either using the OMWS 1.0 or 2.0 APIs. The OMWS responds to client requests by creating unique tickets (that maintains information regarding the request) when soliciting the execution of OM jobs/experiments or can provide status information about the state of the service, job or available OM algorithms and layers.

Although OMWS 1.0 originally came with a *cron* script to support integration with HTCondor, it appeared not to have been completely untested. Thus, the opportunity was taken to redesign and evaluate alternative implementations. Initially, a new HTCondor Adaptor was developed, as a daemon service, to check the existence of new tickets and translate them into HTCondor job description files that can be submitted to the central manager. The HTCondor scheduler then submits each requested individual job to an idle worker. With the advent of OMWS 2.0, two solutions are available: one that uses this adaptor that is now compliant with the new standard and latest version of openModeller, or; a new HTCondor Connector that submits complex experiments to HTCondor's DAGMan to manage dependencies between individual OM operations.

The HTCondor requires a submission file in which the execution environment of the pool is defined. Currently, the HTCondor Adapter and Connector only defines job in the "Vanilla Universe", which does not support "checkpointing" capabilities and other interesting features. The feasibility of using other HTCondor universes was not considered in during this project (but could be evaluated in the future, especially if the opportunistic utilization of resources were to be considered), since, for example, the "Standard Universe" that allows checkpointing requires applications to linked with HTCondor specific libraries which would involve a significant amount of development time. Furthermore, fault tolerance is being provided by a new Local Site Orchestrator component that

manages multiple back-end resources at the same site. Note however, in order to deploy the new parallelised version of the OM projection stage, the HTCondor "Parallel Universe" is required for the execution of MPI applications.

The Local Site Orchestrator (LSO) is a new component that has been developed to support functionalities not provided by the OMWS 2.0. Its principal objective is to distribute experiments (or sub sets of an experiment) across multiple compute back-ends available behind a single oM Server (see Figure 19). In this project, the LSO is being used to interface with both an HTCondor pool and HPC clusters. Additional, the LSO is responsible for coordinating requests for remote environmental layers and publishing monitoring and configuration data in the Information Service.

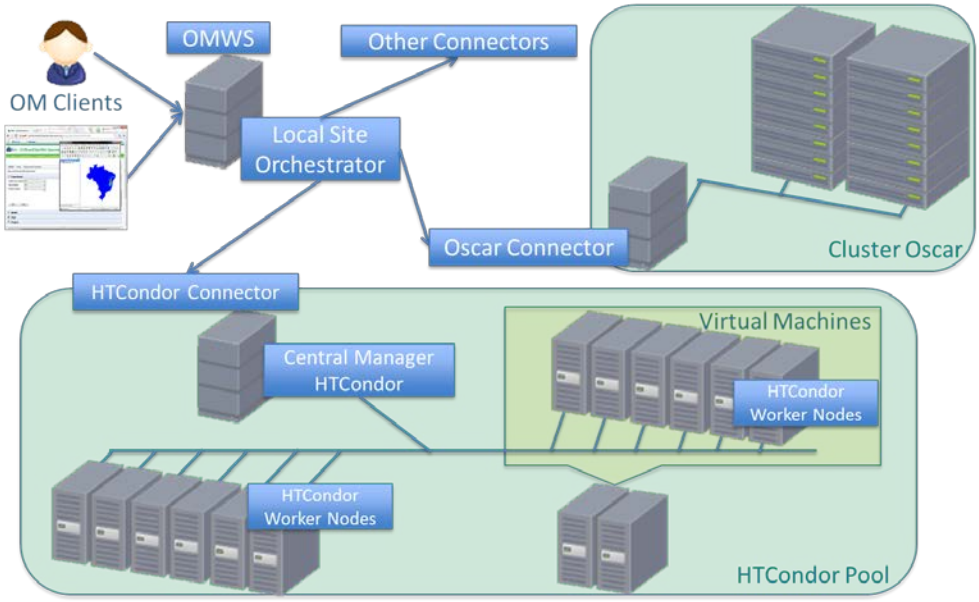

Figure 19 - OMWS 1.0 and 2.0 services with multiple shared back-ends

3.2.3.1 An Initial Evaluation

Three experiments are described here with the purpose of identifying the advantages and pitfalls of the underlying infrastructure configuration. The first experiment studies the effect on the execution times of OM jobs when instantiating multiple virtual machines on the same server while they execute distinct instances of openModeller. The scenario was set on an HT2 machine with environmental layers being stored on the local disk and each VM executing the *om_console* command with the ANN algorithm, i.e. all the three phases were carried out (create model, test model and project model). This algorithm was chosen since it is one of the most compute intensive [7].

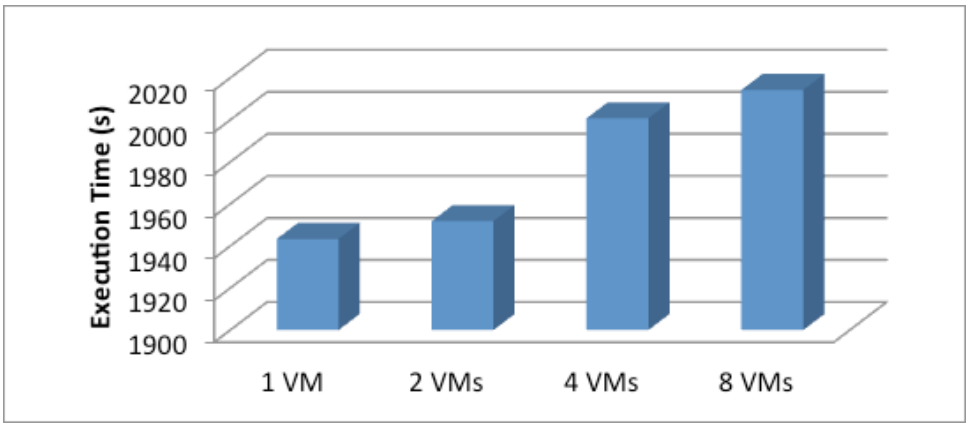

Figure 20 - Average execution times for an increasing number of concurrent VMs

From Figure 20, one can see the performance loss (from 0.4% to 3.6%) with the increasing number of VMs. The results are consistent with concurrent execution of openModeller directly on a HT2 machine, where 12 instances on the 12 cores incurred an 11% overhead. This degradation is more due to the multiple accesses to data stored locally by the various VMs than to the overhead related to the virtualized environment or the configuration of the physical machine.

In order to understand the particularities of a hybrid environment of virtual machines working together with physical ones to execute openModeller, the following experiment considers the impact of the location of data (environmental layers). Note that only one scenario was managed by HTCondor. The goal here is to compare the impact of executing openModeller with data either on the local disk or a remote server accessed by NFS, on real and virtualised machines. In this set of experiments, only one instance of openModeller was executed using ANN as the modelling algorithm via *om_console*.

Figure 21 reveals a small overhead (1.3%) for the execution of the experiment in a virtual machine rather than directly on the physical hardware (based on comparing the running time of 1918.37 seconds of a single instance on a physical machine with that of 1943.37 seconds of the same instance in a virtual machine). The overhead to submit the experiment through HTCondor is 39.73 seconds, which is 2%, for this specific case. The overhead for a VM to access environmental data via NFS rather than from the local disk is almost unnoticeable in this experiment.

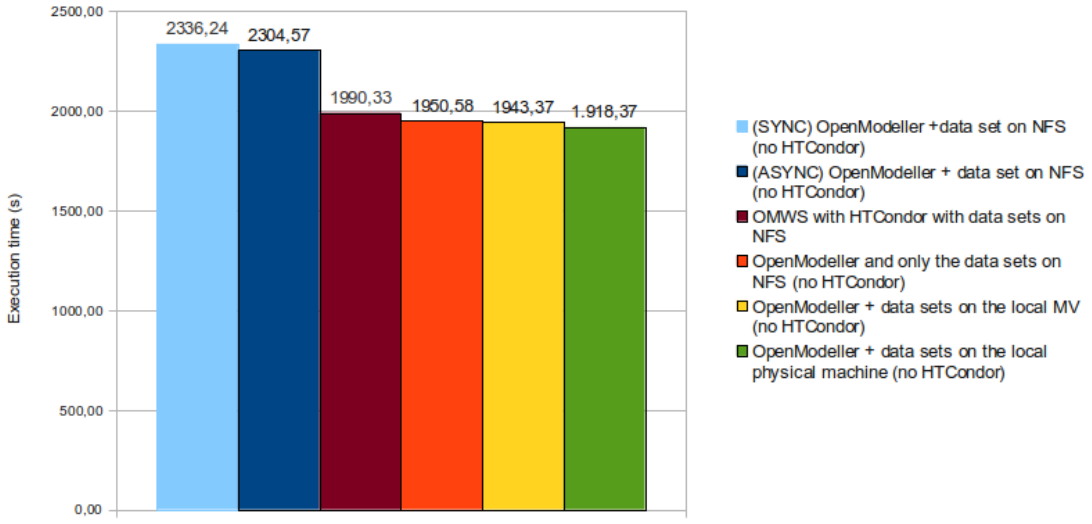

Figure 21 - Average execution time of one instance of OpenModeller with the modelling algorithm ANN

Finally, we evaluated the possibility of keeping the openModeller executable object code in a single location, in order to facilitate future updates. Unfortunately, this incurs a 19.7% overhead when using NFS with the SYNC parameter (NFS servers normally run in SYNC mode, which means a block is actually written to disk before informing the client that it has been written and allowing it to proceed). Using the ASYNC option can bring performance benefits as a client does need to wait for the write to take place. In this experiment, the improvement is slight; the overhead falls to 18.1% and thus remains significant.

These results indicate that reads from data storage have an impact on performance and data locality is important. Although both the openModeller tool and the environmental layers are thus currently stored locally at each resource provider within the EUBrazilOpenBio infrastructure, researchers will be permitted to use environmental layers of their own choice that are not available locally. These layers need to be received and somehow made available to the computation resources that execute the openModeller operations.

A test suite has been created as part of the EUBrazilOpenBio project to test the performance of an openModeller Web Service (OMWS) instance, based on the expected use by the BVH. This application is based on OMWS API v1.0 and uses occurrence points for a single species (*Passiflora luetzelburgii*) to generate models with 5 different algorithms (Maxent, ENFA, one-class SVM, Mahalanobis Distance and GARP BS). Models are generated for each algorithm to perform a 10-fold cross validation with different combinations of occurrence points, and then a final model is generated with all points and projected on a map of Brazil. In this test suite, all model creations are independent, including the final one. As seen before, both the model tests and projections depend on the creation of the corresponding distribution.

The results of the execution of three scenarios under the management of HTCondor, are shown in Figure 22, where the test suite was executed on:

1. only physical machines (PMs) of Host Type 1, all of them with two slots apart from one experiment where one processor was used “1 PM (one slot)”;
2. only virtual machines (VMs) running on HT2 machines;
3. a combination of VMs and PMs.

The aim of this evaluation is to begin to identify bottlenecks when scaling up the heterogeneous execution environment and identify where improvements will be required.

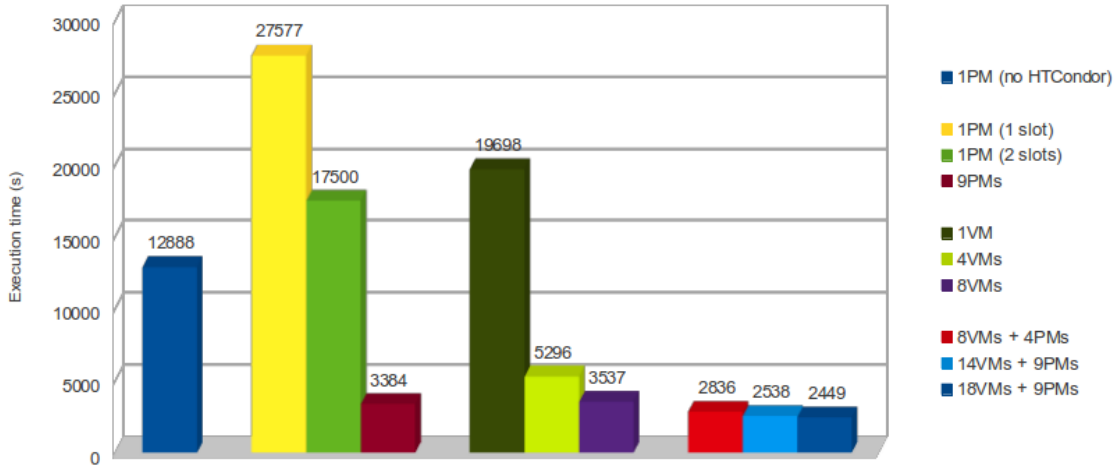

Figure 22 - Average execution time of one instance of openModeller with ANN as the modelling algorithm

One can observe that executing with 2 slots instead of one on a physical machine does not take exactly half the time on one machine and a similar situation occurs when using VMs. This indicates that there is some inefficiency in the system and this is confirmed by comparing times of 1PM (2 slots) with 1PM (no HTCondor). The latter is an OMWS running on a HT1 machine without HTCondor. The 36% overhead is a reflection of the costs of HTCondor´s management and allocation strategy. Although reasonable gains can be observed when incorporating more resources, the identified inefficiencies need to be addressed for scalability. Given the existence of task dependencies and tasks with large differences in execution, a task scheduler will be beneficial.

Based on the performance analysis carried out on openModeller, simple execution models have been incorporated in to the HTCondor Adaptor/Connector and the EasyGrid AMS workflow manager. An initial evaluation of a new version of the HTCondor Adaptor led to a 20% improvement over the results presented in Figure 20. The HTCondor Connector uses the meta-scheduler, DAGMan, to manage the execution of workflows in the HTCondor pool in order to support the new extensions in OMWS 2.0. This will allow the service to receive multi-stage/multi-parameter experiments (such as the test suite) as a single request, in a fashion similar to the COMPSs workflow. The HTCondor Connector can modify experiment requests by generating the job submission files either for DAGMan, containing the description of tasks and their dependencies, or directly for HTCondor as single openModeller operations. DAGMan is then responsible for submitting ready operations with their respective job submission requests to HTCondor.

3.2.4 openModeller Enhancements with EasyGrid AMS

There are two ways to improve the runtime of an ENM experiment with openModeller. One is to execute the stages of an experiment (create model, test the model and project the model) in parallel respecting the dependencies among them, as was investigated through the enhancements with COMPSs and HTCondor. In addition, further improvements are achievable by improving the runtime of individual stages through parallelization. In either of these complementary approaches, EasyGrid AMS can be employed to manage the parallel execution, i.e. both to execute a single parallelised stage, or a complete ENM experiment. This latter possibility might be beneficial for resources that cannot adopt COMPSs or HTCondor.

The results of openModeller profiling analysis identified the model projection stage, in general, to be the most time consuming (except for two algorithms: GARP_BS and ANN) and for this reason, *om_project* was selected to be parallelised. The projection stage uses a model (generated by *om_model* with a chosen algorithm) to project the suitability of a given environmental scenario for a given species. The result is an image representing the potential distribution of the selected species within a chosen region. In principle, the parallelization of *om_project* is simple; the points within the chosen region of interest can be partitioned into smaller subsets and distributed among the available resources for evaluation by the model.

3.2.4.1 Implementation of a new parallel version of om_project

An existing parallel implementation, available within openModeller, partitions the map into blocks of a fixed size and distributes each block to worker processes, on demand. Additionally two extra processes are used: one to divide and distribute the blocks and the other to hold the blocks of the map that have already been projected. An analysis revealed an implementation detail that is important to the performance of *om_project*, the obligation to output (write) the points of the projection map in sequence within the image file, even though the calculation of the projection does not need to be done sequentially. Given that the capacity of this second process is limited, this can lead to a number of periods of synchronisation, which causes delays during execution since dependencies are introduced between blocks.

A new parallel implementation solves this problem by modifying writing procedure for the image file. Now only one extra process is used to divide and distribute tasks among worker processes and hold the subsets of the map already projected. Based on preliminary evaluations of the two parallel implementations, Figures 23 and 24 highlight the reduction in the execution times for the projection stage for a few of the algorithms available in openModeller, as the number of processors is increased from 1 to 24 (for this comparison, the degree of parallelism is kept the same between implementations, thus the new algorithm requires one processor less).

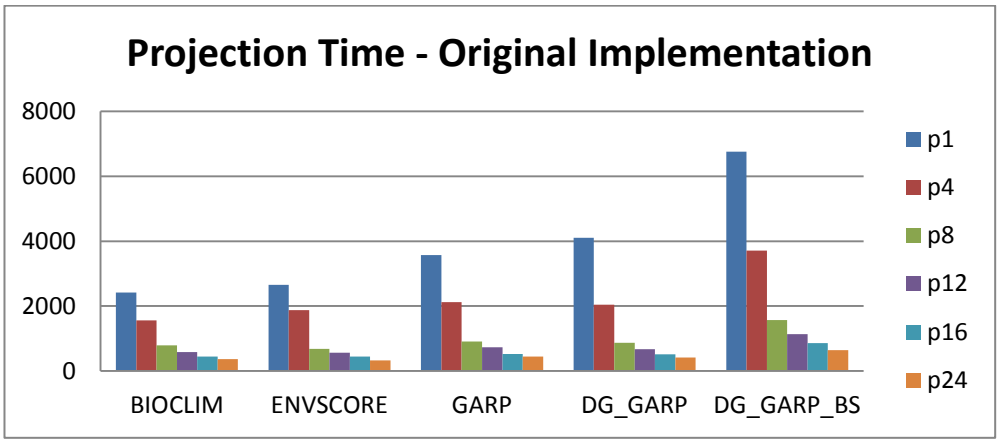

Figure 23 – Execution times (in seconds) obtained by the existing parallel version of *om_project*

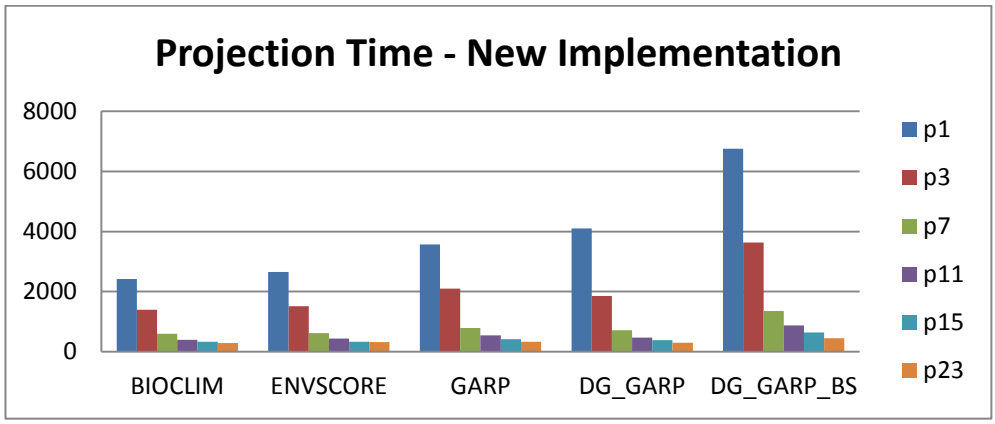

Figure 24 – Execution times (in seconds) obtained by the new parallel version of *om_project*

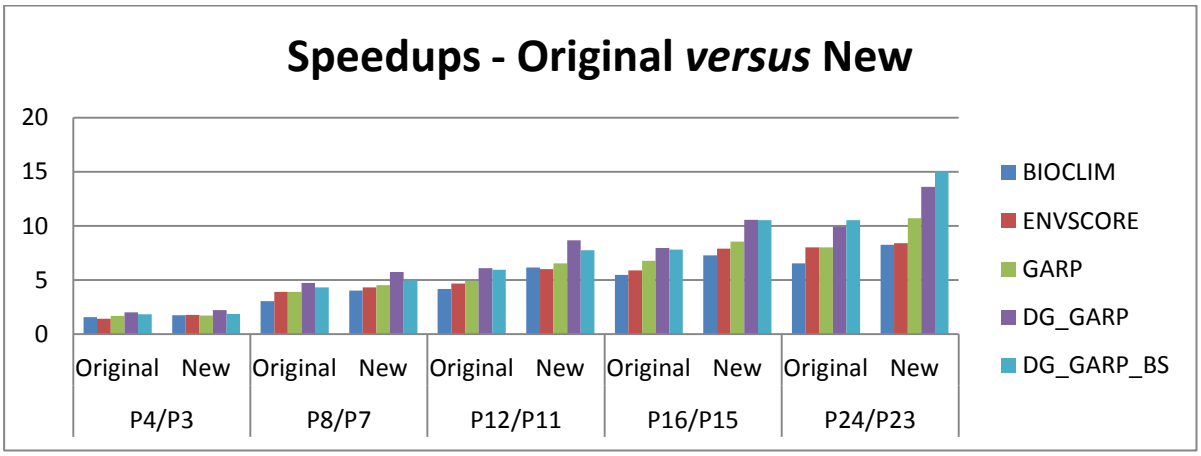

Figure 25 – A comparison of speedups achieved by the parallel versions of *om_project*

In Figure 25, there are two sets of bars, for a given number of worker processes, that show the speed-ups obtained by the original and new parallel implementations, respectively. Results show that the new implementation is better than the original, although the speed-ups are still smaller than expected, i.e., were not close to linear. For these algorithms, the new parallel implementation achieves a speedup of between 8.2 and 15.1 while the original ranged from 6.5 to 10.5. Speed-ups tend to improve as the execution time of the algorithms increase. However, these results still leave room for improvement and further evaluations and a redesign for integration with the EasyGrid AMS was carried out.

3.2.4.2 EasyGrid AMS with an alternative parallel MPI om_project

The EasyGrid execution model suggests implementing parallel tasks with fine granularity. Fine granularity provides a simpler procedure for migrating tasks when they need to be rescheduled and a method of restarting processes in the case of failures, unlike coarse granularity tasks which require the implementation of complex and costly methods of process migration and checkpoint.

The new parallel implementation of *om_project* is characterised as a bag-of-tasks application whose task granularity can be modified, accordingly, during execution. The idea is to define the ideal block size automatically in order to balance data processing and I/O given a possible heterogeneous execution environment. Thus, with the scheduling, fault tolerance and malleability features of the

EasyGrid AMS, the new *om_project* should be able to adapt better to systems, dynamically taking advantage of available or surplus processing capacity.

EasyGrid AMS version for the *om_project* tool uses a novel parallelisation strategy different from that adopted by the two previously presented MPI implementations. The AMS approach involves the allocation of significantly larger number of tasks, of finer granularity, than the number of CPUs. But in this case, not all tasks will run simultaneously at the same time on the available or allocated CPUs. While, the execution appears to be similar to the original and new MPI implementations in relation to the number of processors utilised, note that the workload of tasks can vary even though block sizes are of the same size. The AMS version allows the block size to vary and relies on the internal dynamic scheduler to balance the application workload.

In addition to exploiting parallelism through MPI, the complementary use of threads and GPGPU accelerators by MPI tasks was also investigated. However preliminary results identified significant bottlenecks and thus the need for a more in depth analysis has been left for future work.

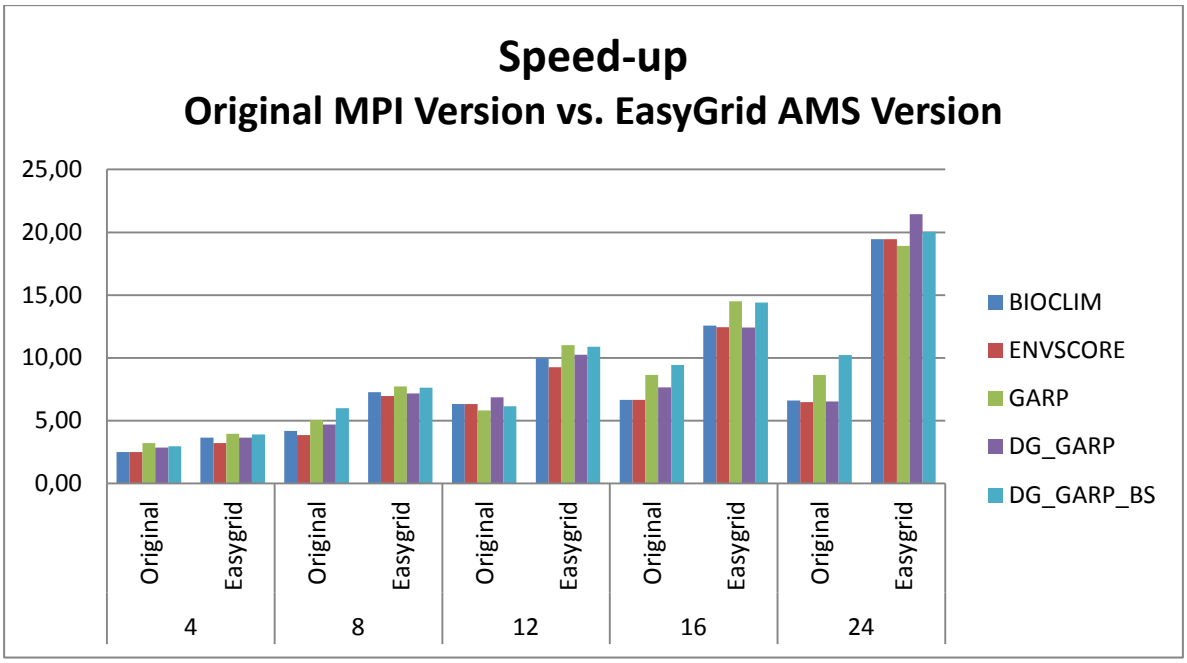

Figure 26 – A comparison of speedups for different modelling algorithms with parallel versions of *om_project*

The improvement obtained by the EasyGrid AMS version over the original MPI implementation can be seen in Figure 26. Independently of the modelling algorithms, the performance has been improved. In particular, the original version is being to show signs of limited scalability with 24 CPUs. The efficiency in terms of resource utilisation is presented in Figure 27, where it is clear that the EasyGrid AMS version has the advantage because of its ability to reach more acceptable level.

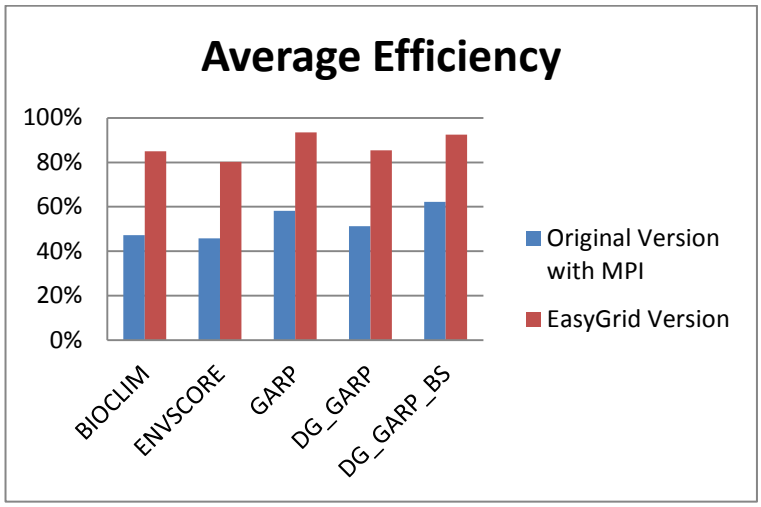

Figure 27 – A comparison of speedup efficiency for up to 24 CPUs

3.3 EUBrazilOpenBio and federated infrastructures

The project has been working on how to support and extend the access to a larger number of biodiversity scientists. In particular the focus has been on the development of extensions to the EUBrazilOpenBio infrastructure to allow the access to federated cloud infrastructures. The main requirement to achieve this goal is interoperability at various levels as resources provision, authentication and authorization mechanisms, and deployment of the applications across multiple providers. The project followed the approach of the EGI Cloud Infrastructure Platform whose federation model provides an abstract cloud management stack to operate an infrastructure integrated with the components of EGI Core Infrastructure Platform. This model defines the deployment of interoperable standard interfaces for VM management (OCCI), data management (CDMI) and information discovery (GLUE 2). The interaction with the core EGI components includes the integration with the AAI VOMS system and with the monitoring and accounting services. An Appliance Repository and a VM Marketplace support the sharing and deployment of appliances across the providers.

The project's infrastructure supports the EGI Federated Cloud through the COMPSs-PMES components. A new connector has been developed in the COMPSs runtime to operate with the interfaces and protocols previously described. The VM management is implemented using the rOCCI client[11] that transparently interoperates with the rOCCI servers deployed in the testbed on top of different middlewares as OpenNebula and OpenStack. The connector supports different authentication methods, including X509 type through VOMS proxy certificates. The connector is able to match the requirements (in terms of CPU, memory, disk, etc.) of each task composing the application with the resource templates available on each provider.

[11] The rocci framework. http://dev.opennebula.org/projects/ogf-occi/wiki

4 Conclusions

This report provides the final picture of the EUBrazilOpenBio infrastructure deployment and of the use cases implementations and present the refinements and modifications that have been made to reflect evolved requirements based on feedback received by the users during the evaluation phase.

The developments described here are based (i) on an analysis described in the previous report [4], (ii) on the scenario requirements [6], and (iii) on a profiling study aimed at identifying the possible enhancements of the use cases and the selection of technologies that implements the EUBrazilOpenBio production e-infrastructure. These components include data management, programming models, execution managers and a set of tools offered by gCube.

Cloud platforms and cluster resources have been used as computational back-ends in order to evaluate the best option for the scenarios. The developed components have been also evaluated on computational backends external to the project and offered to external communities.

5 Acronyms and Abbreviations

Acronym	Definition
AMS	Application Management System
ANN	Artificial Neural Networks
API	Application Programming Interface
AWS	Amazon Web Services
BES	Basic Execution Service
BVH	Brazilian Virtual Herbarium of Flora and Fungi
CNPq	National Council for Scientific and Technological Development of Brazil
CoL	Catalogue of Life
COMPSs	COMP Superscalar
DAGMan	Directed Acyclic Graph Manager
DAO	Data Access Object
EC	European Commission
ENFA	Ecological Niche Factor Analysis
ENM	Ecological Niche Modelling
EOS	Experiment Orchestrator Service
GB	Governance Board
GBIF	Global Biodiversity Information Facility
gHNs	gCube Hosting Nodes
GUI	Graphical User Interface
JRO	Job Resources Optimizer
JSDL	Job Submission Description Language
MPI	Message Passing Interface
NFS	Network File System
OCCI	Open Cloud Computing Interface
OGF	Open Grid Forum
OMWS	openModeller Web Service
PMES	Programming Model Enactment Service
REST	Representational state transfer
ROC	Receiver Operating Characteristic
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVM	Support Vector Machines
ToC	Table of Contents
UUID	Universally Unique IDentifier

VENUS-C	Virtual multidisciplinary Environments Using Cloud infrastructure project
VM	Virtual Machine
VOMS	Virtual Organization Membership Service
VRE	Virtual Research Environment
WP	Work Package
XML	Extensible Markup Language

References

[1]. C. Boeres, V. Rebello, L. Candela. Infrastructure Set-up, Planning, and Coordination. D4.1 EUBrazilOpenBio Project Deliverable. January 2012.
[2]. L. Candela, V. Rebello. Development Planning and Coordination. D3.1 EUBrazilOpenBio Project Deliverable. January 2012.
[3]. L. Candela, V. Rebello. EUBrazilOpenBio Software Platform Core. D3.2 EUBrazilOpenBio Project Deliverable. April 2012.
[4]. A. Nascimento, A. Fonseca, D. Lezzi, E. Torres, I. Blanquer, R. Amaral, R. De Giovanni, V. Garcia, V. Rebello. Infrastructure Enhancement and Optimisation Report. D4.2 EUBrazilOpenBio Project Deliverable. June 2012.
[5]. V. Rebello, R. De Giovanni, L. Candela. Niche Modelling Services. D3.3 EUBrazilOpenBio Project Deliverable. April 2012.
[6]. E. Torres, I. Blanquer, L. Candela, R. De Giovanni. Use Case Requirements Specifications. D2.1 EUBrazilOpenBio Project Deliverable. January 2012.
[7]. Performance Profiling Report for the Ecological Niche Modelling Tool openModeller, Version 1.0, 31/1/2013. Available at http://wiki.eubrazilopenbio.eu/index.php/Performance_Profiling_of_openModeller
[8]. Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing-degrees, models, and applications. ACM Computing Surveys, 40(3):7:1–7:28, August 2008.
[9]. D. Lezzi, E. Torres, F. Quevedo, I. Blanquer, R. De Giovanni, J.Fernando, V. Garcia, R. Assad, C. Boeres, F. Oliveira, R Amaral, V. Rebello, L. Candela. D4.3 Infrastructure Enhancement and Optimisation Final Report. February 2013.