Legion: An extensible lightweight web framework for easy BOINC task submission, monitoring and result retrieval using web services

[a,b,c]Genghis Ríos, [a,b]Pablo Fonseca, [a]Oscar Díaz
[a,b,c]Pontificia Universidad Católica del Perú, Dirección de Informática Académica, Perú
grios@pucp.edu.pe
[a,b]Pontificia Universidad Católica del Perú, Dirección de Informática Académica
pfonseca@pucp.edu.pe
[a]Pontificia Universidad Católica del Perú, Dirección de Informática Académica
diaz.oa@pucp.edu.pe

Abstract. Nowadays, researchers both from industry and academia need to perform computationally intensive calculations as part of their activities. If an institution decides to deploy Berkeley Open Infrastructure for Network Computing (BOINC) Desktop Grid in order to support these needs, the inherent complexity of task submission might represent a barrier to end users. To address this we present Legion, a lightweight framework for generating web interfaces for BOINC that successfully reduces the administration time and hides the complexity to end users. We also present how Legion Framework can be adapted to work with other Grid Management Systems.

Keywords: Legion, BOINC, Framework, Grid Computing, High Throughput Computing, PUCP.

1. Introduction

Today, research centers around the world require running intensive calculations for solving complex problems in many areas such as climate prediction, high energy physics, data mining, protein folding, validation of statistical models, etc. More often the computing power requirements to perform these tasks are beyond the capabilities of a personal computer. In order to bypass this restriction, it is mandatory to count upon a large-scale infrastructure. Berkeley Open Infrastructure for Network Computing (BOINC) Desktop Grid should be considered as a feasible alternative "supercomputer", especially in a constrained-resource scenario or where the required computational power exceeds the limit of the gamma of affordable supercomputers.

Grid management systems such as BOINC [1] are geared to effectively handle task queuing, task distribution in nodes, execution and retrieval of task results; on the other side they are not easy to use. The latter particularly represents a barrier and limits a wider adoption because researchers spend time on the particularities of the grid system instead of working on their core research. The difficulty mainly lies in the complex procedures required to send calculations and retrieve results. Therefore, a framework that successfully hides the complexity of the grid,

exposes utilities for managing tasks and allows to be accessed from any location is highly desirable.

At Pontificia Universidad Católica del Perú (PUCP), BOINC is being used since 2008 to support intensive computing requirements of in-house researchers. This is done by taking advantage of unused computing power of the computer commons without disturbing their users. This scheme enables our institution to require less power than with dedicated hardware and thus being a greener approach.

Through this paper we present Legion, a framework that reduces the adoption barrier by generating web interfaces that enable web access to BOINC. Legion Framework is modular and can also be adapted to other Grid Management Systems that work under the master-worker paradigm.

The rest of the paper is organized as follows: Section 2 presents a comparison with similar work, Section 3 explains how the execution in the grid is supported under the master-worker paradigm; Sections 4 and 5 show the overall architecture of the framework by analyzing the two main components Legion Web Interface and Legion Web Services. Section 6 shows how the integration with other Grid Management Systems is managed. Finally, Section 7 addresses the future work.

2. Related Work

The first version of the Legion System [2] is the direct predecessor of the current framework, it tried to abstract the complexity of the Grid by using the same approach of providing customized web interfaces for task submission, monitoring and result retrieval. However, the work required to add a new interface was complex. In addition, the scheme was not extensible and thus could only work with BOINC.

Another project with similar goals as the one presented in this paper is RBoinc [3], a scientific interface that extends the functionality of BOINC and allows remotely sending tasks and monitoring their progress. This project uses the WebDAV protocol for transferring binary files. The complete system consists of two components: a command-line interface and a server (both developed in Perl).

Virtual Community Grid [4] is an initiative to provide web access to the resources of a Grid based on Globus Toolkit. The project aims to provide access to distributed computing resources for the national research system in Brazil.

Another project which might help in order to build a Grid Infrastructure with BOINC is Jarifa [5]. It allows the grid-admin to centralize client (execution nodes) management and get statistics of use. However, these features don't collide with Legion Framework's ones, and thus both should be considered as complementary tools.

3. Execution on the Grid based on tasks and projects

In order to benefit from Legion, a problem must allow to be solved under the master-worker paradigm. This means that a task should be able to be divided in parts that are independent. Therefore, it would be possible to abstract the concept of task as a collection of independent and smaller computing units.

The tasks go through four stages: 1) creation of computational units, 2) queuing in the Grid Management System, 3) execution at the grid nodes and 4) packaging of the partial results after

the task queue finishes its processing. This way it is possible to obtain satisfactory results for very long tasks by dividing them into many computing units.

The concept of a project in Legion Framework encompasses both: a group of users and the application that they run on the Grid. This application must be batch-kind and capable of performing the calculation of any computing unit, so the only variation between units should consist of parameters and input files.

Every user of a Legion project is assigned a role which could be either creator or subscriber. The first will be allowed to create tasks, delete them, download their results and subscribe other users to the created tasks, while the second will just be allowed to download the results of the tasks.

The framework also supports the management of credits to quantify the use of computational resources proportional to the execution time of submitted tasks. These credits are granted by the institution and represent a limited resource as task submission is subject to the availability of the latter. However, this functionality can be disabled in environments where it is not important to be ruled by a credit economy.

4. Architecture

The main goal while designing Legion was to specify a set of maintainable components that interact with each other and that could be replaced by convenience. For this reason, we chose to use a service-oriented architecture where components are subscribed to a "contract". This means that a single specification of the methods is given, which would make possible the development of new components that can fit without problems.

Legion has two main components as can be seen in Figure 1. Legion Web Services (LWS) is the layer that is inserted on top of the Grid Management System (in this case BOINC) and that can manage it completely, and Legion Web Interface (LWI), a web application that can generate and host the web interfaces that provide remote access to the grid resources.

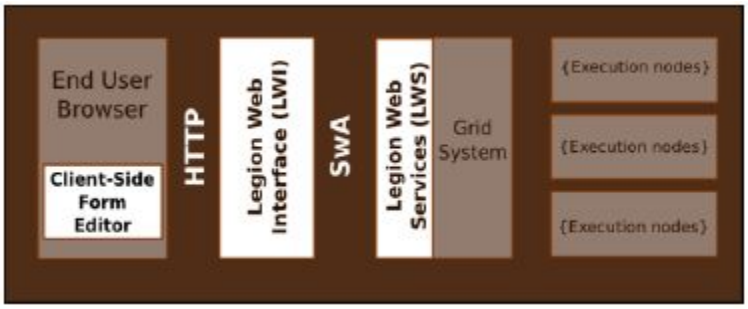

Figure 1. Architecture of Legion Framework

4.1. Legion Web Services (LWS)

LWS is an abstract layer because it represents a specification and not a particular implementation. A list of methods that must be exposed by a web service is given. With the implementation of these methods on top of a Grid Management System, the consumer application should be able to completely manage the grid. In a following section the customized implementation for BOINC is presented.

Sometimes the concept of task is not implemented in a Grid Management Systems, so it is up to LWS to implement it. The methods LWS must expose are:

Task_Inf: Allows retrieving the information of a task.
Task_Progress: Allows retrieving the progress of a task.
Task_Cancel: Allows canceling a task.
Task_Result: Allows retrieving the result of a task.
Task_List: Allows listing running tasks.
Task_Create: Allows creating a task. A configuration file (config.xml) is a required parameter of this method. This file should indicate the input filenames, result filenames, the command line to run the executable files and specification of loops for the generation of computing units. An example of this file is shown on Figure 4.
User_added: Allows adding a user.
User_Validate: Allows validating user's credentials.
The implementation must support the SOAP with Attachments (SwA) standard [6], since the transfer of binary files is mandatory.

```
<form>
  <section name="Integral parameters">
    <textbox>
      <name>txtFx</name>
      <description>f(x)</description>
      <info>Function to integrate</info>
    </textbox>
    <textbox>
      <name>txtStart</name>
      <description>Lower Limit</description>
      <info>Must be an integer</info>
    </textbox>
    <textbox>
      <name>txtStop</name>
      <description>Upper Limit</description>
      <info>Must be an integer</info>
    </textbox>
    <textbox>
      <name>txtStep</name>
      <description>Step</description>
      <info>Must be an integer</info>
    </textbox>
  </section>
  <section name="Precision">
    <textbox>
      <name>txtPrecision</name>
      <description>Precision</description>
      <info>Choose a value like 0.001</info>
      <default>0.001</default>
    </textbox>
  </section>
</form>
```

Figure 2. Example form XML representation entered by administrator

4.2. Legion Web Interface (LWI)

This layer was developed to ease the access to the grid for researchers through the generation of web interfaces. These interfaces include: 1) a custom task submission form per project, 2) a monitoring page that shows the list of submitted tasks with their progress, and 3) tools to share the tasks between users of the same group.

The interfaces are generated within the same application through a series of guided steps commonly known as "wizard". This task, done by the administrator, is easier than developing a

custom interface for each project. The steps include a graphical form editor for the custom task submission page as an alternative to entering the XML form specification directly.

When the end user opens the task submission page for a project, the xml representation of the form, either generated by the graphical form editor or entered directly, is rendered to display the HTML components which will let the user to specify the parameters and input files for a custom task. Figure 2 shows an example for the XML representation of the rendered form for task submission shown in Figure 3.

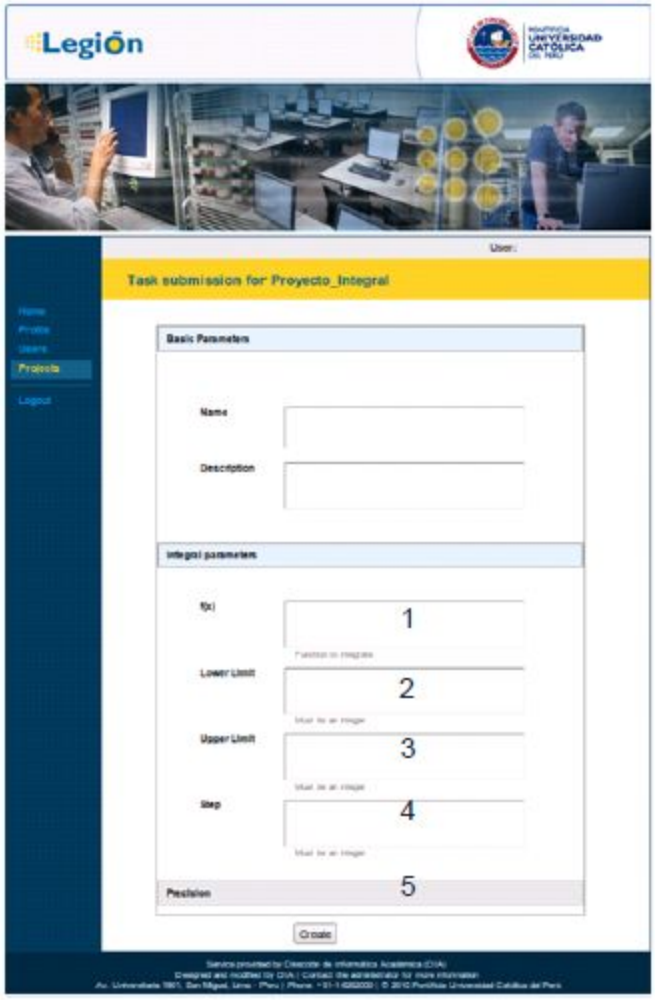

Figure 3. Example of rendered task submission form for researchers in LWI.

Another important step required by the wizard is entering a "config.xml" template; an example is shown in Figure 4. This file is required by LWS in order to break down the task in many smaller computing units and must be conformant to "config.xml" specification when interpreted. This template is entered directly by the system administrator at project creation stage and interpreted after each task submission. The interpreter supports the replacement of

Form and Project variables and the interpretation of simple mathematical expressions with these variables.

The example shown on Figures 2, 3, 4 and 6 belong to a trivial distributed numerical integration project. Lower and upper bounds are specified with the function to integrate. The interval is broken down into smaller intervals with a width equal to the step parameter. The computing units are created by LWS as specified by the config.xml which is interpreted after task submission. The total result of the integration process is given by the sum of partial results.

```
<task>
        <name>{{Form:txtName}}</name>
        <desc>{{Form:txtDescription}}</desc>
        <exec_file>
         <name>integrador</name>
         <cmd>"{{Form:txtFx}}" {x} {{Form:txtStep}} {{Form:txtPrecision}}</cmd>
         <compress>{{Project:compress}}</compress>
         <process>
                <tag>x</tag>
                <start>{{Form:txtStart}}</start>
                <stop>{{Form:txtStop}}</stop>
                <step>{{Form:txtStep}}</step>
        </process>
        </exec_file>
        <out_file>
                <type>cat</type>
                <name>result.txt</name>
        </out_file>
</task>
```

Figure 4. Example config.xml template.

LWI was developed using the Java programming language. The reason for choosing the latter has to do with the robustness of the platform. The inner structure is based on the MVC pattern (as shown in the Figure 5) in order to produce a maintainable product. The business logic is encapsulated in the "Services" component. The "DAO" component and "Web Service Consumer" provide the data access layer to LWI. The MySQL database is accessed with the help of the MyBatis ORM framework. Apache Axis 2 [7], a widely adopted Web Service library for Java is used to interact with the LWS layer via the SwA standard.

The client-side Form Editor was developed using the Google Web Toolkit (used for compiling a subset of Java into Javascript code) because of the useful widgets available and the ease provided by the object oriented programming.

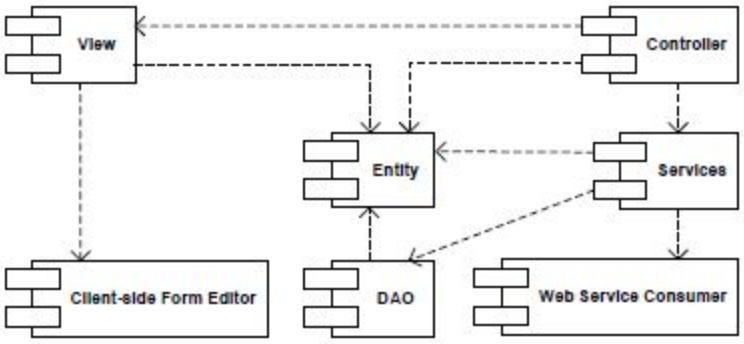

Figure 5. Inner components of LWI.

Figure 6. Researcher's task monitoring page

5. Legion Web Services for BOINC (LWSB)

5.1. BOINC

BOINC is a platform for distributed computing using public resources [8]. It consists of a central server that manages the distribution of tasks between the clients on multiple platforms such as Windows, Linux, Mac OS X, etc. It allows operating geographically distributed infrastructures that can achieve a large scale computing power based on the paradigm of computational resources volunteering.

Projects like SETI@home, Predictor@home and Folding@home benefit from BOINC because a "supercomputer" is not enough to handle their computational requirements. Other applications for BOINC include intra-organizational Grids for universities and companies such as supercomputing virtual campus [9].

The workflow in BOINC is as follows: First, the creation of work units can be done either by a custom application with the help of C++ libraries or directly by the default "create_work" application. After creation, the scheduler will assign work units to BOINC clients for execution. After a work unit is finished, the results are sent back to the server and go through a validation stage if needed (in the case that replicas are being used). Then, the assimilation of work units is performed for general-purpose results post-processing e.g. if results need to be concatenated with other files. The stage of validation and assimilation [10] is done through two programs that must be implemented using either the libraries available in C and Python, or via the "Generic Assimilator Framework" given by BOINC.

Additionally, it is possible to monitor the progress of work units by using the BOINC project web interface for "Project Management" or the C/Python libraries that are provided or by making direct queries to the MySQL BOINC project database.

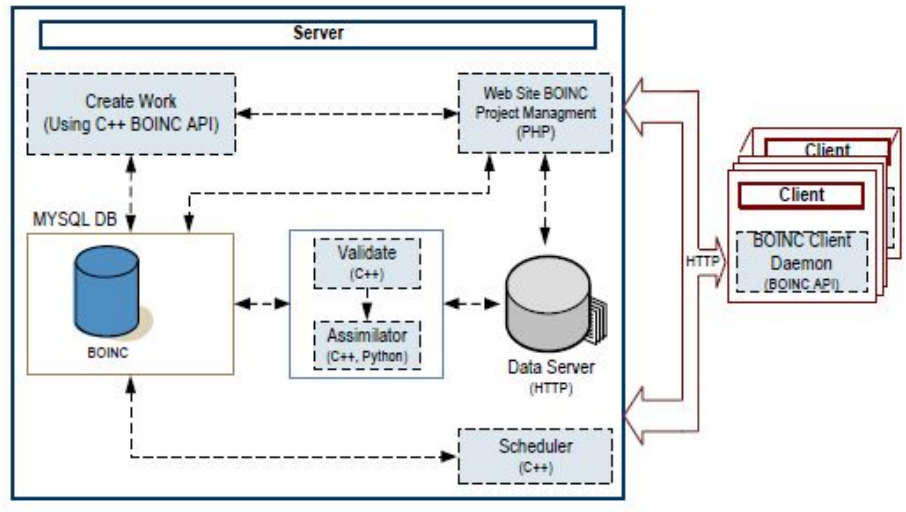

Figure 7. BOINC Architecture

5.2. Development of LWSB

In order to allow Legion Framework to work with BOINC, a custom LWS component was developed. The main purpose of LSWB is to provide access to the resources of a BOINC Desktop Grid via SOAP and not only for LWI consumption, but it is intended to provide a general-purpose web services interface for BOINC.

BOINC doesn't support the concept of task as we see in Legion, so it was necessary to implement it in LWSB. We chose Python as the language to implement the LWSB component because there are already several tools for BOINC written in that language.

The developed component allows the BOINC server to interact with LWI for adding and authenticating users, to create tasks made up of smaller work units following the config.xml specification, to post-process a task, to retrieve the status of tasks, to retrieve the result of tasks when finished and canceling a task.

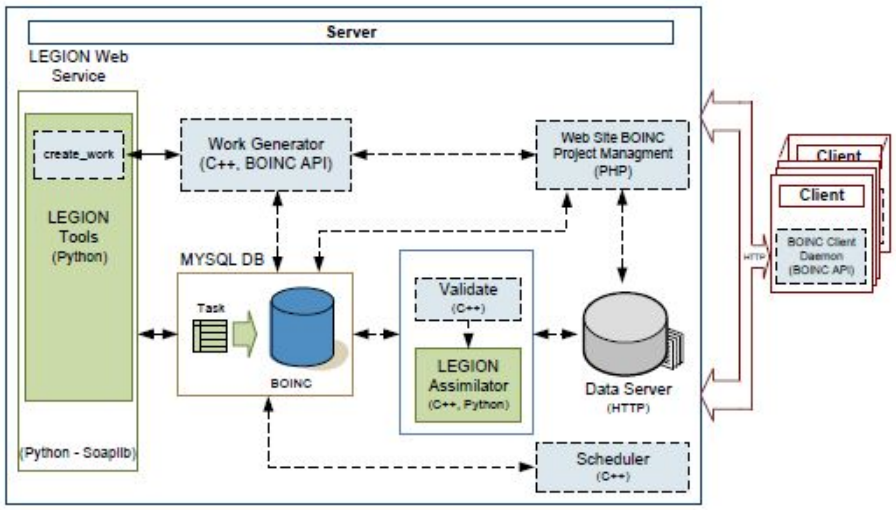

Figure 8. BOINC plus LWSB

In order to work with tasks, it was necessary to save some information about them. With this in mind, a table named 'Task' was created inside the BOINC Project database scheme. This table stores the information associated with the task: name, how many work units it has, the current state which could be: creating, executing, canceled or finished; also, there are fields for storing execution time, CPU time and consumed FLOPS which are obtained from the work units as they are being completed. The way for relating work units inside a task is by using a field named hash whose value is shared between work units in the 'batch' field of the table 'workunit'.

Furthermore, a set of libraries were developed for reading the config.xml file (Figure 4), distributing the input files, and for creating the files required by BOINC: job.xml, work unit templates, result templates. Also a custom Assimilator was developed in order to update the fields in the table Task as work units were finished and in order to execute additional operations such as result-files concatenation and/or compression. After the execution of the task is finished, the Assimilator copies the result-files to a preconfigured folder and sends an email alerting the operator. In case an error happens during execution of a work unit belonging to a task, the Assimilator automatically cancels the task and alerts the operator with an email.

5.3. Centralized BOINC Servers Management

Legion Framework is also suitable for centralized BOINC Server management and thus enabling multiuser remote task submission and monitoring for BOINC Projects. This might be useful when several Intra-Organizational BOINC Desktop Grids are deployed and intended to be managed within a centralized context. Given that the Grid resources are accessed via the

LWS tier it is feasible to enable several back-ends (BOINC Server plus LWSB Component) which might be distributed within a country or bigger regions provided that adequate network infrastructure exists. In order to dynamically manage execution nodes a companion account manager system such as Jarifa [5] might be required. A proposed architecture is seen in Figure 9.

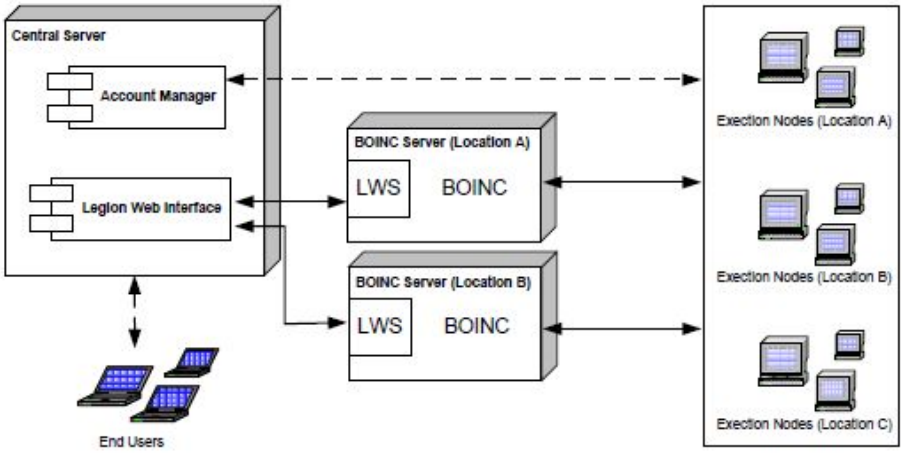

Figure 9. Centralized BOINC Management Architecture

6. Adapting Legion Framework to other Grid Management Systems

It is viable to adapt Legion Framework to work with other Grid Management Systems, for instance Condor. As seen in section 4, the main modification that is needed in order to enable the interaction with other Grid Management System is the development of a customized LWS component for that system. As can be seen in Figure 10, each Legion project is related to a LWS component; indeed, it could be the same LWS for many projects. This makes it possible to enable a different backend for some of the projects registered in the system. The only action required in LWI is to specify the web service address and credentials.

The SwA standard was supported by using the Soaplib library [11] which implements WSDL 1.1 allowing attaching binary files.

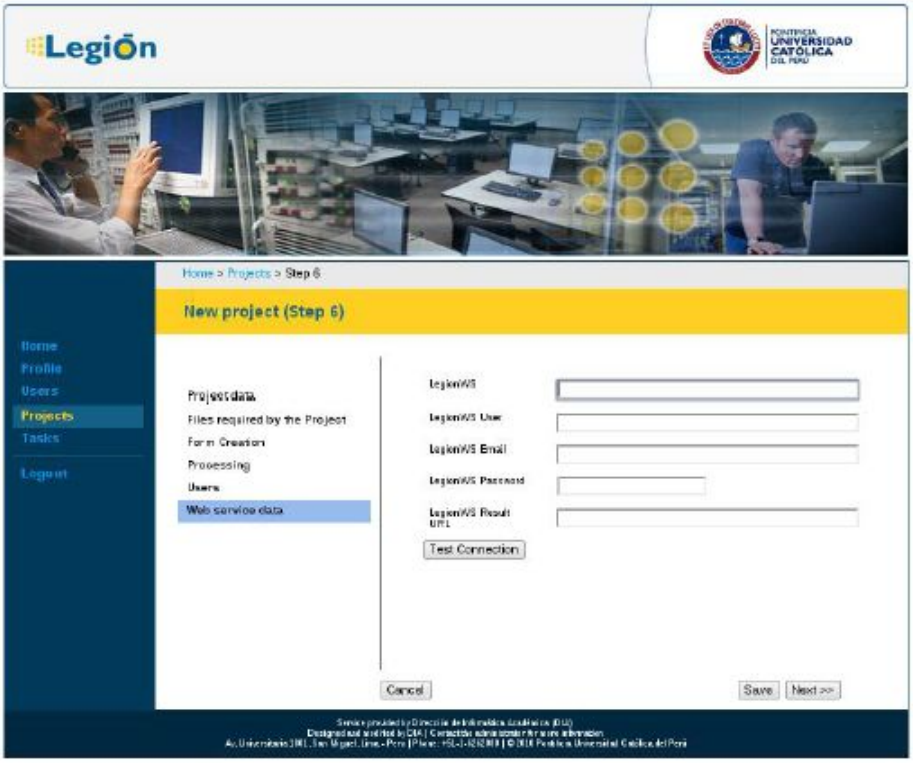

Figure 10. LWS customization step in project creation

7. Future Work

We plan to continue the development of Legion Framework towards the integration with other Grid Management Systems. A next step will be the integration with the Condor Project [12] a major HTC platform.

8. Conclusions

In this paper we presented Legion Framework as a tool both for the side of the grid-admin and for the end user. To sum up, we list the main achievements of the project:

a) Legion Framework successfully hides the complexity of a BOINC Desktop Grid with web interfaces that let end users to stay focused on their core research/activities and reduce simulations feedback cycle.

b) Legion Framework successfully reduces the administration time required to build a web interface for managing BOINC.

c) Legion Web Interface, a key component of Legion Framework, can be easily adapted to other Grid Management Systems.

d) Legion Web Services for BOINC, also a key part of Legion Framework, will let developers to benefit from having a BOINC backend for their custom applications.

e) Legion CLI for BOINC (both PHP and Java Version) will enable users to remotely send tasks to a BOINC server.

f) Legion Framework is suitable for centralized BOINC Servers Management.

In particular, the time required for launching a web interface for computational intensive projects was reduced from a few weeks to a few days.

The service is available at http://legion.pucp.edu.pe.

9. References

1. University of California at Berkeley. "BOINC - Berkeley Open Infrastructure for Network Computing". Available at: http://boinc.berkeley.edu
2. G. Rios, M. Iberico, O. Díaz. "Legion Grid Computing System ". Proceedings of the ULA, 2009, Venezuela.
3. Giorgino, Harvey, Fabritiis. Distributed computing as a virtual supercomputer: tools to run and manage large-scale BOINC simulations. *Dynamics, High Performance Computing*. Elsevier. 2010. pp. 1402-1409
4. Bruno Schulze, Workgroup Proposal: VCG - Virtual Community Grid, Rede Nacional de Pesquisa e Ensino, 2006.
5. "Jarifa". Project Page. Available at: https://github.com/teleyinex/jarifa/wiki
6. W3C. "SOAP Messages with Attachments". Available at: http://www.w3.org/TR/SOAP-attachments
7. The Apache Software Foundation. "Apache Axis 2". Available at: http://ws.apache.org/axis2/.
8. D.P. Anderson. "BOINC: A System for Public-Resource Computing and Storage". *5th IEEE / ACM International Workshop on Grid Computing*, Pittsburgh, PA. Nov. 8 2004 pp. 365-372.
9. University of California at Berkeley. "Virtual Campus Supercomputer Center". Available at: *http://boinc.berkeley.edu/trac/wiki/VirtualCampusSupercomputerCenter*
10. University of California at Berkeley. "Assimilator". Available at: http://boinc.berkeley.edu/svn/branches/boinc_core_release_6_8/sched/assimilator.py
11. "Soaplib". Readme. Available at: https://github.com/arskom/rpclib/tree/soaplib-0.8.2
12. University of Wisconsin – Madison. "Condor Project". Available at: http://www.cs.wisc.edu/condor/