GANGA + DIANE + mini-dashboard evaluation document

1. Objectives

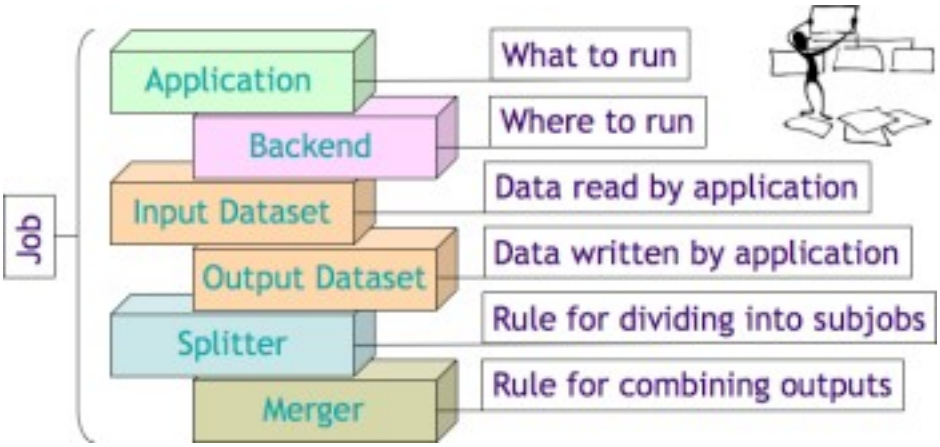
In this document we evaluate the DIANE/GANGA tools integrated with the mini-dashboard framework proposed by the Dashboard team [1]. This work starts with the installation and test of each tool, and follows with a short summary of the advantages and disadvantages offered under the scope of New Emerging Communities, and from two different perspectives: User activities and VO management activities.

2. GANGA

2.1 Introduction to GANGA

GANGA [2] aims to be an easy tool for job submission and management. It is built on python and provides client command tools, a Graphical User Interface (GUI), and a WebGUI. A job in GANGA is constructed from a set of building blocks. All jobs must specify the software to be run (application) and the processing system (backend) to be used. Many jobs will specify an input dataset to be read and/or an output dataset to be produced. Therefore, GANGA provides a framework for handling different types of applications, backends and datasets, implemented as plugin classes. Pragmatically, this means that GANGA can be used to submit jobs to the localhost where it is installed, to a local farm or to a computing grid such as LCG/EGI, as long as the appropriate clients command tools are available to GANGA. From the list of offered default backends, it seems that for grid infrastructures, only the gLite middleware is available. Nevertheless, since it seems to be a very modular tool, other extensions (middlewares) could be easily integrated.

GANGA is presently used by ATLAS and LHCb users among other collaborations. For complete details on how GANGA is used in the framework of those collaborations, please consult [3].



2.2 GANGA Installation

From a user point of view, GANGA can be easily installed under a user home directory without any special privileges. It only requires python 2.3.4 or greater, and needs to have access to the client commands of the backend that it will use. For example, to be able to submit gLite jobs to LCG/EGI infrastructure, GANGA has to be deployed on top of a (properly working) gLite (3.2) User interface.

```
[goncalo@ui01 ~]$ wget http://ganga.web.cern.ch/ganga/download/ganga-install
```

```
[goncalo@ui01 ~]$ python ganga-install --prefix=~/opt/ganga --extern=GangaAtlas,GangaGUI,GangaPlotter 5.5.21
[goncalo@ui01 ~]$ export PATH=/home/ingrid/csys/goncalo/Ganga/opt/ganga/install/5.5.21/bin/:$PATH
```

However, from a VO perspective, case the VO decides to propose this tool to their users, it seems better that the VO offers a central and unique access point for their users to use, in order to avoid multiple (possible unconfigured instances). However, this raises a question about a proper evaluation of the GANGA scalability and performance degradation when used under such shared environments..

2.3 GANGA Client Command Tool

When GANGA is started for the first time, it generates the configuration file ~/.gangarc with default definitions. The GANGA syntax is similar to python's one so if a user is already used to it, it can adapt easily. The user can choose between a whole set of default backends to submit job, implemented as plugin classes, and making it easily to develop and implement additional ones, as for example, other middleware stacks.

```
[goncalo@ui01 ~]$ ganga
In [1]:plugins("backends")
Out[1]: ['LSF', 'Remote', 'PBS', 'Condor', 'SGE', 'Batch', 'LCG', 'Local', 'Interactive']
```

In the example bellow, GANGA is used to submit a simple “Hello World” job to the locahost where it is installed, which is defined as the default backend.

```
In [10]:j = Job(application=Executable(exe='/bin/echo',args=['Hello World']))

In [11]:j.submit()
Ganga.GPIDev.Lib.Job          : INFO    submitting job 40
Ganga.GPIDev.Lib.Job          : INFO    job 40 status changed to "submitting"
Ganga.GPIDev.Adapters         : INFO    submitting job 40 to Local backend
Ganga.GPIDev.Lib.Job          : INFO    job 40 status changed to "submitted"
Ganga.GPIDev.Lib.Job          : INFO    job 40 status changed to "submitted"
Out[11]: 1

In [12]:jobs
Out[12]:
Registry Slice: jobs (1 objects)
--------------
   fqid |   status |     name | subjobs |   application |       backend |                    backend.actualCE
-----------------------------------------------------------------------------------------------------------------
     40 |  running |          |         |   Executable |         Local |                    ui01.ncg.ingrid.pt
Ganga.GPIDev.Lib.Job          : INFO    job 40 status changed to "running"

In [13]:jobs
Out[13]:
Registry Slice: jobs (1 objects)
--------------
   fqid |   status |     name | subjobs |   application |       backend |                    backend.actualCE
-----------------------------------------------------------------------------------------------------------------
     40 | completed |          |         |   Executable |         Local |                    ui01.ncg.ingrid.pt
Ganga.GPIDev.Lib.Job          : INFO    job 40 status changed to "completed"

In [14]:outfile = file(j.outputdir+'stdout')

In [15]:print outfile.read()
Hello World
```

```
In [16]:j.peek('stdout')

In [17]:!cat $j.outputdir/stdout
Hello World
```

In the following example, GANGA is used to submit a job to EGI infrastructure. This is only possible because GANGA has been installed on top of a gLite 3.2 User interface

```
In [2]:j = Job(application=Executable(exe='/bin/echo',args=['Hello World']))

In [3]:config['LCG']['GLITE_ENABLE'] = True

In [4]:j=Job(backend=LCG())

In [5]:j.backend.middleware = 'GLITE'

In [6]:j.submit()
Ganga.GPIDev.Lib.Job          : INFO    submitting job 36
Ganga.GPIDev.Lib.Job          : INFO    job 36 status changed to "submitting"
Ganga.GPIDev.Adapters          : INFO    submitting job 36 to LCG backend
Ganga.GPIDev.Lib.Job          : INFO    job 36 status changed to "submitted"
Ganga.GPIDev.Lib.Job          : INFO    job 36 status changed to "submitted"
Out[6]: 1

In [7]:jobs
Out[7]:
Registry Slice: jobs (1 objects)
--------------
   fqid |   status |     name | subjobs |   application |       backend |                          backend.actualCE
-----------------------------------------------------------------------------------------------------------------------
     36 | submitted |          |         |    Executable |           LCG |ce131.cern.ch:2119/jobmanager-lcglsf-grid_2nh
In [8]:jobs
Out[8]:
Registry Slice: jobs (1 objects)
--------------
   fqid |   status |     name | subjobs |   application |       backend |                          backend.actualCE
-----------------------------------------------------------------------------------------------------------------------
     36 |  running |          |         |    Executable |           LCG |ce131.cern.ch:2119/jobmanager-lcglsf-grid_2nh
Ganga.GPIDev.Lib.Job          : INFO    job 36 status changed to "running"

In [12]:jobs
Out[12]:
Registry Slice: jobs (1 objects)
--------------
   fqid |   status |     name | subjobs |   application |       backend |                          backend.actualCE
-----------------------------------------------------------------------------------------------------------------------
     36 |  completed |          |         |    Executable |           LCG |ce131.cern.ch:2119/jobmanager-lcglsf-grid_2nh
Ganga.GPIDev.Lib.Job          : INFO    job 36 status changed to "completed"
```

Complete instructions on how to use GANGA client tools please consult [4]

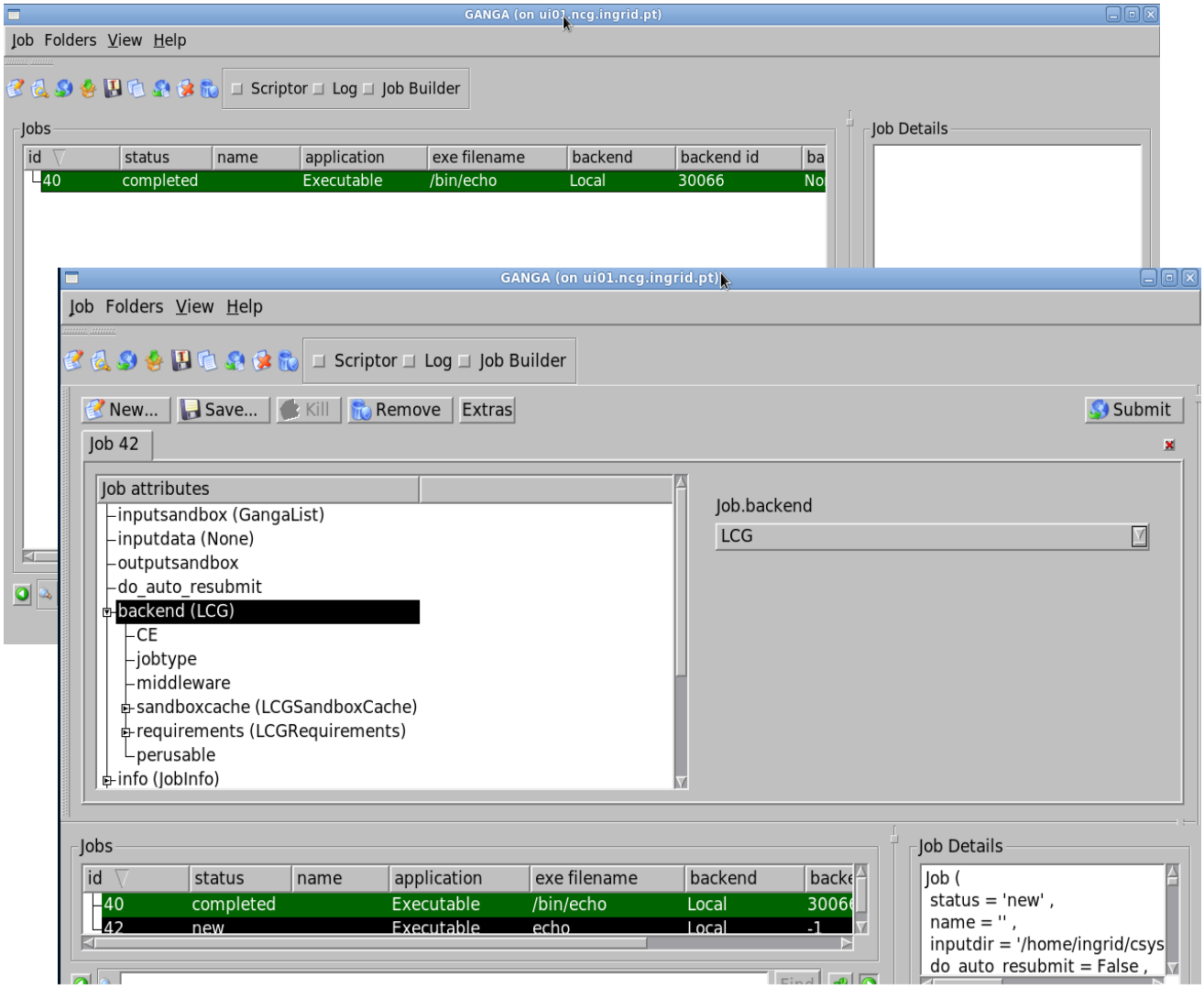
2.4 GANGA GUI and WebGUI

GANGA presents a QT3 based graphical user interface (GUI) and a WebGUI.

GANGA GUI is the GUI (Graphical User Interface) front-end to GANGA allowing users not comfortable working at the console the choice to work in a graphical environment. It is developed with PyQt (Python-bindings to the Qt graphical toolkit) and is built on top of the GANGA GPI (GANGA Public Interface). GANGA GUI not only allows the user to build a GANGA job, submit it (i.e. execute it) locally or to a selection of distributed systems (e.g. batch systems, the Grid) and subsequently retrieve the results, it also provides a customisable job monitoring window that keeps track of running jobs and their status, a job management facility to organise past jobs and quick-scripting tools to run favourite code snippets all within an integrated graphical environment.

The evaluation of the WebGUI functionalities is an ongoing work.

For further details on how to use the GUI, please consult [5].



2.5 GANGA Analysis and Evaluation

GANGA is a tool putting the emphasis on users regarding job management. It hides the difficulties of using a certain middleware stack, making the job submission a more transparent process. It is highly extensible making it easy to incorporate new middleware stacks, and offers a nice Graphical User Interface which decreases the user learning curve to use the VO infrastructure. On the other hand, it introduces a new python-like syntax (in the client command tools), and represents an additional layer of

software increasing an already complex system. It is also not completely clear if GANGA offers the exact same functionalities of the bellow backends such as, for example, submission of MPI jobs.

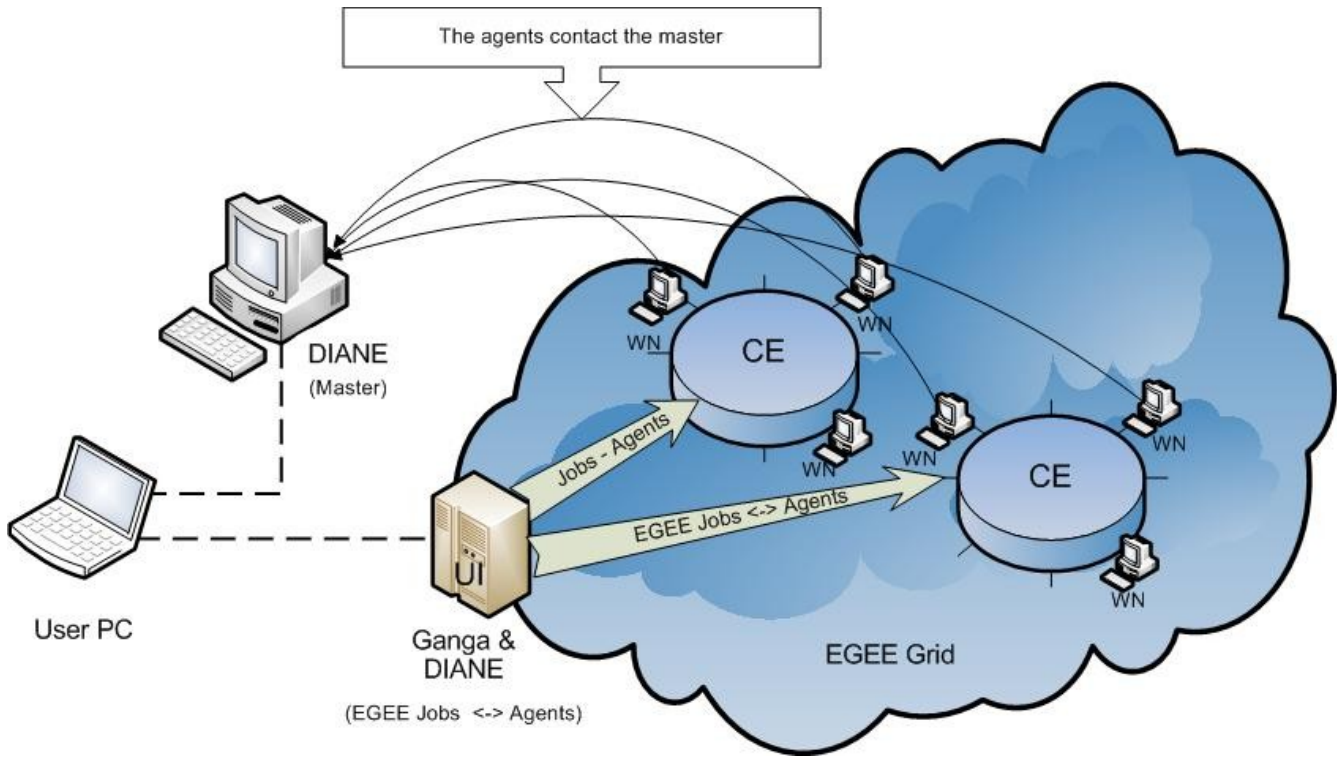
From a VO perspective, it would be better if a central installation is offered to all users, since the correct usage of the tool seems to be dependent of how the user itself configures it. This raises a question about the evaluation of scalability and performance degradation of GANGA.

3. DIANE

3.1 Introduction to DIANE

DIANE [6] is a lightweight job execution control framework for parallel scientific applications aiming to improve the reliability and efficiency of job execution by providing automatic load balancing, fine-grained scheduling and failure recovery. The backbone of DIANE communication model is based on master-worker architecture. This approach is also known as agent-based computing or pilot jobs in which a set of worker agents controls the resources. The resource allocation is independent from the application execution control and therefore may be easily adapted to various use cases. DIANE uses the GANGA to allocate resources by sending worker agent jobs, hence the system supports a large of computing backends: LSF, PBS, SGE, Condor, LCG/EGI Grid.

As opposed to standard message passing libraries such as MPI, the DIANE framework takes care of all synchronization, communication and workflow management details on behalf of the application. The execution of a job is fully controlled by the framework which decides when and where the tasks are executed. Thus the existing applications are *very simple to interface* as python plugin modules. Application plugin modules contain only the essential code directly related to the application itself without bothering about networking details.



3.2 DIANE installation

DIANE [6,7] may be installed under a user home directory without any special privileges. Since it automatically pulls, installs and configures GANGA, the same requirements apply: python 2.3.4 or greater, and the need to access to backend client commands that the system will use. However, DIANE execution includes an additional network configuration step: The master thread has to allow inbound connections in a dedicated TCP port (20500 in the example bellow).

```
[goncalo@ui01 Diane]$ wget http://cern.ch/diane/packages/diane-install
[goncalo@ui01 Diane]$ python diane-install 2.2
[goncalo@ui01 Diane]$ /home/ingrid/csys/goncalo/diane/install/2.2/bin/diane-env -d bash
[goncalo@ui01 Diane]$ export PATH=/home/ingrid/csys/goncalo/diane/ganga/install/5.5.2/bin:$PATH
[goncalo@ui01 Diane]$ export ORBendPoint=giop:tcp::20500
```

3.2 DIANE client command tools

In the following example we will try to demonstrate how a user can benefit from the DIANE framework through their command client tools. Suppose that a user has a hello script that looks like:

```
[goncalo@ui01 Diane]$ cat hello
#!/usr/bin/env bash
rm -f message.out
echo hello $* > message.out
echo "I said hello $* and saved it in message.out"
```

If the user wants to run 20 times this "hello" executable script, changing its arguments every time, he should define the work to be done using a run file which is a simple python file:

```
[goncalo@ui01 Diane]$ cat hello.run
# tell DIANE that we are just running executables
# the ExecutableApplication module is a standard DIANE test application
from diane_test_applications import ExecutableApplication as application

# the run function is called when the master is started
# input.data stands for run parameters
def run(input,config):
        d = input.data.task_defaults # this is just a convenience shortcut
        # all tasks will share the default parameters (unless set otherwise in individual task)
        d.input_files = ['hello']
        d.output_files = ['message.out']
        d.executable = 'hello'

        # here are tasks differing by arguments to the executable
        for i in range(20):
                t = input.data.newTask()
                t.args = [str(i)]
```

At this point, the DIANE master is ready to be started. The master will start in its own run directory typically located in `~/diane/runs/nnn`. The default location may be changed with `$DIANE_USER_WORKSPACE` environment variable.

```
[goncalo@ui01 Diane]$  diane-run hello.run &
```

```
[1] 6791
2011-01-20 18:37:36,462: run directory: /home/ingrid/csys/goncalo/diane/runs/0004
2011-01-20 18:37:36,462: this stderr and stdout is stored in:
/home/ingrid/csys/goncalo/diane/runs/0004/master.stdouterr
2011-01-20 18:37:36,463: full log is stored in: /home/ingrid/csys/goncalo/diane/runs/0004/master.log
2011-01-20 18:37:36,541: new_task_created: tid=1  []
(...)
2011-01-20 18:37:36,560: new_task_created: tid=20  []
```

Once the master is up and running, the user can start worker agents directly in GANGA, submitted as normal jobs. Each of the worker agent jobs can process multiple diane tasks. If you have many worker agent jobs the run completion time will be shorter. If you have less worker agent jobs or if some of the worker jobs crash for some reason than the only noticeable effect will be the slowdown of the run but everything will continue to run without you intervention. You may also add new worker agents at any time.

The following command will run 2 worker agents locally on your computer. After a while, the processing should be terminated and the user should be ready to see the results. All results are stored by the master in the run directory (this behavior may be customized and depends on the application plugins).

```
[goncalo@ui01 Diane]$ ganga LocalSubmitter.py --diane-worker-number=2
*** Welcome to Ganga ***
Version: Ganga-5-5-2
(...)
*******************************************************************************
DIANE Ganga Submitter
INFO: the workers will connect to the master specified by /home/ingrid/csys/goncalo/diane/runs/0004/MasterOID
2011-01-20 18:37:56,331: worker 1 has been initialized and is now ready
2011-01-20 18:37:56,355: worker 2 has been initialized and is now ready
2011-01-20 18:37:56,711: task 1 completed (application_label='')
(...)
2011-01-20 18:37:59,242: task 20 completed (application_label='')
[1]+  Done                    diane-run hello.run
```

Once the tasks are completed, the user can access to the job outputs stored in the master working directory:

```
goncalo@ui01 Diane]$ ll /home/ingrid/csys/goncalo/diane/runs/0004/output_files/
total 80
drwxr-xr-x 2 goncalo csys 4096 Jan 20 18:37 00001
(...)
drwxr-xr-x 2 goncalo csys 4096 Jan 20 18:37 00020

[goncalo@ui01 Diane]$ ll /home/ingrid/csys/goncalo/diane/runs/0004/output_files/00001/
total 8
-rw-r--r-- 1 goncalo csys  8 Jan 20 18:37 message.out
-rw-r--r-- 1 goncalo csys  0 Jan 20 18:37 _stderr
-rw-r--r-- 1 goncalo csys 43 Jan 20 18:37 _stdout
```

To submit to a different backend, a different plugin must be invoked when starting the working agents in ganga. For example, to submit to LCG/EGI infrastructure, one should use:

```
[goncalo@ui01 Diane]$  diane-run hello.run &
```

```
[goncalo@ui01 Diane]$ ganga LCGSubmitter.py –diane-worker-number=2
```

For more information on how to use GANGA, please consult [8].

3.3 DIANE Analysis and Evaluation

DIANE has showed to be valuable tool enabling the successful execution of large numbers of production jobs. From a VO perspective, this seems to be an important added value if a production activity is something foreseen by the VO.

However, from the user point of view, it offers a decoupled framework to the user where the startup of the master server and working instances is delegated to the user, which has to do it in two separate steps. Also, to use this framework, the user has to know the concept of the pilot framework, understand it benefits, and understand what the master, the tasks and the agents concept. There is more to it than only "pushing a button", which is what the majority of the newly users will want. Moreover, since DIANE implements a pilot approach (although without the authentication problems seen in the WLCG context), the VO infrastructure provider will still need to approve the conditions in which their resources will be used.

It is also not completely clear which backend submitter plugins are available to use as arguments to GANGA, and what are their capabilities. Documentation on how to produce new plugins should be in place. We also have not seen any reference if it is possible to use the GANGA GUI under this framework (to submit and control the agents).

4. Mini-Dashboards

4.1 Mini-Dashboard Introduction

The mini-Dashboard monitoring service provides a web-based interface where users may easily keep track of GANGA [10] and DIANE [11] jobs. The mini-Dashboard is a service which runs a simple mysql DB at the backend (as opposed to Oracle DB used by HEP VOs) but uses the same web interface technology as HEP VOs. This web2.0 technology (hBrowseFramework) allows easily customizing and expanding the views via a settings file. For some settings (e.g. selection and ordering of columns) this may be done even without deploying a separate service instance - so with very little overhead from a VO.

The current web interface of mini-Dashboard offered @ CERN has not been extensively configured and it is quite basic. It is meant to grow together with the new user communities, by integrating their customizations and contributions if they are of general interest. There are many levels of customization of this system you may achieve for a VO, depending on their needs and also effort they want to spend on it. With some customization effort, the VO users may consult job statuses and other information (which may be easily added via a configuration file) via graphical summaries and charts. In the development plan, the mini-dashboard supporters plan to add charts such as status pie-charts, but they prefer that further development requirement come as external contributions from interested parties.

The VO may also desire to install their own monitoring instance, customizing and configure it according to their own specific needs. There are some mini-dashboard instances in production (for ATLAS) pointed by the mini-dashboard developers to understand the present functionalities and customization levels in place. However, every time I've tried to access the system, it was unavailable.

4.2 Setup GANGA/DIANE to report to the Mini-Dashboards

Report of jobs/tasks to the mini-dashboards is done using the ActiveMQ messaging system. While DIANE sents messages automatically, GANGA client command tools must be configured according to:

```
.gangarc:
[MonitoringServices]
Executable/* = Ganga.Lib.MonitoringServices.MSGMS.MSGMS

[MSGMS]
server = gridmsg101.cern.ch
port = 6163
```

4.3 Mini-Dashboards Analysis and Evaluation

Presently, the Mini-Dashboard instance offered at CERN is a limited framework to users: a user can only check the jobs / tasks he sent, and access their status. The user can not check where the jobs are running, nor check their associated data as charts (a function which seems to exist but not working). However, according to the developers, this instance was only install as a “proof of concept” and can be customized via hBrowseFramework, with inputs / requirements/ effort from the VO, to show additional information, and aggregate under a graphical view (charts, pie charts) information which is found relevant for the users. According to the developers, there are already production instances used by some communities (ATLAS) with highly configured views, but it was not possible to check them due to the unavailability of those services.

From a VO perspective, one still has to understand the effort on the customization of a specific VO request, and on the operation of this mini-dashboards, if the VO decides to deploy a separate instance. It was also not clear if a VO responsible could have an historical integrated view of the usage (active jobs, pending jobs, failed jobs, etc). Finally, it seems the platform is very user centered, and does not offer many additional added values from the VO Management point of view, which is one of the final aims of this work.

5. Conclusions

GANGA and DIANE, working as standalone tools, hide the most complex aspects of the grid environment, and decrease the slope of what used to be a time consuming learning curve for users. VOs could enhance this further offering to their users a central installation that they can use; assuming the responsibility of the operation of the system instead of delegating it to the user side. Although with some restrictions that each VO must analyze, these tools seems appropriate to process large amounts of production jobs with high reliability and success rates, and boost user access to grid resources.

The mini-dashboard framework is very user centered, providing a solution for monitoring of tasks and jobs for a VO. From the instance tested at CERN, it was not possible to understand how customized and configurable this system can be, and therefore inferred the advantages for VO users. However, according to the supporters, there is great margin of progress and enhancement if VO communities also decide to deliver some degree of commitment and effort on this work.

However, the following problem was identified: The EGI DoW says: "*To simplify access to the infrastructure and to promote collaboration within the VO, EGI.eu will (...) operate access to*

dashboard infrastructure where the status of the resource fabric being used by a particular VO will be reported upon. Both the portal and dashboard offered by this activity will be basic, but they will provide a core framework around which the particular community can, through their own work, customize their web presence and VO specific monitoring of the infrastructure. The dashboard infrastructure will be based on the work being undertaken in Section 1.3.3.4.3.1: TSA3.2.1 Dashboards." This particular solution doesn't seem to fullfill the requirement "*a dashboard infrastructure where the status of the resource fabric being used by a particular VO will be reported upon*", and *"... customize their web presence and VO specific monitoring of the infrastructure*". I do not see how and where the previous framework does offer those functionalities. It seems that the people who wrote that paragraph had in mind the same kind of solution as used for HUCS, but much simpler and naive. However, it seems ORACLE is preventing that from happening. There is still missing a mechanism where emerging user communities can have an integrated view of the state of their infrastructure.

6. Summary table

	User		VO	
	Pros	Cons	Pros	Cons
GANGA	1./ Easy installation 2./ Extensible for integration of other backends 3./ Easy command tools (if you are used to python) 4./ Easy GUI with the capacity to re-use jobs and job templates.	1./ One more abstraction software layer on top of the middleware 2./ Effort on learning a new syntax language (if a user is not used to python) 3./ Not clear if it supports all middleware functionalities (ex: MPI job submissions)	1./ The VO may want to offer a central installation to be used by all users 2./ GANGA GUI may increase the user learning curve on using the VO infrastructure	1./ If not installed centrally, the right use of the tool depends on how the user configures it 2./ A central installation of the tool opens questions about GANGA scalability and performance degradation
DIANE	1./ Easy installation 2./ Increase reliability and success rate for job management	1./ Users need to deeply understand the framework since it is up to the user to start the master thread and the working agents separately 2./ Unknown if GANGA GUI can be used to start and control the agents 3./ Missing documentation regarding the GANGA submitter plugins	1./ Proper for VO production needs 2./ The VO may want to offer a central installation to be used by all users	1./ If not installed centrally, the right use of the tool depends on how it is configured locally 2./ Pilot framework must be agreed by the VO infrastructure providers
Mini-	1./ Possibility to have	1./ The current instance	1./ VOs could install	1./ Needs to

Dashboards	an aggregated / integrated graphical customized view of individual user usage at a given time	at CERN offers very limited functionalities.	their own instance, customize it for VO specific needs.	understand the effort on operation and customizations 2./ Not able to offer an historical view in terms of integrated usage metrics 3./ Platform is very user centered, and does not offer many additional added value from the VO Management point of view

6. References

[1] https://twiki.cern.ch/twiki/bin/view/ArdaGrid/EGIIntroductoryPackage
[2] http://ganga.web.cern.ch/ganga/
[3]http://www.google.pt/url?sa=t&source=web&cd=4&ved=0CCkQFjAD&url=http%3A%2F%2Fhomepages.physik.uni-muenchen.de%2F~Johannes.Elmsheuser%2Fdocs%2Fchep10.pdf&ei=sDpFTauHBcXa4Aai-Yw2&usg=AFQjCNEmHgoSZFlt3UIGWRTSS8vHlZsPgA
[4] http://ganga.web.cern.ch/ganga/user/html/GangaIntroduction/
[5] http://ganga.web.cern.ch/ganga/user/html/GUI_User_Manual/
[6] http://it-proj-diane.web.cern.ch/it-proj-diane/
[7] http://groups.google.com/group/diane-announcements/feed/rss_v2_0_msgs.xml
[8] https://twiki.cern.ch/twiki/bin/view/ArdaGrid/DIANETutorial#DIANE_Tutorial
[9] https://twiki.cern.ch/twiki/bin/view/ArdaGrid/DIANEQuestionsAndAnswers
[10] http://gangamon.cern.ch/ganga
[11] http://dianemon.cern.ch/diane