

# Pilot Jobs

Last review date	Reviewer
2010-03-25	Marco Bencivenni Enrico Fattibene

---

## Table of Contents

### [Pilot Jobs](#)

[Concept of Pilot jobs](#)

[General architecture](#)

[DIANE: Distributed Analysis Environment](#)

[GANGA](#)

[How to run an application using Ganga/ DIANE: Example on Bioinformatics](#)

## Pilot Jobs

This document describes a use case on pilot jobs using Ganga and DIANE, extracted from the biomed VO of EGEE. This use case includes a brief introduction, the installation steps, some recommendations, a sample case of executing BLAST on the piloting schema and the references.

### Concept of Pilot jobs

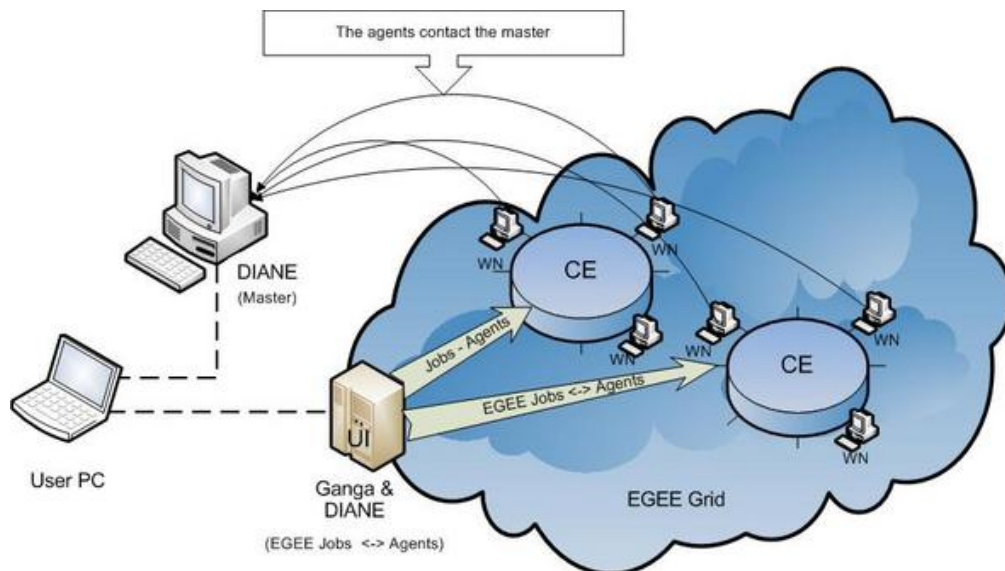
Job submission in gLite Grid environments rely on using metaschedulers (such as WMPProxy, Gridway, the former Resource Broker, etc.), which directly interact with the Local Resource Management Systems (LRMS) installed at the different sites offering resources. Therefore, this process requires, at least, two queuing and scheduling processes, which leads to an additional overhead. Notwithstanding this process fits very well large experiments involving long- time running jobs, real- time needs or massive short running jobs will find this approach inefficient. Moreover, when jobs can exploit locality of reference (such as massive jobs using a single large database for their processing), piloting jobs can group several jobs on the same node, notably reducing the pre- processing overhead needed by each job. Finally, the dynamic nature of master/ slave scheduling could improve general behaviour.

Therefore, strategies of piloting or master / slave submission, in which the jobs submitted, are the executors that communicate with central managers that dynamically dispatch several jobs that are executed without additional queuing and scheduling.

### General architecture

A general architecture of a pilot jobs schema has the following main components:

- Worker Agents, as jobs submitted on the resources which execute the jobs received.
  - Master process, which submits the different jobs to the agents.
-



When the process starts, a master process (a server) is started locally, which will provide tasks to the worker nodes until all the tasks have been completed, being them dismissed. On the other side, the worker agents are jobs running on the Working Nodes of the Grid which communicate with the master. The master must keep track of the tasks to verify that all of them are successfully completed while workers offer the access to a CPU, previously reached through scheduling.

If for any reason, a task fails or a worker losses contact with the master, the master will immediately reassign the task to another worker. Therefore, in this way, a worker job could process more than one task.

However, before initiating the process or execution of the master/ worker jobs, it is necessary to define their characteristics. First, the specification of a run must include the master configuration (workers and heartbeat timeout). It is also necessary to establish master scheduling policies such as the maximum number of times that a lost or failed task is assigned to a worker; the reaction when a task is lost or fails; and the number of resubmissions before a worker is removed. Finally, the master must know the arguments of the tasks and the files shared by all tasks (executable and auxiliary files) and the files that are specific for each task (basically the input sequence chunk). Once started, the master will wait for incoming connections from the workers.

However there are also some disadvantages that must be mentioned. The first issue refers to the unidirectional connectivity between the master host and the worker hosts (Grid node). While the master host needs inbound connectivity, the worker node needs outbound connectivity. The connectivity problem in the master can be solved easily by opening a port in the local host; however the connectivity in the worker will rely on the remote system configuration. So this extra detail must be taken into account when selecting the computing resources. Another issue is defining an adequate timeout value. If, for some reason, a task working correctly suffers from temporary connection problems and exceeds the timeout threshold it will cause the worker being removed by the master. Finally, a key factor will be to identify the rightmost number of worker agents and tasks.

Ganga and DIANE are two environments developed by the ARDA team that assist on the submission of large amounts of jobs on a pilot based system. It has been successfully used in different disciplines and VOs.

## DIANE: Distributed Analysis Environment

DIANE is a lightweight job execution control framework for parallel scientific applications. DIANE improves the reliability and efficiency of job execution by providing automatic load balancing, fine- grained scheduling and failure recovery.

### Installation

DIANE is developed, compiled and tested on Scientific Linux 4 (slc4\_amd64\_gcc34) however it will work on any Unix- based platform. DIANE has intentionally few package dependencies and should run without problems on GNU/ Linux, Solaris, MacOS X etc.

The only compiled packaged which is needed by DIANE is the omniORB library ([http:// omniorb.sourceforge.net](http://omniorb.sourceforge.net)). For user convenience diane install script will try to download it automatically and put it inside the diane installation tree. So in many cases a users does not need to do anything more - just install and run. Use diane- install -- help to see all the installation options.

## GANGA

Ganga is a front- end for job definition and management, implemented in Python.

## Installation

First download the Ganga installation script: `ganga-install`, a given version of Ganga for `slc4_gcc34` can then be downloaded and installed using:

```
python ganga-install [OPTIONS] VERSION
```

Where you need python 2.3.4 or greater. You can optionally request installation of additional Ganga packages, and the external packages on which they depend. The following are strongly recommended:

```
--extern=GangaGUI,GangaPlotter
```

For information on additional options, use:

```
python ganga-install -help
```

## How to run an application using Ganga/ DIANE: Example on Bioinformatics

Next, it is described how to use the DIANE/ Ganga framework for executing a typical Bioinformatics application: performing the alignment of novel sequences against reference databases with BLAST (Basic Local Alignment Search Tool) from the NCBI.

The problem consist on executing the script `gridBLAST.sh` on different files from the `./ SEQS` directory, named `seqfile1.sqf.gz`, `seqfile2.sqf.gz`, `seqfile3.sqf.gz`, etc. Thus the execution command would be:

```
gridBLAST.sh seqfile1.sqf.gz biomed
gridBLAST.sh seqfile2.sqf.gz biomed
...
gridBLAST.sh seqfile500.sqf.gz biomed
```

This is for an experiment involving 500 files on the VO `biomed`. The script will have to check if it is the first instance to be executed on the agent, which will require performing the preprocessing tasks (e.g. downloading and compiling the BLAST executables, downloading and formatting the databases), or if not, it can start directly the processing of the input data.

This approach reduces strongly the preprocessing time, improving the locality of reference, and it reduces also the failure rate, since reutilizing a resource that has been recently used will have more chances of success.

In order to launch such experiment, the user should define the characteristics of the execution in a python file, also known as 'run file'.

File `BLAST.run`:

```
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):
    # Definition of masters' policies
    input.scheduler.policy.STOP_IF_FAILED_TASKS = False
    input.scheduler.policy.FAILED_TASK_MAX_ASSIGN = 99999
    input.scheduler.policy.LOST_TASK_MAX_ASSIGN = 99999
    input.scheduler.policy.REMOVE_FAILED_WORKER_ATTEMPTS = 1
    d = input.data.task_defaults # This is just a convenience shortcut

    # Definition of common parameters for all tasks
    d.executable = 'gridBLAST.sh'
    d.output_files = ['std.out','std.err']

    # Definition of particular arguments for all tasks
    for i in range(500):
        t = input.data.newTask()
        t.input_files = ['./SEQs/seqfile'+str(i)+'.sqf.gz','gridBLAST.sh']
        t.args = ['seqfile'+str(i)+'.sqf','biomed']
```

Now the master can be started using the run file as follows:

```
$ diane-run BLAST.run
```

Once the master has been initiated, it is time to submit the worker agents as jobs through Ganga. Each of the worker agent jobs can process multiple DIANE tasks.

```
$ ganga LCGSubmitter.py -diane-worker-number=N
```

This command will run N worker agents on the resources of the Grid and it can be used to further add more resources (worker agents) to the master. The communication among ganga agents and DIANE master is automatically performed through the execution of both commands on the same machine. The data needed to reach the master is stored in a proper directory in the DIANE. Another interesting possibility is querying the master. Here are some commands which directly talk to the master.

```
diane-master-ping : checks if the master is alive,  
diane-master-ping getStatusReport : gets the summary of the master status,  
diane-master-ping getStatusReport : gets more detailed information about the master status,  
diane-master-ping kill : kills the master.
```