MonALISA Extensions Guide

May 18, 2006
Chapter 1

MonALISA Extensions

1.1 Writing new Monitoring Modules

New Monitoring modules can be easily developed. These modules may use SNMP requests or can simply run any script (locally or on a remote system) to collected the requested values. The mechanism to run these modules under independent threads, to perform the interaction with the operating system or to control a snmp session are inherited from a basic monitoring class. The user basically should only provide the mechanism to collect the values, to parse the output and to generate a result object. It is also required to provide the names of the parameters that are collected by this module.


This interface has the following structure:

```java
package lia.Monitor.monitor;

public interface MonitoringModule extends lia.util.DynamicThreadPoll.SchJobInt {

    public MonModuleInfo init( MNode node, String args ) ;
    public String[] ResTypes() ;
    public String getOsName();
    public Object doProcess() throws Exception ;
    public MNode getNode();
    public String getClusterName();
    public String getFarmName();
    public boolean isRepetitive();
    public String getTaskName();
    public MonModuleInfo getInfo();
}
```

The `doProcess` is actually the function that collects and returns the results. Usually the return type is a `Vector` of `lia.Monitor.monitor.Result` objects. It can also be a simple `Result` object.

The `init` function initializes the useful information for the module, like the cluster that contains the monitoring nodes, the farm and the command line parameters for this module. This function is the first called when the farm loads the module.

The `isRepetitive` function tells if the module has to collect results only once or repeatedly. The return values is the `isRepetitive` module boolean variable. If true, then the module is called from
time to time. The repetitive time is specified in the `<farm>.conf` file. If not there, then the default repetitive call time is 30s.

The rest of functions returns different module information.

Examples to generate new modules can be found in `${MonaLisa HOME}/Service/usr code`.

In `usr code/MDS` is an example of writing the received values into MDS. This is done using a unix pipe to communicate between the dynamically loadable java module and the script performing the update into the LDAP server.

Another simple example which simply print all the values on sysout can be found on `usr code/ SimpleWriter`.

Another example to write the values into UDP sockets is in `usr code/UDPWriter`.

## 1.2 Writing new Filters

Filters allow to dynamically create any new type of derived value from the collected values. Es an example it allow to evaluate the integrated trafc over last n minutes, or the number of nodes for which the load is less than x. Filters may also send an email to a list or SMS messages when predefined complex condition occur. These lters are executed in independent threads and allow any client to register for its output. They may be used to help application to react on certain conditions occur, or to help in presenting global values for large computing facilities.

## 1.3 Writing new Agents

Agents are entities loaded on farm that process the monitoring gathered data and communicate between them for resolving a distributed task based on these data.

An agent respects a given interface. Writing an agent actually means creating a class that implements `lia.Monitor.monitor.AgentI` interface. This interface has the following structure:

```java
import lia.Monitor.monitor.AgentInfo;

public interface AgentI {
    public void init(AgentsCommunication comm);
    public void doWork();
    public String getName();
    public String getGroup();
    public String getAddress();
    public AgentInfo getAgentInfo();
    public void processMsg(Object msg);
    public void processErrorMsg(Object msg);
}
```

For an agent to be able to communicate, the agent-to-agent communication environment has to be initiated. An agent can do this by implementing the `init` method. This method is called by the Agents Engine when first loading the agent.

Agents hosted on the monitoring service usually communicate using the agents communication platform created over the tcp connections to all the proxy services. The communication is one reliable, secure, fast and scalable.

The `AgentCommunication` has methods to send agent-to-agent messages (the `sendMsg` method), or agent-to-proxy message (the `sendCtrlMsg` method) for getting information about other agents from the distributed system (the list of agents from a group or the number of agents from a group).

```java
package lia.Monitor.DataCache;
```
public interface AgentsCommunication {
    public void sendMsg (Object o);
    public void sendCtrlMsg (Object o, String cmd);
}

Messages sent between agents are of a specified format:

public class AgentMessage implements java.io.Serializable {
    public Integer messageID;
    public Long timeStamp;
    public Integer messageType;
    public Integer priority;
    public String agentAddrS;
    public String agentAddrD;
    public String agentGroupS;
    public String agentGroupD;
    public Integer messageTypeAck;
    public Object message;
}

In the messages sent between clients there are the following fields:

- messageID - an integer number for messages sequence.
- timeStamp - time in milliseconds when the messages was sent from the source.
- messageType - type of the message.
- priority - messages priority, a number between 1 and 10, default 5. If the priority is high, the message is forwarded faster by the proxy service than the other messages.
- agentAddrS - address of the source agent.
- agentAddrD - address of the destination agent(s). Can be a multicast address sent to all the agents registered in a group.
- agentGroupS - the group of the source agent. If the source agent hasn’t had registered in a group yet, then this field is null. When specified for the first time, the agent registers in the group. If is the first agent that registers in the specified group, then the new group is created in the proxy service.
- agentGroupD - the group of the destination agent.
- messageTypeAck - if its an ACK message, then a confirmation is required when reaching the destination.
- message - the effective message transmitted. Can be any serializing object.

What an agent does is implemented in the doWork function. An agent is loaded on the monitoring service calling the addAgent function from the lia.Monitor.DataCache.AgentsEngine. Anytime an agent is loaded a new execution thread is created. This thread executes the agent’s doWork function.

An agent is identified in the monitoring service by its name. Every agent has to have a unique name. Based on this name and on the monitoring service (hosting service) ID, an agent has a distinct address in the whole distributed system, agentName@farmID. Also, an agent can register itself in an agent group. Agent groups make possible multicast messages sent to all agents registered in a group. If the agent doesn’t want to register in a group, it doesn’t set the group field. All the information about agent’s name, group, address can be known by calling getName, getAddress or getAgentInfo methods. For the last mentioned method, an object of AgentInfo type is returned, containing all the information about an agent. The lia.Monitor.monitor.AgentInfo class has the following structure:

public class AgentInfo {
    public String agentName;
    public String agentGroup;
    public String farmID;
    public String agentAddr;
}
public AgentInfo (String agentName, String agentGroup, String farmID) {
    this.agentName = agentName;
    this.agentGroup = agentGroup;
    this.farmID = farmID;
    this.agentAddr = agentName + "@" + farmID;
}

Messages can be received from other agents from the distributed system. Messages are processed by the `processMsg` method.

If a message sent by the agent couldn’t arrive to the destination, an error message comes to the sending agent to announce it about communication failure. The error message is processed by the `processErrorMsg` method.

An abstract class, `lia.Monitor.Agents.AbstractAgent` exists to simplify the agents development. This class wraps the `AgentI` interface, defining all `AgentI` methods, but `processMsg` and `doWork` methods. There also is a method for messages creating, `public AgentMessage createMsg(int messageId, int messageType, int messageTypeAck, int priority, String agentAddrD, String agentGroupD, Object message)` method.
Chapter 2

User Guides for the MonALISA Modules

In the following pages you can find specific information useful for installing and running the MonALISA modules:

- PN PBS, Condor and LSF - modules that collect information on processing nodes’ status from PBS, Condor and LSF
- OsgVOJO - module that monitors the GridFTP traffic for VOs
- OgVoJobs - module that provides accounting statistics for VOs by collecting information from different queue managers

2.1 The PN PBS, Condor and LSF Modules

2.1.1 General Description

The PN modules offer monitoring information about the processing nodes from a cluster. The metrics provided are a subset of the Ganglia metrics (see section 2), but the information is obtained from a job manager running on the cluster instead of Ganglia. Currently the modules work with Condor, OpenPBS/Torque nd LSF and the commands used to obtain the nodes’ status are:

For Condor:

```bash
condor_status [-pool <server_name>] -l
```

For OpenPBS/Torque:

```bash
pbsnodes [-s <server_name>] -a
```

For LSF:

```bash
bhosts -l
lshosts
```
2.1.2 Results Provided by the Modules

The parameters provided by the PN_Condor and PN_PBS modules are:

PN_Condor/PN_PBS
|____node1
|____node2
|   (parameters)
|   |____NoCPUs
|   |____VIRT_MEM_free
|   |____MEM_total
|   |____Load1
|......
|____nodeN

where:
- **No_CPUs** - the number of CPUs on the node
- **VIRT_MEM_free** - the amount of free memory (physical + swap) in MB
- **MEM_total** - the total amount of physical memory, in MB
- **Load1** - load average for 1 minute on the node

The parameters provided by the PN_LSF module are:

PN_LSF
|____node1
|____node2
|   (parameters)
|   |____NoCPUs
|   |____MEM_free
|   |____MEM_total
|   |____SWAP_free
|   |____SWAP_total
|   |____Load1
|   |____Load15
|......
|____nodeN

where:
- **No_CPUs** - the number of CPUs on the node
- **MEM_free** - the amount of free physical memory (in MB)
- **MEM_total** - the total amount of physical memory, in MB
- **SWAP_free** - the amount of free swap memory, in MB
- **SWAP_total** - the total amount of swap memory, in MB
- **Load1** - load average for 1 minute on the node
- **Load15** - load average for 15 minutes on the node

If the modules are initialized with the Statistics argument, an additional cluster with statistical information about the number of nodes is provided:

For PBS:
2.1. THE PN PBS, CONDOR AND LSF MODULES

Chapter 2. User Guides for the Monalisa Modules

PN_PBS_Statistics
|___Statistics
|    (parameters)
|    |___Total Nodes
|    |___Total Available Nodes
|    |___Total Free Nodes
|    |___Total Down Nodes
|___...

where:

- Total Nodes - total number of nodes registered to the PBS server
- Total Available Nodes - total number of nodes that are currently communicating with the server
- Total Free Nodes - total number of nodes which are in the “free” state (can execute incoming jobs)
- Total Down Nodes - number of nodes whose state is unknown to the server

For Condor:

PN_Condor_Statistics
|___Statistics
|    (parameters)
|    |___Total Nodes
|    |___Total Slots
|    |___Total CPUs
|    |___Total Available Slots
|    |___Total Free Slots
|    |___Total Owner Slots
|___...

where:

- Total Nodes - total number of nodes from the Condor pool (a multi-processor machine counts as a single node)
- Total Slots - total number of slots (virtual machines in Condor). For a multi-processor machine, separate virtual machines are usually created for each processor.
- Total CPUs - total number of CPUs (should be equal with the number of slots; if it is not the case, a factor will be set to obtain a correct value).
- Total Available Slots - total number of slots that are available for Condor (i.e., the user is not executing his/her own jobs on them)
- Total Free Slots - total number of nodes which are in the "free" state (can execute incoming jobs)
- Total Owner Slots - number of slots in "Owner" state (the user is executing his/her own jobs on them)

The Statistics cluster also contains a “Status” node which indicates the module’s status (0 if it was executed correctly and non-zero if there was an error).

For LSF:

PN_LSF_Statistics
|___Statistics
|    (parameters)
|    |___Total Nodes
|    |___Total Slots

...
CHAPTER 2. USER GUIDES FOR THE MONALISA MODULES

2.1. THE PN PBS, CONDOR AND LSF MODULES

| | | Total Free Slots
| | | Total Down Nodes
| | | ...

where:

- Total Nodes - total number of nodes
- Total Slots - total number of job slots
- Total Free Slots - total number of free job slots
- Total Down Nodes - number down nodes (nodes for which LSF bhosts does not report the "ok" status)

2.1.3 Compiling and Running the Modules

In order to use the VO accounting modules, you should have MonALISA 1.2.38 or newer.
If you have the OSG distribution and put the modules in your folder in your code folder from MonALISA/Service, it is necessary to source two scripts:

```
./OSG/setup.sh
./OSG/MonaLisa/Service/CMD/ml_env
```

(replace "/OSG" with the path to your OSG directory)

To compile, just run the "comp" script from the modules’ directory:

```
./comp
```

Compiling is only necessary if you use the version of the module from the user_code/ directory.

To enable the modules you should add to the farm configuration file a line of the following form:

```
*<cluster_name>{moduleName, localhost, <arguments>}%<time_interval>
```

where:

- cluster_name - the cluster name for the results that this module produces (PN_Condor or PN_PBS)
- moduleName - name of module: monPN_Condor or monPN_PBS
- <arguments> - list of arguments. The arguments that may be passed to the modules are Statistics and Server.

If the Statistics argument appears in the list of arguments, the module will provide an additional "cluster" that contains statistics about number of nodes in the cluster, as described above.

The Server argument indicates the name of PBS server / Condor central manager that will be queried. For example:

```
Server=lcfg.rogrid.pub.ro
```

is a valid entry for this parameter. If this argument is used for the PN_Condor module, the "condor_status" command will be run with the "-pool" option, and for the PN_PBS module the "pbsnodes" command will be run with the "-server" option. The "Server" argument is optional and it can appear more than once in the list, to specify multiple servers from which information should be collected; if it doesn’t appear, the PBS server / Condor central manager corresponding to the local machine will be used.

The "SlotsFactor" argument can be used for Condor, in order to display correctly the number of CPUs (the Total CPUs result), if it is different from the number of Condor slots. The number of CPUs will be calculated as the number of Condor slots times the SlotsFactor; for example, if you have 100 CPUs and 400 Condor slots, you should set "SlotsFactor = 0.25".
CondorConstraints = <constraints> - with this argument you can specify a constraint expression that will be used with condor_status (for example, CondorConstraints = HasCheckpointing==TRUE). Multiple Condor constraints can be specified with an expression containing "&&", "||", etc. (for example: CondorConstraints = HasCheckpointing==TRUE&&TotalVirtualMachines<4). To use quoted strings in the constraints expressions it is a little more complicated because the quotes should be also quoted with 3 backslashes: CondorConstraints = FileSystemDomain="cithep90.ultralight.org"

Examples:

*PN_Condor{monPN_Condor, localhost}%120
Here, the PN_Condor module is used with the default settings. The information will be obtained from the local Condor central manager and no statistics about the number of nodes will be created.

*PN_Condor{monPN_Condor, localhost, Statistics}%240
In this example the module will provide statistical information about the number of nodes.

*PN_Condor{monPN_Condor, localhost, Server=pccil.cern.ch, Statistics}%80
Here, only information from the Condor manager running on pccil.cern.ch will be collected; statistical information about the number of nodes will also be provided.

*PN_Condor{monPN_Condor, localhost, Server=lcfg.rogrid.pub.ro, Server=wn1.rogrid.pub.ro}%180
In this example the module will provide information collected from the lcfg.rogrid.pub.ro and wn1.rogrid.pub.ro Condor managers.

*PN_Condor{monPN_Condor, localhost, Server=cithep90.ultralight.org, CondorConstraints = HasCheckpointing==TRUE}%80
In this example the module will provide information collected from the cithep90.ultralight.org Condor manager, restricted to the nodes that satisfy the condition HasCheckpointing==TRUE.

*PN_PBS{monPN_PBS, localhost}%120
In this example the PN_PBS module is used with the default settings. The information will be obtained from the local PBS server and no statistics about the number of nodes will be created.

*PN_PBS{monPN_PBS, localhost, Statistics}%120
In this example the module will provide statistical information about the number of nodes.

*PN_PBS{monPN_PBS, localhost, Server=pccil.cern.ch, Statistics}%90
Here, only information from the PBS server running on pccil.cern.ch will be collected; statistical information about the number of nodes will also be provided.

*PN_PBS{monPN_PBS, localhost, Statistics, Server=gw01.rogrid.pub.ro, server=lcfg.rogrid.pub.ro}%180
In this example, information is collected from the gw01.rogrid.pub.ro and lcfg.rogrid.pub.ro servers, and statistical data about the number of nodes is provided.

*PN_LSF{monPN_LSF, localhost, Statistics}%120
In this example the PN_LSF module will provide statistical information about the number of nodes. Note: The verification of the parameter names for these modules is case insensitive (i.e., you can write "statistics" or "Statistics").

When the modules are run, there are some environment variables that should be set, which indicate the location of the available queue managers:

- for PBS: if you have PBS, you should set the PBS_LOCATION variable; this variable should be set such that the path to the pbsnodes command is ${PBS_LOCATION}/bin/pbsnodes.

- for Condor: if you have Condor, you should set the CONDOR_LOCATION variable; this variable should be set such that the path to the condor_status command is ${CONDOR_LOCATION}/bin/condor_status.

- for LSF: if you have LSF, you should set the LSF_LOCATION variable; this variable should be set such that the path to the lshosts command is ${LSF_LOCATION}/bin/lshosts.

If you have the OSG distribution and you sourced the OSG/setup.sh script, all the needed variables are already set and it is not necessary to set any other environment variables.
2.2 The OSG VOIO Module

2.2.1 General Description

The OsgVOIO module holds statistical information about the ftp trafic for OSG. The input and output and the rates represent the value for the last time interval (this interval is set before you run the ML service). These values are displayed in the ML client and in the OSG repository (integrated values).

2.2.2 Compiling and Running the Module

In order to use the VO accounting modules, you should have MonALISA 1.2.38 or newer. If you have the OSG distribution, it is necessary to source two scripts:

```
./OSG/setup.sh
./OSG/MonaLisa/Service/CMD/ml_env
```

(replace “/OSG” with the path to your OSG directory)

To compile, just run the “comp” script from the modules’ directory:

```
./comp
```

Initialization of the VOIO module with node and arguments configuration file entry:

```
*osgVO_IO{OsgVO_IO, localhost, <arguments>}%<TIME>
```

where `<arguments>` is a comma separated list. Accepted arguments are:

- ftplog - gridftp.log
- mapfile=/path-to-mapfile (grid3-user-vo-map.txt)
- debug - argument for displaying debug informations in ML log file. This is an optional argument.

TIME represents the interval in seconds between two calls of doProcess method.

The module needs two environment variables to be set:

- for Globus: if you have Globus, you should set the GLOBUSLOCATION variable. This environment variable should be set by sourcing the setup.sh file form your OSG/ folder.
- for VDT: if you have vdt, you should set the VDTLOCATION variable. This environment variable should be set by sourcing the setup.sh file form your OSG/ folder. For OSG the vdt folder is in OSG folder (OSG/vdt).

For example, in the OSG distribution ftplog and mapfile are:

```
ftplog=/OSG/globus/var/gridftp.log
mapfile=/OSG/monitoring/grid3-user-vo-map.txt
```

In farm’s config file you should put the line:

```
*osgVO_IO{OsgVO_IO, localhost, ftplog=/OSG/globus/var/gridftp.log, mapfile=/OSG/monitoring/grid3-user-vo-map.txt, debug}%120
```
2.2.3 Results Provided by the Module

There are two categories of VO parameters: parameters that represent values obtained in the last time interval (between the previous run of the module and the current one) and parameters that represent rates (calculated as the difference between the current value of a parameter and the value obtained at the previous run, divided by the length of the time interval between runs).

The parameters that represent the values obtained in the last time interval are:

- ftpInput and ftpOutput (in KB) represents the total ftp transfer in the last time interval
- ftpRateIn and ftpRateOut (in KB/s) the rates for ftp traffic.
- ftpInput\_SITENAME and ftpOutput\_SITENAME
- ftpRateIn\_SITENAME and ftpRateOut\_SITENAME the same semantics. The values represent the ftp transfer for a domain (SITENAME) (for example ftpInput\_caltech.edu)

Another type of parameters are the ones representing rates for each ftp traffic (under the VO\_IO\_Transfers cluster). The value of these parameters represent a rate that is the value of one transfer divided by the length of the time interval specified in the log file by START and DATA fields (difference between DATA and START).

There are also some “total” parameters, which represent the sum of the parameters above for all the VOs. Under the VO\_IO\_Totals cluster there is “Status” node that indicates the status of the module (0 if the execution was correct and non-zero in case of error).

2.3 The OsgVoJobs Module

2.3.1 General Description

The OsgVoJobs module collects information from different queue managers in order to obtain accounting statistics for VOs. The current version is able to work with Condor, PBS, LSF and SGE; if there are several queue managers on a cluster, the values obtained from them are summed.

The module parses the output of some specific commands that give information about the current jobs (like condor\_q from Condor, qstat from PBS etc.) and produces results for each job (CPU time consumed, run time, the amount of memory etc.). In the case of Condor and PBS, the history file / accounting logs can be also processed in order to obtain more detailed information.

These results are then added to the statistics made per VO; the association between the Unix account from which a job is run and the VO to which the job belongs is made on the base of a map file which specifies the corresponding VO for each account.

2.3.2 Results Provided by the Module

The module provides two categories of parameters: parameters specific to a single job and parameters for a VO.

**Job Parameters** - The job parameters provided by this module are:
CHAPTER 2. USER GUIDES FOR THE MONALISA MODULES 2.3. THE OSGVOJOBS MODULE

- CPUTime - the CPU time consumed so far by the job, in seconds (available in Condor, PBS, LSF, SGE)
- RunTime - wall clock time, in seconds (available in Condor, LSF and in PBS if the PBQuickMode argument is set to "off" - see the paragraph which describes the module’s arguments)
- Size - the size of the job, in MB (available in Condor, LSF, SGE)
- DiskUsage - the disk usage of the job, in MB (available in Condor)

**VO Parameters:**

There are two categories of VO parameters: parameters that represent values obtained in the last time interval (between the previous run of the module and the current one) and parameters that represent rates (calculated as the difference between the current value of a parameter and the value obtained at the previous run, divided by the length of the time interval between runs).

The parameters that represent values obtained in the last time interval are:

- RunningJobs - the number of running jobs owned by the VO
- IdleJobs
- HeldJobs
- UnknownJobs
- TotalJobs
- SubmittedJobs - the number of jobs submitted in the last time interval
- FinishedJobs
- FinishedJobs_Success - the number of jobs finished successfully (with 0 as exit status); this parameter is provided only if the parsing of the Condor history file / PBS accounting logs is enabled.
- FinishedJobs_Error - the number of jobs finished with error (with non-zero exit status); this parameter is provided only if the parsing of the Condor history file / PBS accounting logs is enabled.
- CPUTime - CPU time in seconds (sum for all the VO’s jobs)
- CPUTimeCondorHist - the CPU time for Condor jobs, obtained from the history file
- RunTime - wall clock time in minutes
- JobsSize - the size of the jobs in MB
- DiskUsage - disk usage for the VO, in MB

The parameters that represent rates are:

- SubmittedJobs_R - rate for the SubmittedJobs parameter
- FinishedJobs_R
- RunTime_R
- CPUTime_R

There are also some “total” parameters, which represent the sum of the parameters above for all the VOs. Under the osgVOJOBS_Totals cluster there are also some nodes that give information about the module’s status:

- Status - indicates the status of the module (0 if the execution was correct and non-zero in case of error).
- ExecTime,<manager_name> - the execution time, in ms, for the job manager command
- TotalProcessingTime - the total amount of time, in ms, needed for the module’s execution.
2.3.3 Compiling and Running the Module

In order to use the VO accounting modules, you should have MonALISA 1.2.38 or newer. If you have the OSG distribution, it is necessary to source two scripts:

```
./OSG/setup.sh
./OSG/Monalisa/Service/CMD/ml_env
```

(replace "/OSG" with the path to your OSG directory)

To compile, just run the "comp" script from the modules’ directory:

```
./comp
```

Compiling is only necessary if you run the version of the module placed in the `usr-code/` directory.

When the module is run, there are some environment variables that should be set, which indicate the location of the available queue managers. Usually, the name of the variables are of the form `<JOB_MANAGER>_LOCATION`.

- for PBS: if you have PBS, you should set the `PBS_LOCATION` variable; this variable should be set such that the path to the `qstat` command is `$(PBS_LOCATION)/bin/qstat`.
- for Condor: if you have Condor, you should set the `CONDOR_LOCATION` variable; this variable should be set such that the path to the `condor_q` command is `$(CONDOR_LOCATION)/bin/condor_q`.
  The module also parses the Condor history file, and it may be necessary that you also set the `CONDOR_LOCAL_DIR` variable, which should indicate the location of the Condor local directory (i.e., the directory that contains `log/`, `spool/` etc.). By default, this directory is assumed to be `$(CONDOR_LOCATION)/local.<machine_name>`. If this is the case in your Condor installation, it is not necessary to set the `CONDOR_LOCAL_DIR` variable. If the Condor history file is not in the Condor local directory, use the module with the `CondorHistoryFile` parameter (see below) to specify the exact path of the history file.
- for LSF: if you have LSF, you should set the `LSF_LOCATION` variable; this variable should be set such that the path to the `bjobs` command is `$(LSF_LOCATION)/bin/bjobs`.
- for SGE: if you have SGE, you should set the `SGE_LOCATION` variable; this variable should be set such that the path to the `qstat` command is `$(SGE_LOCATION)/bin/$(SGE_ARCH)/qstat`. The `SGE_ARCH` variable is either set by the user, or, if the user has not set the variable, its value is determined automatically by running the SGE “arch” utility.

If you have the OSG distribution and you sourced the `OSG/setup.sh` script, all the needed variables are already set and it is not necessary to set any other environment variables.

To enable the module you should add to the farm configuration file a line of the following form:

```
*<cluster_name>{OsgVoJobs, localhost [,arguments]}%<time_interval>
```

where `cluster_name` is the cluster name for the results that this module produces; it is recommended that the cluster name be “osgVO_JOBS”. If you use the module included in the MonALISA service and not the one form `usr_code`, you should replace “OsgVoJobs” with “monOsgVoJobs”. The possible arguments for the module are:

- `doNotPublishJobInfo` - the module will not produce results for each running job, but only VO statistics
- `mapfile=<mapfile>` - the location of the mapfile which contains the associations between user accounts and VOs. By default it is considered to be in `$(MONALISA_HOME)/../monitoring/grid3-user-vo-map.txt.
- `CheckCmdExitStatus = ON | OFF` (default is ON) - flag that specifies whether the module should verify the exit status of the commands that it executes. If this is enabled, the commands’ output is only taken into account if the exit status is 0.
• CondorUseGlobal - if Condor is available, the information will be collected from all the submit machines in the pool (i.e., from all the machines on which there are condor_schedd daemons running). This is done by using the "-g" option for the "condor_q" command.

• Server[n]=<hostname> - if Condor is available, the information will be collected from the submit machine that has the specified hostname. This is done by using the "-name" option for the "condor_q" command. The "Server" argument may appear more than once, to specify multiple submit machines. Together with this argument, the "Pool" argument can be used to specify a Condor pool in which the server is situated. In this case, you should add a number after "Server", which should correspond to the number added after the "Pool" argument (e.g., Server2=cithep90.ultralight.org, Pool2=cithep.ultralight.org). If you do not use the "Pool" argument, it is not required to add a number after "Server" (e.g., Server=cithep90.ultralight.org).

• Pool<n>=<poolname> - specifies a Condor pool in which the Condor server given in the Server<n> argument is situated.

• CondorFormatOption = ON | OFF | ALTERNATIVE (the default is ALTERNATIVE) - this argument specifies whether the condor_q command should be used with the -format option. If the argument's value is ALTERNATIVE, the -format option is used only if the output of the regular condor_q command cannot be parsed correctly.

• CondorConstraints = <constraints> With this argument you can specify, for Condor, a constraint expression that will be used with condor_q (for example, CondorConstraints = JobUniverse==1). Multiple Condor constraints can be specified with an expression containing "&&"-s, "||"-s, etc. (for example: CondorConstraints = JobUniverse==5&&TotalSuspensions<3). To use quoted strings in the constraints expressions it is a little more complicated because the quotes should be also quoted with 3 backslashes: CondorConstraints = Owner=="condor"

• CondorQuickMode - only the "condor_q -l" command will be used to obtain information on the running jobs. By default, "plain" condor_q is also used to obtain a more accurate value of the run time.

• CondorHistoryCheck - the module will parse the Condor history log, computing the additional parameters CPUTimeCondorHist, FinishedJobs_Success, FinishedJobs_Error. You should not use this option if you enabled CondorUseGlobal or if you specified a Server argument, because the history log only provides information about jobs submitted from the local machine. Also, this option should not be used if the module has a CondorConstraint argument.

• NoCondorHistoryCheck - the module will not parse the Condor history log and the parameter CPUTimeCondorHist will not be computed. This is the default behavior.

• CondorHistoryFile=<history_file_location> - this parameter specifies the exact location of the Condor history file. Use it if you enabled CondorHistoryCheck and if the history file is in a non-default location.

• PBSHistoryCheck - the module will parse the PBS accounting logs, computing the additional parameters FinishedJobs_Success, FinishedJobs_Error (based on the jobs' exit status)

• NoPBSHistoryCheck - the module will not parse the PBS accounting logs. This is the default behavior.

• PBSLogDir=<pbs_log_dir_location> - this parameter specifies the exact location of the PBS accounting log directory. Use it if you enabled PBSHistoryCheck if the history file is in the default location /usr/spool/PBS/server_priv/accounting.

• MixedCaseVOs - if this argument is given, the names of the VOs will be displayed with mixed cases (by default, they are displayed in upper cases).

• NoVoStatistics = ON | OFF (default is ON) - if this flag is ON, the jobs of the users that do not belong to any VO will be reported under the name "NO_VO". If the flag is OFF, the jobs will not be reported.

• CanSuspend = ON | OFF (default is OFF) - if this flag is enabled, the module is suspended for a period of time if there are 3 consecutive executions with errors.
Note: The parsing of the module’s argument names is not case sensitive. The “boolean” arguments (the ones having ON/OFF as possible values) are considered to be ON if their names appear in the list without any associated value.

Examples:

*osgVO_JOBS{OsgVoJobs, localhost, mapfile=/mymapfile.txt}%120

Here, the module is initialized with another user-VO mapfile than the default one and will be run at every 120 seconds. Apart from the “non-standard” mapfile, the default settings are used (this means that information will be collected only from the local condor_schedd daemon).

*osgVO_JOBS{OsgVoJobs, localhost, mapfile=/mymapfile.txt, CondorUseGlobal}%180

In this example the module is initialized with a non-default mapfile, and it will execute the “condor_q" command with the "-g" option, thus providing information from all the submit machines in the Condor pool.

*osgVO_JOBS{OsgVoJobs, localhost, Server=tier2b.cacr.caltech.edu}%100

In this example the module will collect information from the Condor submit machine tier2b.cacr.caltech.edu, not from the machine on which it is run.

*osgVO_JOBS{OsgVoJobs, localhost, Server=lcfg.rogrid.pub.ro, Server=wn1.rogrid.pub.ro}%60

Here, the module will collect information from two submit machines: lcfg.rogrid.pub.ro and wn1.rogrid.pub.ro.

*osgVO_JOBS{OsgVoJobs, localhost, PBSHistoryCheck, CondorHistoryCheck, CondorHistoryFile=/home/condor/history}%60

Here, the module is initialized both for Condor and PBS and will process the history logs. The PBS logs are assumed to be in the default location and the Condor history file is in the non-default location /home/condor/history.

*osgVO_JOBS{OsgVoJobs, localhost, Server1=cithep90.ultralight.org, Pool1=cithep.ultralight.org, Server2=cithep91.ultralight.org, Pool2=cithep.ultralight.org}%120

In this example, the module will gather data from the Condor servers cithep90.ultralight.org and cithep91.ultralight.org, situated in the pool named cithep.ultralight.org.

*osgVO_JOBS{OsgVoJobs, localhost, CondorConstraints = JobUniverse==1}%180

Here, the module will only process information for the Condor jobs that satisfy the constraint "JobUniverse==1”.

*osgVO_JOBS{OsgVoJobs, localhost, PBSHistoryCheck=on, PBSLogDir=/pbs/server_priv/accounting}%180

In this example, the module will collect information from the PBS accounting logs, searching for the logs in the directory /pbs/server_priv/accounting.