OAR Documentation

Release 2.5

Bruno Bzeznik, Nicolas Capit, Joseph Emeras, Salem Harrache, M

Feb 11, 2021

CONTENTS

OAR is a versatile resource and task manager (also called a batch scheduler) for HPC clusters, and other computing infrastructures (like distributed computing experimental testbeds where versatility is a key).

OAR is suitable for production use.

OAR is also a support for scientific researches in the field of distributed computing.

See the OAR web site (http://oar.imag.fr/) for further information.

This documentation is organized into a few different sections below:

- *User Documentation*
- *Admin Documentation*
- *Other Documentations*
- *Changelog*.
- Copyright and license.

CHAPTER

ONE

USER DOCUMENTATION

1.1 Using OAR - Basic steps

1.1.1 Visualising the cluster State

Many tools are available to visualize the cluster state.

Shell commands:

- oarstat: this command shows information about running or planned jobs. (The -f option shows full infomation)
- oarnodes: this command shows the resources states. Warning: in our context, a resource is not necessary a machine. It is generally a cpu, a core or a host, but it can be much more… like licence tokens, vlan, … The oarnodes command gives information about the network address where is located this resource, its type, its state and many other (interesting) information.

Graphical tools:

- Monika: this web page shows current resources states and jobs information. On this page you can have more information about a particular resource or job.
- DrawGantt: this web page shows the gantt diagram of the scheduling. It represents the current, former and future jobs.

1.1.2 Submitting a job in an interactive shell

Submission

To submit an interactive job we use the "oarsub" command with the "-I" option:

```
frontend:~$> oarsub -I
```

OAR returns then an unique job ID that will identify your job in the system:

```
OAR_JOB_ID=1234
```

Once the job is scheduled, when the requested resources are available, OAR connects you to the first allocated node. OAR initiates environment variables that inform you of your submission properties:

```
node:~$> env | grep OAR
```

Particularly, the allocated nodes list is contained in the $OAR_NODEFILE:

```
node:~$> cat $OAR_NODEFILE
```

Visualisation

You can get information about your job by looking at the Monika or DrawGantt interfaces or by typing in a command line console:

```
frontend:~$> oarstat -fj OAR_JOB_ID
```

Exiting the job

To terminate an interactive job you just have to disconnect from the resource:

```
node:~$> exit
```

You can likewise kill the job by typing:

```
frontend:~$> oardel OAR_JOB_ID
```

In this case, the session will be killed ("kill -9").

Interactive submission on many resources

The "-l" option allows to specify wanted resources. For example, if we need to work in interactive mode on 2 cpu for a max duration of 30 minutes we will ask:

```
frontend:~$> oarsub -I -l /cpu=2,walltime=00:30:00
```

The walltime is the job's max duration. If the job overruns its walltime, it will be killed by the system. Thus, you better have to set your walltime correctly depending on how long will take your job to prevent being killed if the walltime has been set too short or being scheduled later if it is too long. Then, once the job is scheduled and started, OAR connects you on the first reserved node. You still can access the list of the other resources via the $OAR_NODEFILE env variable.

1.1.3 Batch submission

OAR allows to execute scripts in "passive mode". In this mode, the user specifies a script at the submission time. This script will be executed on the first reserved node. It's within this script that the user will define the way to operate parallel resources. All the `$OAR_*` env variables are reachable within the script.

The script must be executable.

Submission

In this case, the principle is the same that interactive submission, just replace the "-I" option with the path of your script:

```
frontend:~$> oarsub -l /cpu=2,walltime=00:30:00 ./hello_mpi.sh
```

Getting the results of the submission

In passive mode, OAR creates 2 files: OAR.<OAR_JOB_ID>.stdout for the stdout and OAR.<OAR_JOB_ID>.stderr for the stderr. The name of these 2 files can be changed (see “man oarsub”).

Connecting a running job

You can connect a running job with the “-C” option to oarsub:

```
frontend:~$> oarsub -C <OAR_JOB_ID>
```

Thus, you will be connected to the first reserved node.

1.1.4 Reservations

Until now we only asked for immediate start for our submission. However it is also possible to plan a job in the future. This feature is available through the “-r <date>” option:

```
frontend:~$> oarsub -r '2008-03-07 16:45:00' -l nodes=2,walltime=0:10:00 ./hello_mpi.
↪sh
```

1.2 OAR Use Cases

1.2.1 Interactive jobs

Job submission

```
jdoe@idpot:~$ oarsub -I -l /nodes=3/core=1
[ADMISSION RULE] Set default walltime to 7200.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=4924
Interactive mode : waiting...
[2007-03-07 08:51:04] Starting...

Connect to OAR job 4924 via the node idpot5.grenoble.grid5000.fr
jdoe@idpot5:~$
```

Connecting to the other cores

```
jdoe@idpot5:~$ cat $OAR_NODEFILE
idpot5.grenoble.grid5000.fr
idpot8.grenoble.grid5000.fr
idpot9.grenoble.grid5000.fr
jdoe@idpot5:~$ oarsh idpot8
Last login: Tue Mar  6 18:00:37 2007 from idpot.imag.fr
```

(continues on next page)

(continued from previous page)

```
jdoe@idpot8:~$ oarsh idpot9
Last login: Wed Mar  7 08:48:30 2007 from idpot.imag.fr
jdoe@idpot9:~$ oarsh idpot5
Last login: Wed Mar  7 08:51:45 2007 from idpot5.imag.fr
jdoe@idpot5:~$
```

Copying a file from one node to another

```
jdoe@idpot5:~$ hostname > /tmp/my_hostname
jdoe@idpot5:~$ oarcp /tmp/my_hostname idpot8:/tmp/my_hostname
jdoe@idpot5:~$ oarsh idpot8 cat /tmp/my_hostname
idpot5
jdoe@idpot5:~$
```

Connecting to our job

```
jdoe@idpot:~$ OAR_JOB_ID=4924 oarsh idpot9
Last login: Wed Mar  7 08:52:09 2007 from idpot8.imag.fr
jdoe@idpot9:~$ oarsh idpot5
Last login: Wed Mar  7 08:52:18 2007 from idpot9.imag.fr
jdoe@idpot5:~$
```

1.2.2 Batch mode job

Submission using a script

```
jdoe@paramount:~$ oarsub -l core=10 ./runhpl
Generate a job key...
[ADMISSION RULE] Set default walltime to 3600.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=199522
```

Watching results

```
jdoe@paramount:~$ cat OAR.199522.stdout
...
```

Submission using an inline command

Sometimes it is very useful to run a little command in oarsub:

```
jdoe@paramount:~$ oarsub -l core=1 'echo $PATH;which ssh'
Generate a job key...
[ADMISSION RULE] Set default walltime to 3600.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=199523
```

Watching results

```
jdoe@paramount:~$ cat OAR.199523.stdout
...
```

1.2.3 Reservations

The date format to pass to the -r option is YYYY-MM-DD HH:MM:SS:

```
jdoe@paramount:~$ oarsub -l core=10 ./runhpl -r "2007-10-10 18:00:00"
Generate a job key...
[ADMISSION RULE] Set default walltime to 3600.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=199524
Reservation mode : waiting validation...
Reservation valid --> OK
jdoe@paramount:~$
```

1.2.4 Examples of resource requests

Using the resource hierarchy

- ask for 1 core on 15 nodes on a same cluster (total = 15 cores)

```
oarsub -I -l /cluster=1/nodes=15/core=1
```

- ask for 1 core on 15 nodes on 2 clusters (total = 30 cores)

```
oarsub -I -l /cluster=2/nodes=15/core=1
```

- ask for 1 core on 2 cpus on 15 nodes on a same cluster (total = 30 cores)

```
oarsub -I -l /cluster=1/nodes=15/cpu=2/core=1
```

- ask for 10 cpus on 2 clusters (total = 20 cpus, information regarding the node ou core count depend on the topology of the machines)

```
oarsub -I -l /cluster=2/cpu=10
```

- ask for 1 core on 3 different network switches (total = 3 cores)

```
oarsub -I -l /switch=3/core=1
```

Using properties

See OAR properties for a description of all available properties, and watch Monika.

- ask for 10 cores of the cluster azur

```
oarsub -I -l core=10 -p "cluster='azur'"
```

- ask for 2 nodes with 4096 GB of memory and Infiniband 10G

```
oarsub -I -p "memnode=4096 and ib10g='YES'" -l nodes=2
```

- ask for any 4 nodes except gdx-45

```
oarsub -I -p "not host like 'gdx-45.%'" -l nodes=4
```

Mixing every together

- ask for 1 core on 2 nodes on the same cluster with 4096 GB of memory and Infiniband 10G + 1 cpu on 2 nodes on the same switch with bicore processors for a walltime of 4 hours

```
oarsub -I -l "{memnode=4096 and ib10g='YES'}/cluster=1/nodes=2/core=1+{nbcore=2}/
↪switch=1/nodes=2/cpu=1,walltime=4:0:0"
```

Warning

1. walltime must always be the last argument of -l <...>
2. if no resource matches your request, oarsub will exit with the message

```
Generate a job key...
[ADMISSION RULE] Set default walltime to 3600.
[ADMISSION RULE] Modify resource description with type constraints
There are not enough resources for your request
OAR_JOB_ID=-5
Oarsub failed: please verify your request syntax or ask for support to your admin.
```

Moldable jobs

- ask for 4 nodes and a walltime of 2 hours or 2 nodes and a walltime of 4 hours

```
oarsub -I -l nodes=4,walltime=2 -l nodes=2,walltime=4
```

Types of job

OAR features the concept of job "type". For example:

- submit besteffort jobs

```
for param in $(< ./paramlist); do
    oarsub -t besteffort -l core=1 "./my_script.sh $param"
done
```

- ask for 4 nodes on the same cluster in order to deploy a customized environment:

```
oarsub -I -l cluster=1/nodes=4,walltime=6 -t deploy
```

Check the man of oarsub to get the other job types.

1.2.5 X11 forwarding

If you have a DISPLAY configured in your shell then oarsub will automatically forward the X11 to it.

For example:

```
jdoe@idpot:~$ oarsub -I -l /nodes=2/core=1
OAR_JOB_ID=4926
Interactive mode : waiting...
[2007-03-07 09:01:16] Starting...

Initialize X11 forwarding...
Connect to OAR job 4926 via the node idpot8.grenoble.grid5000.fr
jdoe@idpot8:~$ xeyes &
[1] 14656
jdoe@idpot8:~$ cat $OAR_NODEFILE
idpot8.grenoble.grid5000.fr
idpot9.grenoble.grid5000.fr
[1]+  Done                    xeyes
jdoe@idpot8:~$ oarsh idpot9 xeyes
Error: Can't open display:
jdoe@idpot8:~$ oarsh -X idpot9 xeyes
```

1.2.6 Using a parallel launcher: taktuk

Warning: Taktuk MUST BE installed on all nodes to test this point

Shell 1

Unset DISPLAY so that X does not bother...

```
jdoe@idpot:~$ unset DISPLAY
```

Job submission

```
jdoe@idpot:~$ oarsub -I -l /nodes=20/core=1
[ADMISSION RULE] Set default walltime to 7200.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=4930
Interactive mode : waiting...
[2007-03-07 09:15:13] Starting...

Connect to OAR job 4930 via the node idpot1.grenoble.grid5000.fr
```

Running the taktuk command

```
jdoe@idpot1:~$ taktuk -c "oarsh" -f $OAR_FILE_NODES broadcast exec [ date ]
idcalc12.grenoble.grid5000.fr-1: date (11567): output > Thu May  3 18:56:58 CEST 2007
idcalc12.grenoble.grid5000.fr-1: date (11567): status > Exited with status 0
idcalc4.grenoble.grid5000.fr-8: date (31172): output > Thu May  3 19:00:09 CEST 2007
```

(continues on next page)

(continued from previous page)

```
idcalc2.grenoble.grid5000.fr-2: date (32368): output > Thu May  3 19:01:56 CEST 2007
idcalc3.grenoble.grid5000.fr-5: date (31607): output > Thu May  3 18:56:44 CEST 2007
idcalc3.grenoble.grid5000.fr-5: date (31607): status > Exited with status 0
idcalc7.grenoble.grid5000.fr-13: date (31188): output > Thu May  3 18:59:54 CEST 2007
idcalc9.grenoble.grid5000.fr-15: date (32426): output > Thu May  3 18:56:45 CEST 2007
idpot6.grenoble.grid5000.fr-20: date (16769): output > Thu May  3 18:59:54 CEST 2007
idcalc4.grenoble.grid5000.fr-8: date (31172): status > Exited with status 0
idcalc5.grenoble.grid5000.fr-9: date (10288): output > Thu May  3 18:56:39 CEST 2007
idcalc5.grenoble.grid5000.fr-9: date (10288): status > Exited with status 0
idcalc6.grenoble.grid5000.fr-11: date (11290): output > Thu May  3 18:57:52 CEST 2007
idcalc6.grenoble.grid5000.fr-11: date (11290): status > Exited with status 0
idcalc7.grenoble.grid5000.fr-13: date (31188): status > Exited with status 0
idcalc8.grenoble.grid5000.fr-14: date (10450): output > Thu May  3 18:57:34 CEST 2007
idcalc8.grenoble.grid5000.fr-14: date (10450): status > Exited with status 0
idcalc9.grenoble.grid5000.fr-15: date (32426): status > Exited with status 0
idpot1.grenoble.grid5000.fr-16: date (18316): output > Thu May  3 18:57:19 CEST 2007
idpot1.grenoble.grid5000.fr-16: date (18316): status > Exited with status 0
idpot10.grenoble.grid5000.fr-17: date (31547): output > Thu May  3 18:56:27 CEST 2007
idpot10.grenoble.grid5000.fr-17: date (31547): status > Exited with status 0
idpot2.grenoble.grid5000.fr-18: date (407): output > Thu May  3 18:56:21 CEST 2007
idpot2.grenoble.grid5000.fr-18: date (407): status > Exited with status 0
idpot4.grenoble.grid5000.fr-19: date (2229): output > Thu May  3 18:55:37 CEST 2007
idpot4.grenoble.grid5000.fr-19: date (2229): status > Exited with status 0
idpot6.grenoble.grid5000.fr-20: date (16769): status > Exited with status 0
idcalc2.grenoble.grid5000.fr-2: date (32368): status > Exited with status 0
idpot11.grenoble.grid5000.fr-6: date (12319): output > Thu May  3 18:59:54 CEST 2007
idpot7.grenoble.grid5000.fr-10: date (7355): output > Thu May  3 18:57:39 CEST 2007
idpot5.grenoble.grid5000.fr-12: date (13093): output > Thu May  3 18:57:23 CEST 2007
idpot3.grenoble.grid5000.fr-3: date (509): output > Thu May  3 18:59:55 CEST 2007
idpot3.grenoble.grid5000.fr-3: date (509): status > Exited with status 0
idpot8.grenoble.grid5000.fr-4: date (13252): output > Thu May  3 18:56:32 CEST 2007
idpot8.grenoble.grid5000.fr-4: date (13252): status > Exited with status 0
idpot11.grenoble.grid5000.fr-6: date (12319): status > Exited with status 0
idpot9.grenoble.grid5000.fr-7: date (17810): output > Thu May  3 18:57:42 CEST 2007
idpot9.grenoble.grid5000.fr-7: date (17810): status > Exited with status 0
idpot7.grenoble.grid5000.fr-10: date (7355): status > Exited with status 0
idpot5.grenoble.grid5000.fr-12: date (13093): status > Exited with status 0
```

Setting the connector definitively and running taktuk again

```
jdoe@idpot1:~$ export TAKTUK_CONNECTOR=oarsh
jdoe@idpot1:~$ taktuk -m idpot3 -m idpot4 broadcast exec [ date ]
idpot3-1: date (12293): output > Wed Mar  7 09:20:25 CET 2007
idpot4-2: date (7508): output > Wed Mar  7 09:20:19 CET 2007
idpot3-1: date (12293): status > Exited with status 0
idpot4-2: date (7508): status > Exited with status 0
```

1.2.7 Using MPI with OARSH

To use MPI, you must setup your MPI stack so that it uses OARSH instead of the default RSH or SSH connector. All required steps for the main different flavors of MPI are presented below.

MPICH1

Mpich1 connector can be changed using the P4_RSHCOMMAND environment variable. This variable must be set in the shell configuration files. For instance for bash, within ~/.bashrc

```
export P4_RSHCOMMAND=oarsh
```

Please consider setting the P4_GLOBMEMSIZE as well.

You can then run your mpich1 application:

```
jdoe@idpot4:~/mpi/mpich$ mpirun.mpich -machinefile $OAR_FILE_NODES -np 6 ./hello
Hello world from process 0 of 6 running on idpot4.grenoble.grid5000.fr
Hello world from process 4 of 6 running on idpot6.grenoble.grid5000.fr
Hello world from process 1 of 6 running on idpot4.grenoble.grid5000.fr
Hello world from process 3 of 6 running on idpot5.grenoble.grid5000.fr
Hello world from process 2 of 6 running on idpot5.grenoble.grid5000.fr
Hello world from process 5 of 6 running on idpot6.grenoble.grid5000.fr
```

MPICH2

Tested version: 1.0.5p2

MPICH2 uses daemons on nodes that may be started with the "mpdboot" command. This command takes oarsh has an argument (–rsh=oarsh) and all goes well:

```
jdoe@idpot2:~/mpi/mpich/mpich2-1.0.5p2/bin$ ./mpicc -o hello ../../../hello.c
jdoe@idpot2:~/mpi/mpich/mpich2-1.0.5p2/bin$ ./mpdboot --file=$OAR_NODEFILE --
↪rsh=oarsh -n 2
jdoe@idpot2:~/mpi/mpich/mpich2-1.0.5p2/bin$ ./mpdtrace -l
idpot2_39441 (129.88.70.2)
idpot4_36313 (129.88.70.4)
jdoe@idpot2:~/mpi/mpich/mpich2-1.0.5p2/bin$ ./mpiexec -np 8 ./hello
Hello world from process 0 of 8 running on idpot2
Hello world from process 1 of 8 running on idpot4
Hello world from process 3 of 8 running on idpot4
Hello world from process 2 of 8 running on idpot2
Hello world from process 5 of 8 running on idpot4
Hello world from process 4 of 8 running on idpot2
Hello world from process 6 of 8 running on idpot2
Hello world from process 7 of 8 running on idpot4
```

MVAPICH2

You can use the hydra launcher with the options "-launcher" and "-launcher-exec", for example:

```
mpiexec -launcher ssh -launcher-exec /usr/bin/oarsh -f $OAR_NODEFILE -n 4 ./app
```

LAM/MPI

Tested version: 7.1.3

You can use export LAMRSH=oarsh before starting lamboot; otherwise the "lamboot" command takes -ssi boot_rsh_agent "oarsh" option has an argument (this is not in the manual!). Also note that OARSH doesn't automatically sends the environnement of the user, so, you may need to specify the path to LAM distribution on the nodes with this option: -prefix

```
jdoe@idpot2:~/mpi/lam$ ./bin/lamboot -prefix ~/mpi/lam \
                                 -ssi boot_rsh_agent "oarsh" \
                                 -d $OAR_FILE_NODES
jdoe@idpot2:~/mpi/lam$ ./bin/mpirun -np 8 hello
Hello world from process 2 of 8 running on idpot2
Hello world from process 3 of 8 running on idpot2
Hello world from process 0 of 8 running on idpot2
Hello world from process 1 of 8 running on idpot2
Hello world from process 4 of 8 running on idpot4
Hello world from process 6 of 8 running on idpot4
Hello world from process 5 of 8 running on idpot4
Hello world from process 7 of 8 running on idpot4
```

OpenMPI

Tested version: 1.1.4

The magic option to use with OpenMPI and OARSH is "-mca pls_rsh_agent "oarsh"". Also note that OpenMPI works with daemons that are started on the nodes (orted), but "mpirun" starts them on-demand. The "-prefix" option can help if OpenMPI is not installed in a standard path on the cluster nodes (you can replace the "-prefix" option by using the absolute path when invoking the "mpirun" command).

```
jdoe@idpot2:~/mpi/openmpi$ ./bin/mpirun -prefix ~/mpi/openmpi \
                        -machinefile $OAR_FILE_NODES \
                        -mca pls_rsh_agent "oarsh" \
                        -np 8 hello
Hello world from process 0 of 8 running on idpot2
Hello world from process 4 of 8 running on idpot4
Hello world from process 1 of 8 running on idpot2
Hello world from process 5 of 8 running on idpot4
Hello world from process 2 of 8 running on idpot2
Hello world from process 6 of 8 running on idpot4
Hello world from process 7 of 8 running on idpot4
Hello world from process 3 of 8 running on idpot2
```

You can make the option "oarsh" automatically by adding it in a configuration file in the OpenMPI installation directory named "$OPENMPI_INSTALL_DIR/etc/openmpi-mca-params.conf"

```
plm_rsh_agent=/usr/bin/oarsh
```

So, with this configuration, this is transparent for the users.

Note: In OpenMPI 1.6, "pls_rsh_agent" was replaced by "orte_rsh_agent". **Note**: In OpenMPI 1.8, "orte_rsh_agent" was replaced by "plm_rsh_agent".

Intel MPI

Example using the hydra launcher:

```
mpiexec.hydra -genvall -f $OAR_NODE_FILE -bootstrap-exec oarsh -env I_MPI_DEBUG 5 -n␣
↪8 ./ring
```

1.2.8 Tests of the CPUSET mechanism

Processus isolation

In this test, we run 4 "yes" commands in a job whose resources is only one core. (syntax tested with bash as the user's shell)

```
jdoe@idpot:~$ oarsub -l core=1 "yes > /dev/null & yes > /dev/null & yes > /dev/null &␣
↪yes > /dev/null"
[ADMISSION RULE] Set default walltime to 7200.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=8683
```

Then we connect to the node and run ps or top for monitoring purposes:

```
jdoe@idpot:~$ oarsub -C 8683
Initialize X11 forwarding...
Connect to OAR job 8683 via the node idpot9.grenoble.grid5000.fr
jdoe@idpot9:~$ ps -eo fname,pcpu,psr | grep yes
yes      23.2   1
yes      23.1   1
yes      24.0   1
yes      23.0   1
```

This shows that the 4 processus are indeed restricted to the core the job was assigned to, as expected.

Don't forget to delete your job:

```
jdoe@idpot:~$ oardel 8683
```

1.2.9 Using best effort mode jobs

Best effort job campaign

OAR provides a way to specify that jobs are best effort, which means that the server can delete them if room is needed to fit other jobs. One can submit such jobs using the besteffort type of job.

For instance you can run a job campaign as follows:

```
for param in $(< ./paramlist); do
    oarsub -t besteffort -l core=1 "./my_script.sh $param"
done
```

In this example, the file ./paramlist contains a list of parameters for a parametric application.

The following demonstrates the mechanism.

Best effort job mechanism

Running a besteffort job in a first shell

```
jdoe@idpot:~$ oarsub -I -l nodes=23 -t besteffort
[ADMISSION RULE] Added automatically besteffort resource constraint
[ADMISSION RULE] Redirect automatically in the besteffort queue
```

(continues on next page)

(continued from previous page)

```
[ADMISSION RULE] Set default walltime to 7200.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=9630
Interactive mode : waiting...
[2007-05-10 11:06:25] Starting...

Initialize X11 forwarding...
Connect to OAR job 9630 via the node idcalc1.grenoble.grid5000.fr
```

Running a non best effort job on the same set of resources in a second shell

```
jdoe@idpot:~$ oarsub -I
[ADMISSION RULE] Set default walltime to 7200.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=9631
Interactive mode : waiting...
[2007-05-10 11:06:50] Start prediction: 2007-05-10 11:06:50 (Karma = 0.000)
[2007-05-10 11:06:53] Starting...

Initialize X11 forwarding...
Connect to OAR job 9631 via the node idpot9.grenoble.grid5000.fr
```

As expected, meanwhile the best effort job was stopped (watch the first shell):

```
jdoe@idcalc1:~$ bash: line 1: 23946 Killed                 /bin/bash -l
Connection to idcalc1.grenoble.grid5000.fr closed.
Disconnected from OAR job 9630
jdoe@idpot:~$
```

1.2.10 Testing the checkpointing trigger mechanism

Writing the test script

Here is a script feature an infinite loop and a signal handler trigged by SIGUSR2 (default signal for OAR's checkpointing mechanism).

```
#!/bin/bash

handler() { echo "Caught checkpoint signal at: `date`"; echo "Terminating."; exit 0; }
trap handler SIGUSR2

cat <<EOF
Hostname: `hostname`
Pid: $$
Starting job at: `date`
EOF
while : ; do sleep 1; done
```

Running the job

We run the job on 1 core, and a walltime of 1 hour, and ask the job to be checkpointed if it lasts (and it will indeed) more that walltime - 900 sec = 45 min.

```
jdoe@idpot:~/oar-2.0/tests/checkpoint$ oarsub -l "core=1,walltime=1:0:0" --checkpoint␣
↪900 ./checkpoint.sh
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=9464
jdoe@idpot:~/oar-2.0/tests/checkpoint$
```

Result

Taking a look at the job output:

```
jdoe@idpot:~/oar-2.0/tests/checkpoint$ cat OAR.9464.stdout
Hostname: idpot9
Pid: 26577
Starting job at: Fri May  4 19:41:11 CEST 2007
Caught checkpoint signal at: Fri May  4 20:26:12 CEST 2007
Terminating.
```

The checkpointing signal was sent to the job 15 minutes before the walltime as expected so that the job can finish nicely.

Interactive checkpointing

The oardel command provides the capability to raise a checkpoint event interactively to a job.

We submit the job again

```
jdoe@idpot:~/oar-2.0/tests/checkpoint$ oarsub -l "core=1,walltime=1:0:0" --checkpoint␣
↪900 ./checkpoint.sh
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=9521
```

Then run the oardel -c #jobid command…

```
jdoe@idpot:~/oar-2.0/tests/checkpoint$ oardel -c 9521
Checkpointing the job 9521 ...DONE.
The job 9521 was notified to checkpoint itself (send SIGUSR2).
```

And then watch the job's output:

```
jdoe@idpot:~/oar-2.0/tests/checkpoint$ cat OAR.9521.stdout
Hostname: idpot9
Pid: 1242
Starting job at: Mon May  7 16:39:04 CEST 2007
Caught checkpoint signal at: Mon May  7 16:39:24 CEST 2007
Terminating.
```

The job terminated as expected.

1.2.11 Testing the mechanism of dependency on an anterior job termination

First Job

We run a first interactive job in a first Shell

```
jdoe@idpot:~$ oarsub -I
[ADMISSION RULE] Set default walltime to 7200.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=9458
Interactive mode : waiting...
[2007-05-04 17:59:38] Starting...

Initialize X11 forwarding...
Connect to OAR job 9458 via the node idpot9.grenoble.grid5000.fr
jdoe@idpot9:~$
```

And leave that job pending.

Second Job

Then we run a second job in another Shell, with a dependence on the first one

```
jdoe@idpot:~$ oarsub -I -a 9458
[ADMISSION RULE] Set default walltime to 7200.
[ADMISSION RULE] Modify resource description with type constraints
OAR_JOB_ID=9459
Interactive mode : waiting...
[2007-05-04 17:59:55] Start prediction: 2007-05-04 19:59:39 (Karma = 4.469)
```

So this second job is waiting for the first job walltime (or sooner termination) to be reached to start.

Job dependency in action

We do a logout on the first interactive job. . .

```
jdoe@idpot9:~$ logout
Connection to idpot9.grenoble.grid5000.fr closed.
Disconnected from OAR job 9458
jdoe@idpot:~$
```

. . . then watch the second Shell and see the second job starting

```
[2007-05-04 18:05:05] Starting...

Initialize X11 forwarding...
Connect to OAR job 9459 via the node idpot7.grenoble.grid5000.fr
```

. . . as expected.

1.3 User commands

All user commands are installed on cluster login nodes. So you must connect to one of these computers first.

1.3.1 oarsub

The user can submit a job with this command. So, what is a job in our context?

A job is defined by needed resources and a script/program to run. So, the user must specify how many resources and what kind of them are needed by his application. Thus, OAR system will give him or not what he wants and will control the execution. When a job is launched, OAR executes user program only on the first reservation node. So this program can access some environment variables to know its environment:

```
$OAR_NODEFILE                  contains the name of a file which lists
                               all reserved nodes for this job
$OAR_JOB_ID                    contains the OAR job identificator
$OAR_RESOURCE_PROPERTIES_FILE contains the name of a file which lists
                               all resources and their properties
$OAR_JOB_NAME                  name of the job given by the "-n" option
$OAR_PROJECT_NAME              job project name
```

See the manual page of the command for its syntax.

Wanted resources have to be described in a hierarchical manner using the “-l” syntax option.

Moreover it is possible to give a specification that must be matched on properties.

So the long and complete syntax is of the form:

```
"{ sql1 }/prop1=1/prop2=3+{sql2}/prop3=2/prop4=1/prop5=1+...,walltime=1:00:00"
```

where:

- *sql1* : SQL WHERE clause on the table of resources that filters resource names used in the hierarchical description
- *prop1* : first type of resources
- *prop2* : second type of resources
- + : add another resource hierarchy to the previous one
- *sql2* : SQL WHERE clause to apply on the second hierarchy request
- ...

So we want to reserve 3 resources with the same value of the type *prop2* and with the same property *prop1* and these resources must fit *sql1*. To that possible resources we want to add 2 others which fit *sql2* and the hierarchy */prop3=2/prop4=1/prop5=1*.

Examples

```
# oarsub -l /nodes=4 test.sh
```

(the “test.sh” script will be run on 4 entire nodes in the default queue with the default walltime)

```
# oarsub --stdout='test12.%jobid%.stdout' --stderr='test12.%jobid%.stderr' -l
  /nodes=4 test.sh
  ...
  OAR_JOB_ID=702
  ...
```

(same example than above but here the standard output of “test.sh” will be written in the file “test12.702.stdout” and the standard error in “test12.702.stderr”)

```
# oarsub -q default -l /nodes=10/cpu=3,walltime=2:15:00 \
  -p "switch = 'sw1'" /home/users/toto/prog
```

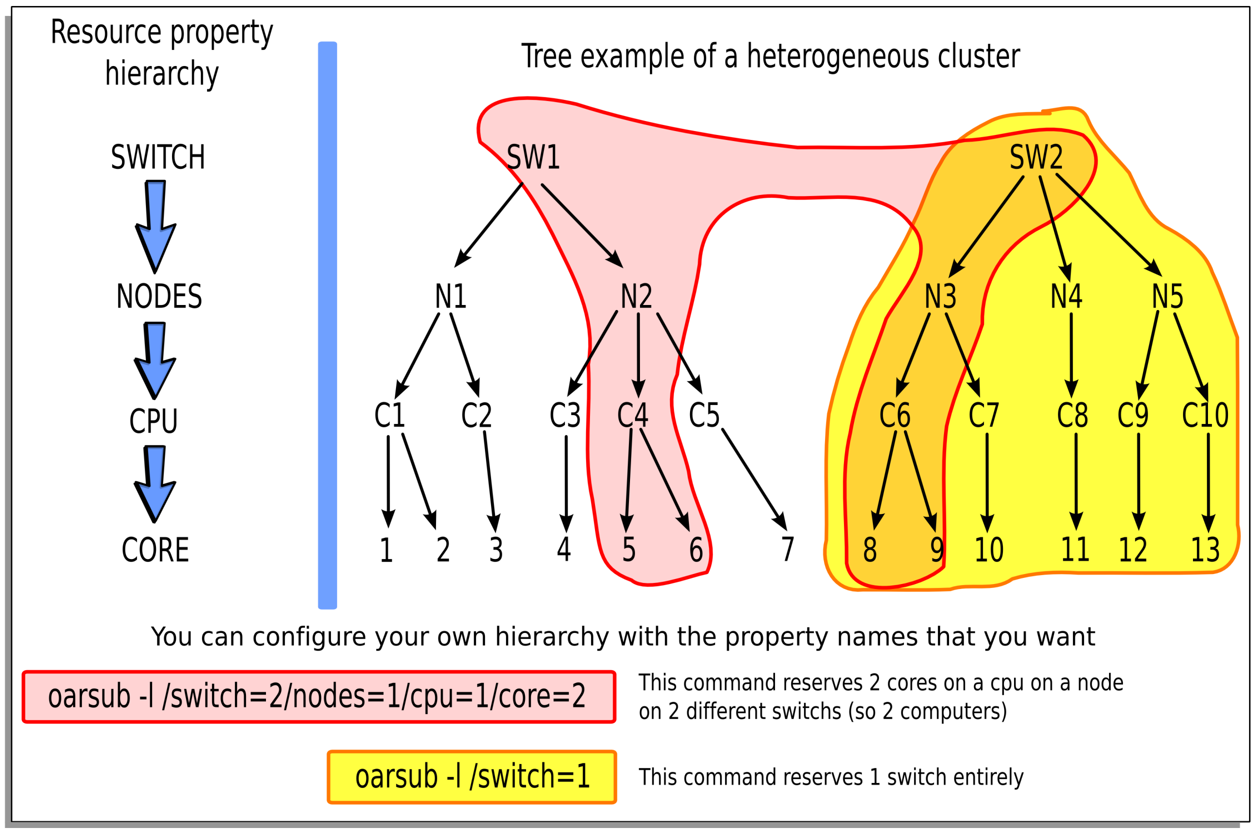

Resource property hierarchy
SWITCH
NODES
CPU
CORE
Tree example of a heterogeneous cluster
SW1
SW2
N1
N2
N3
N4
N5
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
1
2
3
4
5
6
7
8
9
10
11
12
13
You can configure your own hierarchy with the property names that you want
oarsub -l /switch=2/nodes=1/cpu=1/core=2
This command reserves 2 cores on a cpu on a node on 2 different switchs (so 2 computers)
oarsub -l /switch=1
This command reserves 1 switch entirely

Fig. 1: Example of a resource hierarchy and 2 different oarsub commands

(the "/home/users/toto/prog" script will be run on 10 nodes with 3 cpus (so a total of 30 cpus) in the default queue with a walltime of 2:15:00. Moreover "-p" option restricts resources only on the switch 'sw1')

```
# oarsub -r "2009-04-27 11:00:00" -l /nodes=12/cpu=2
```

(a reservation will begin at "2009-04-27 11:00:00" on 12 nodes with 2 cpus on each one)

```
#  oarsub -C 42
```

(connects to the job 42 on the first node and set all OAR environment variables)

```
#  oarsub -p "not host like 'nodename.%'"
```

(To exclude a node from the request)

```
# oarsub -I
```

(gives a shell on a resource)

1.3.2 oardel

This command is used to delete or checkpoint job(s). They are designed by their identifier.

Examples

```
# oardel 14 42
```

(delete jobs 14 and 42)

```
# oardel -c 42
```

(send checkpoint signal to the job 42)

1.3.3 oarsh & oarcp

Use oarsh to connect to a node from the job submission frontend of the cluster or any other node.

Use oarcp to copy files from a node or to a node.

Examples

```
oarsh node-23
```

Connect from within our job, from one node to another one (node23).

```
OAR_JOB_ID=4242 oarsh node-23
```

Connect to a node (node23) of our job (Id: 4242) from the frontal of the cluster.

```
OAR_JOB_KEY_FILE=~/my_key oarsh node-23
```

Connect to a node (node23) of our job that was submitted using a job-key.

```
oarsh -i ~/my_key node-23
```

Same thing but using OpenSSH-like *-i* option.

1.3.4 oarwalltime

This command manages requests to change the walltime of a job.

Walltime changes can only be requested for a running job.

There is no warranty that walltime can be increased, since it depends on the resources availability (next jobs).

Once a request is registered, it will by handled during the next pass of scheduling and granted if it fits with other jobs.

As per configuration:

- the walltime change functionality may be disabled in your installation, and if not there is a maximum to the possible walltime increase
- a walltime increase request may only be applied some time before the predicted end of the job. That apply time may be computed as a percentage of the walltime of the job
- a walltime increase may happen incrementally, so that other scheduled jobs get more priority. That increment may be computed as a percentage of the walltime of the job
- the functionality may be configured differently from one queue to another.

Read your site's documentation or ask your administrator to know the configured settings.

See the manual page of the command for its usage.

1.3.5 oarstat

oarstat prints jobs information.

Examples

```
oarstat
```

All current jobs

```
oarstat -u
```

Current jobs of user

```
oarstat -j 42 -f
```

Detailed information for job 42

1.3.6 oarnodes

This command prints informations about cluster resources (state, which jobs on which resources, resource properties, . . .).

Examples

```
# oarnodes
# oarnodes -s
# oarnodes --sql "state = 'Suspected'"
```

1.3.7 oarprint

Pretty printer for a job resources.

Examples

From the job head node (where $OAR_RESOURCE_PROPERTIES_FILE is defined):

```
oarprint host -P host,cpu,core -F "host: % cpu: % core: %" -C+
```

On the submission frontend:

```
oarstat -j 42 -p | oarprint core -P host,cpuset,mem -F "%[%] (%)" -f -
```

1.3.8 oarhold

This command is used to remove a job from the scheduling queue if it is in the "Waiting" state.

Moreover if its state is "Running" *oarhold* can suspend the execution and enable other jobs to use its resources.

1.3.9 oarresume

This command resumes jobs in the states *Hold* or *Suspended*

For more details, read the manual pages of the commands.

1.4 Mechanisms

1.4.1 How does an interactive *oarsub* work?

1.4.2 Job launch

For PASSIVE jobs, the mechanism is similar to the *interactive* one, except for the shell launched from the frontal node.

The job is finished when the user command ends. Then oarexec return its exit value (what errors occured) on the Almighty via the SERVER_PORT if DETACH_JOB_FROM_SERVER was set to 1 otherwise it returns directly.

1.4.3 CPUSET

The cpuset name is effectively created on each nodes and is composed as `user_jobid`.

OAR system steps:

1. Before each job, the Runner initialize the CPUSET (see *CPUSET definition*) with OPENSSH_CMD and an efficient launching tool : Taktuk. If it is not installed and configured (TAKTUK_CMD) then OAR uses an internal launching tool less optimized. The processors assigned to this cpuset are taken from the defined database field by JOB_RESOURCE_MANAGER_PROPERTY_DB_FIELD in the table resources.
2. After each job, OAR deletes all processes stored in the associated CPUSET. Thus all nodes are clean after a OAR job.

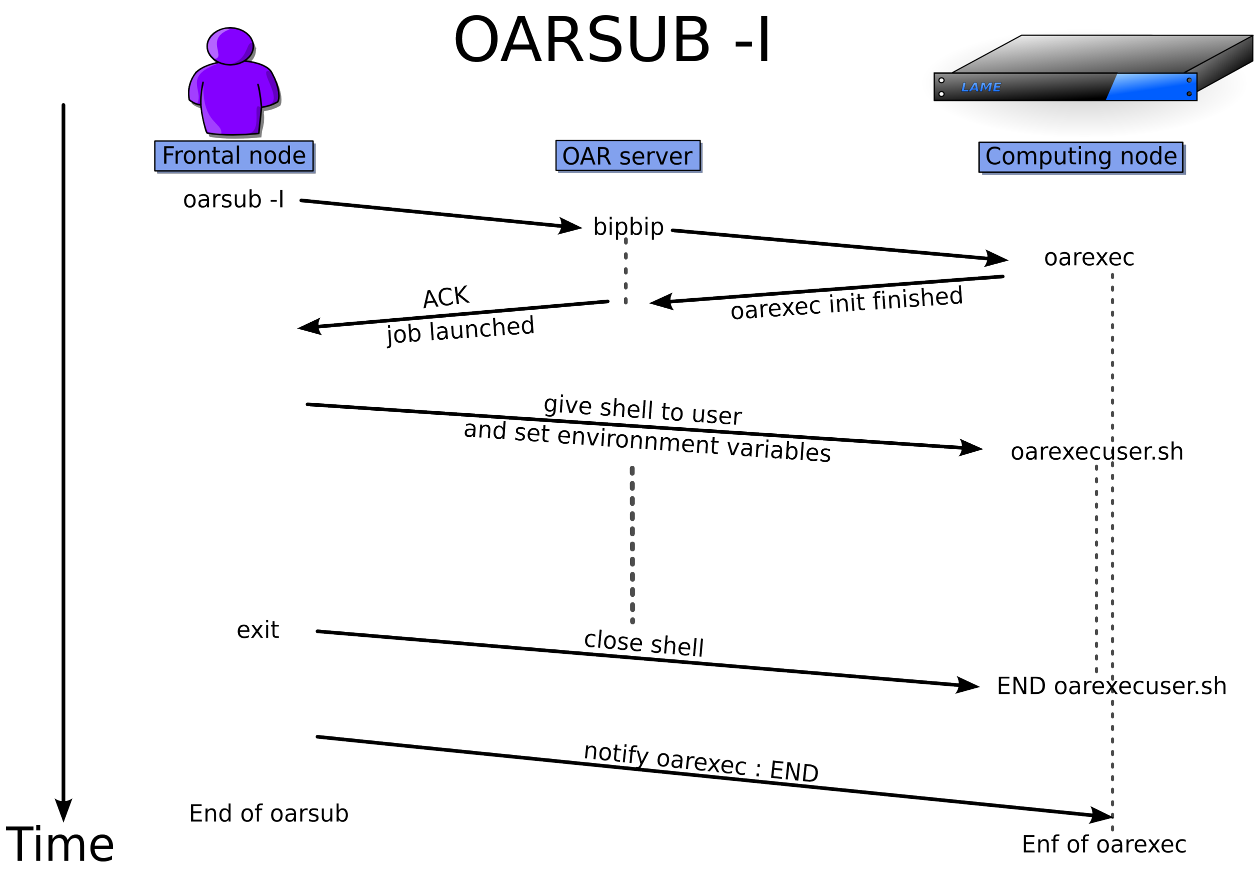


Fig. 2: Interactive oarsub decomposition

If you don't want to use this feature, you can, but nothing will warranty that every user processes will be killed after the end of a job.

If you want you can implement your own cpuset management. This is done by editing 3 files (see also *CPUSET installation*):

- cpuset_manager.pl : this script creates the cpuset on each nodes and also delete it at the end of the job. For more informations, you have to look at this script (there are several comments).
- oarsh : (OARSH) this script is used to replace the standard `ssh` command. It gets the cpuset name where it is running and transfer this information via `ssh` and the `SendEnv` option. In this file, you have to change the `get_current_cpuset` function.
- oarsh_shell : (OARSH_SHELL) this script is the shell of the oar user on each nodes. It gets environment variables and look at if there is a cpuset name. So if there is one it assigns the current process and its father to this cpusetname. So all further user processes will remind in the cpuset. In this file you just have to change the `add_process_to_cpuset` function.

1.4.4 SSH connection key definition

This function is performed by *oarsub* with the –ssh_private_key and –ssh_public_key options.

It enables the user to define a ssh key pair to connect on their nodes. So oarsh can be used on nodes of different clusters to connect each others if the same ssh keys are used with each *oarsub*.

So a grid reservation (`-r` option of *oarsub* on each OAR batch scheduler of each wanted clusters) can be done with this functionality.

Example:

```
ssh-keygen -f oar_key
oarsub --ssh_private_key ``$(cat oar_key)`` --ssh_public_key ``$(cat oar_key.pub)`` ./
↪script.sh
```

1.4.5 Suspend/resume

Jobs can be suspended with the command *oarhold* (send a `SIGSTOP` on every processes on every nodes) to allow other jobs to be executed.

`Suspended` jobs can be resumed with the command *oarresume* (send a `SIGSTOP` on every suspended processes on every nodes). They will pass into `Running` when assigned resources will be free.

IMPORTANT: This feature is available only if *CPUSET* is configured.

You can specify 2 scripts if you have to perform any actions just after (JUST_AFTER_SUSPEND_EXEC_FILE) suspend and just before resume (JUST_BEFORE_RESUME_EXEC_FILE).

Moreover you can perform other actions (than send signals to processes) if you want: just edit the `suspend_resume_manager.pl` file.

1.4.6 Job deletion

Leon tries to connect to OAR Perl script running on the first job node (find it thanks to the file `/tmp/oar/pid_of_oarexec_for_jobId_id`) and sends a `SIGTERM` signal. Then the script catch it and normally end the job (kill processes that it has launched).

If this method didn't succeed then Leon will flush the OAR database for the job and nodes will be `Suspected` by NodeChangeState.

If your job is check pointed and is of the type *idempotent* (*oarsub* `-t` option) and its exit code is equal to 99 then another job is automatically created and scheduled with same behaviours.

1.4.7 Checkpoint

The checkpoint is just a signal sent to the program specified with the *oarsub* command.

If the user uses `--checkpoint` option then Sarko will ask the OAR Perl script running on the first node to send the signal to the process (`SIGUSR2` or the one specified with `--signal`).

You can also use *oardel* command to send the signal.

1.4.8 Scheduling

General steps used to schedule a job:

1. All previous scheduled jobs are stored in a Gantt data structure.
2. All resources that match property constraints of the job(`-p` option and indication in the `{...}` from the `-l` option of the *oarsub*) are stored in a tree data structure according to the hierarchy given with the `-l` option.
3. Then this tree is given to the Gantt library to find the first hole where the job can be launched.
4. The scheduler stores its decision into the database in the gantt_jobs_predictions and gantt_jobs_resources tables.

See User section from the FAQ for more examples and features.

1.4.9 Job dependencies

A job dependency is a situation where a job needs the ending of another job to start. OAR deals with job dependency problems by refusing to schedule dependant jobs if their required job is in Terminated state and have an exit code != 0 (an error occured). If the required job is resubmited, its jobId is no longer the same and OAR updates the database and sets the job_id_required field to this new jobId for the dependant job.

> **Note** The queues configured with the quota features (*oar_sched_gantt_with_timesharing_and_fairsharing_and_quotas*) have a different behaviour. This scheduler always launches dependant jobs even if there required jobs are in *Error* state or with an exit code != 0.

1.4.10 User notification

This section explains how the `--notify` *oarsub* option is handled by OAR:

- The user wants to receive an email: The syntax is `mail:name@domain.com`. Mail section in the *Configuration file* must be present otherwise the mail cannot be sent. The subject of the mail is of the form:

  OAR [*TAG*]: job_id (job_name) on OAR_server_hostname

- The user wants to launch a script: The syntax is `exec:/path/to/script args`. OAR server will connect (using OPENSSH_CMD) on the node where the *oarsub* command was invoked and then launches the script with the following arguments : *job_id*, *job_name*, *TAG*, *comments*.

***TAG* can be:**

- RUNNING : when the job is launched

- END : when the job is finished normally
- ERROR : when the job is finished abnormally
- INFO : used when oardel is called on the job
- SUSPENDED : when the job is suspended
- RESUMING : when the job is resumed

1.4.11 Accounting aggregator

In the *Configuration file* you can set the ACCOUNTING_WINDOW parameter. Thus the command oaraccounting will split the time with this amount and feed the table accounting.

So this is very easily and faster to get usage statistics of the cluster. We can see that like a `data warehousing` information extraction method.

1.4.12 Dynamic nodes coupling features

We are working with the Icatis company on clusters composed by Intranet computers. These nodes can be switch in computing mode only at specific times. So we have implemented a functionality that can request to power on some hardware if they can be in the cluster.

We are using the field *available_upto* from the table resources to know when a node will be inaccessible in the cluster mode (easily settable with oarnodesetting command). So when the OAR scheduler wants some potential available computers to launch the jobs then it executes the command SCHEDULER_NODE_MANAGER_WAKE_UP_CMD.

Moreover if a node didn't execute a job for SCHEDULER_NODE_MANAGER_IDLE_TIME seconds and no job is scheduled on it before SCHEDULER_NODE_MANAGER_SLEEP_TIME seconds then OAR will launch the command SCHEDULER_NODE_MANAGER_SLEEP_CMD.

1.4.13 Timesharing

It is possible to share the slot time of a job with other ones. To perform this feature you have to specify the type *timesharing* when you use *oarsub*.

1.4.14 Container jobs

With this functionality it is possible to execute jobs within another one. So it is like a sub-scheduling mechanism.

First a job of the type *container* must be submitted, for example:

```
oarsub -I -t container -l nodes=10,walltime=2:10:00
...
OAR_JOB_ID=42
...
```

Then it is possible to use the *inner* type to schedule the new jobs within the previously created container job:

```
oarsub -I -t inner=42 -l nodes=7
oarsub -I -t inner=42 -l nodes=1
oarsub -I -t inner=42 -l nodes=10
```

Notes:

- In the case:

  ```
  oarsub -I -t inner=42 -l nodes=11
  ```

  This job will never be scheduled because the container job `42` reserved only 10 nodes.
- `-t container` is handled by every kind of jobs (passive, interactive and reservations). But `-t inner=...` cannot be used with a reservation.

1.4.15 Besteffort jobs

Besteffort jobs are scheduled in the besteffort queue. Their particularity is that they are deleted if another not besteffort job wants resources where they are running.

For example you can use this feature to maximize the use of your cluster with multiparametric jobs. This what it is done by the CIGRI project.

When you submit a job you have to use `-t besteffort` option of *oarsub* to specify that this is a besteffort job.

Important : a besteffort job cannot be a reservation.

If your job is of the type *besteffort* and *idempotent* (*oarsub* `-t` option) and killed by the OAR scheduler then another job is automatically created and scheduled with same behaviours.

1.4.16 Cosystem jobs

This feature enables to reserve some resources without launching any program on corresponding nodes. Thus nothing is done by OAR on computing nodes when a job is starting except on the COSYSTEM_HOSTNAME defined in the configuration file.

This is useful with an other launching system that will declare its time slot in OAR. So yo can have two different batch scheduler.

When you submit a job you have to use `-t cosystem` option of *oarsub* to specify that this is a cosystem job.

These jobs are stopped by the *oardel* command or when they reach their walltime or their command has finished. They also use the node COSYSTEM_HOSTNAME to launch the specified program or shell.

1.4.17 Deploy jobs

This feature is useful when you want to enable the users to reinstall their reserved nodes. So the OAR jobs will not log on the first computer of the reservation but on the DEPLOY_HOSTNAME.

So prologue and epilogue scripts are executed on DEPLOY_HOSTNAME and if the user wants to launch a script it is also executed on DEPLOY_HOSTNAME.

OAR does nothing on computing nodes because they normally will be rebooted to install a new system image.

This feature is strongly used in the Grid5000 project with Kadeploy tools.

When you submit a job you have to use `-t deploy` option of *oarsub* to specify that this is a deploy job.

1.4.18 Quotas

The administrator can limit the number of resources used by user, job types, project ans queue (or a combination of them). This feature acts like quotas. When one of the defined rules is reached then next jobs will not be scheduled at this time. The scheduler will find another slot when the quotas will be satisfied.

This feature is available in queues which use the scheduler `oar_sched_gantt_with_timesharing_and_fairsharing_and`

The quota rules are defined in `/etc/oar/scheduler_quotas.conf`.

By default no quota is applied.

Note1: Quotas are applied globally, only the jobs of the type `container` are not taken in account (but the inner jobs are used to compute the quotas).

Note2: Besteffort jobs are not taken in account except in the besteffort queue.

1.4.19 Desktop computing

If you cannot contact the computers via SSH you can install the `desktop computing` OAR mode. This kind of installation is based on two programs:

- oar-cgi : this is a web CGI used by the nodes to communicate with the OAR server via a HTTP server on the OAR server node.
- oar-agent.pl : This program asks periodically the server web CGI to know what it has to do.

This method replaces the SSH command. Computers which want to register them into OAR just has to be able to contact OAR HTTP server.

In this situation we don't have a NFS file system to share the same directories over all nodes so we have to use a stagein/stageout solution. In this case you can use the *oarsub* option `stagein` to migrate your data.

1.5 Desktop computing

If you want to compute jobs on nodes without SSH connections then this feature is for you.

On the nodes you have to run "oar-agent.pl". This script polls the OAR server via a CGI HTTP script.

Usage examples:

- if you want to run a program that you know is installed on nodes:

  ```
  oarsub -t desktop_computing /path/to/program
  ```

  Then /path/to/program is run and the files created in the oar-agent.pl running directory is retrieved where oarsub was launched.

- if you want to copy a working environment and then launch the program:

  ```
  oarsub -t desktop_computing -s . ./script.sh
  ```

  The content of "." is transfred to the node, "./script.sh" is run and everything will go back.

1.6 User REST API

The OAR REST API allows to interact with OAR over http using a REST library. Most of the operations usually done with the oar Unix commands may be done using this API from your favourite language.

1.6.1 Concepts

Access

A simple GET query to the API using wget may look like this:

```
# Get the list of resources
wget -O - http://www.mydomain.org/oarapi/resources.yaml?structure=simple
```

You can also access to the API using a browser. Make it point to http://www.myoarcluster.local/oarapi/index.html and you'll see a very simple HTML interface allowing you to browse the cluster resources, post a job using a form or even create resources if you are a OAR administrator. (of course, replace www.myoarcluster.local by a valid name allowing you to join the http service of the host where the API is installed).

But generally, you'll use a REST client or a REST library provided for your favorite language. You'll see examples using a ruby rest library in the next parts of this document.

Check your system administrator to know on which URI the OAR API is installed.

Authentication

Most of the time, you'll make requests that needs you to be authenticated. The way you are authenticated depends on what your local admistrator configured. There's almost as many possibilities as what Apache (the http server used by this API) may manage. The simplest method is a "Basic authentication" with a login/password. It may be binded to a local directory (for example LDAP). You may also find an "ident" based authentication that guesses automatically your login from a little daemon running on your client host. If the "ident" method is used, your unix login is automatically used. But as only a few hosts may be trusted, you'll probably have to open a tunnel to one of this host. You may use ssh to do this. For example, supposing access.mycluster.fr is a gateway host trusted by the api host:

```
$ ssh -NL 8080:api.mycluster.fr:80 login@access.mycluster.fr

Then, point your REST client to::

# http://localhost:8080
```

Formats and data structure types

The API currently can serve data into *YAML*, *JSON* or *HTML*. Posted data can also be coded into *YAML*, *JSON* or *x-www-form-urlencoded* (for HTML from posts). You may specify the requested format by 2 ways:

- giving an extension to resources: **.yaml**, **.json** or **.html**
- setting the **HTTP_ACCEPT** header variable to **text/yaml**, **application/json** or **text/html**

For the posted data, you have to correctly set the **HTTP_CONTENT_TYPE** variable to **text/yaml**, **application/json** or **application/x-www-form-urlencoded**.

Sometimes, the data structures returned (not the coding format, but the contents: array, hashes, array of hashes,. . .) may be changed. Currently, we have 2 available data structure types: *simple* and *oar*. The structure is passed through the variable *structure* that you may pass in the url, for example: ?structure=simple

- The **simple** data structure tries to be as simple as possible, using simple arrays in place of hashes wherever it is possible
- The **oar** data structure serves data in the way oar does with the oarnodes/oarstat export options (-Y, -D, -J,. . .) Be aware that this data structure is not meant to be maintained since 2.5 release of OAR. The simple data structure is highly recommended.

By default, we use the *simple* data structure.

Here are some examples, using the ruby restclient (see next section):

```
# Getting resources infos
  # in JSON
irb(main):004:0> puts get('/resources.json')
  # in YAML
irb(main):005:0> puts get('/resources.yaml')
  # Same thing
irb(main):050:0> puts get('/resources', :accept=>"text/yaml")
  # Specifying the "oar" data structure
irb(main):050:0> puts get('/resources.json?structure=oar')
  # Specifying the "simple" data structure
irb(main):050:0> puts get('/resources.json?structure=simple')
```

Errors and debug

When the API returns an error, it generally uses a standard HTTP return status (404 NOT FOUND, 406 NOT ACCEPTABLE, …). But it also returns a body containing a hash like the following:

```
{
 "title" : "ERROR 406 - Invalid content type required */*",
 "message" : "Valid types are text/yaml, application/json or text/html",
 "code" : "200"
}
```

This error body is formated in the requested format. But if this format was not given, it uses JSON by default.

To allow you to see the error body, you may find it useful to activate the **debug=1** variable. It will force the API to always return a 200 OK status, even if there's an error so that you can see the body with a simple browser or a rest client without having to manage the errors. For example:

```
wget -nv -O - "http://localhost:8080/oargridapi/sites/grenoble?debug=1"
```

Here is an example of error catching in ruby:

```
# Function to get objects from the api
# We use the JSON format
def get(api,uri)
  begin
    return JSON.parse(api[uri].get(:accept => 'application/json'))
  rescue => e
    if e.respond_to?('http_code')
      puts "ERROR #{e.http_code}:\n #{e.response.body}"
    else
      puts "Parse error:"
      puts e.inspect
    end
    exit 1
  end
end
```

1.6.2 Ruby REST client

One of the easiest way for testing this API is to use the rest-client ruby module:

http://rest-client.heroku.com/rdoc/

It may be used from ruby scripts (http://www.ruby.org/) or interactively. It is available as a rubygem, so to install it, simply install rubygems and do "gem install rest-client". Then, you can run the interactive client which is nothing else than irb with shortcuts. Here is an example irb session:

```
$ export PATH=$PATH:/var/lib/gems/1.8/bin
$ restclient http://localhost/oarapi
irb(main):001:0> puts get('/jobs.yaml')
---
- api_timestamp: 1246457384
  id: 551
  name: ~
  owner: bzizou
  queue: besteffort
  state: Waiting
  submission: 1245858042
  uri: /jobs/551
=> nil
irb(main):002:0>
```

or, if an http basic auth is required:

```
restclient http://localhost/api <login> <password>
...
```

1.6.3 Pagination and common rules into output data

Results served by the API are mainly of 2 kinds: "items" and "collections". A collection is actually an array of items. Some uris serve collections that may have a very big amount of items (for example all the terminated jobs of a cluster). For that reason, collections are often "paginated". It means that the collections are presented into pages that have got meta data to give you informations about offset, numbers, and links to previous/next page. Furthermore, items are often composed of commonly used kind of data, especially 'id' and 'links'. We have tried to normalize this as much as possible, so, here is a description of the common properties of items and collections:

Items

Items have the following features:

Hash Items should be hashes (sometimes hash of hashes for the 'oar' data structure, but it is to be avoided)

the 'id' key In general, when an item may be uniquely identified by an integer, it is given in the "id" key. Note that OAR nodes are identified by the 'network_address' key that is an integer, but this is an exception.

the 'links' array Items, especially when listed in a collection, often give links to more informations or relative data. The links are listed in the links array. Each element of this array (a link) is composed of at least: a 'rel' key and a 'href' key. The 'rel' key is a string telling what is the relation between the current item and the resource pointed by the link. The 'href' key is a string giving the URI of the link relative to the root of the API. It's possible that other keys will be implemented in the future (for example a 'title' key.) Common values for 'rel' are: 'self', 'parent', 'next', 'previous'.

the 'api_timestamp' value Each item has a 'api_timestamp' key giving the epoch unix date at which the API constructed the item. This field may be omitted when items are listed inside a collection; then the collection has a global api_timestamp value. This date is given in the timezone provided by the "GET /timezone uri".

Collections

Collections have the following features:

the 'items' array The items array is the purpose of a collection. It lists all the items of the current page of a collection.

the 'total' number It's an integer giving the total number of items in the collection. If the items array contains less elements than this number, then the collection has been paginated and a 'next' and/or 'previous' link should be provided.

the 'offset' number It gives the offset at which the paginated list starts. If 0, then, it is the first page.

the 'limit' parameter This is not in the output, but a parameter common to all paginable uris. If you specify a limit, then it gives the size of the pages.

the 'links' array For a collection, the links array often gives the uri of the next/previous page. But it also gives the uri of the current page ('self') and may point to more informations relative to this collection. See the links array description from above for items, it is similar for the collection.

Examples

An item looks like this (yaml output):

```
api_timestamp: 1286894740
available_upto: 2147483646
besteffort: YES
core: 1
cpu: 1
cpuset: 0
deploy: NO
desktop_computing: NO
expiry_date: 0
finaud_decision: NO
id: 1
last_available_upto: 0
last_job_date: 1286885902
links:
  - href: /resources/nodes/fake1
    rel: node
  - href: /resources/1
    rel: self
  - href: /resources/1/jobs
    rel: jobs
network_address: fake1
next_finaud_decision: NO
next_state: UnChanged
resource_id: 1
scheduler_priority: 0
state: Alive
state_num: 1
suspended_jobs: NO
type: default
```

A collection looks like this (yaml output):

```
api_timestamp: 1286894823
items:
  - api_timestamp: 1286894823
    id: 2
    links:
      - href: /jobs/2
        rel: self
      - href: /jobs/2/resources
        rel: resources
    name: ~
    owner: kameleon
    queue: default
    state: Error
    submission: 1284034267
  - api_timestamp: 1286894823
    id: 3
    links:
      - href: /jobs/3
        rel: self
      - href: /jobs/3/resources
        rel: resources
    name: ~
    owner: kameleon
    queue: default
    state: Error
    submission: 1284034383
links:
  - href: /jobs.yaml?state=Error&limit=2&offset=0
    rel: self
  - href: /jobs.yaml?state=Error&limit=2&offset=2
    rel: next
offset: 0
total: 2623
```

1.6.4 REST requests description

Examples are given in the YAML format because we think that it is the more human readable and so very suitable for this kind of documentation. But you can also use the JSON format for your input/output data. Each resource uri may be postfixed by .yaml, .jso of .html.

In this section, we describe every REST resources of the OAR API. The authentication may be:

- public: everybody can query this resource
- user: only authenticated and valid users can query this resource
- oar: only the oar user can query this resource (administration usage)

GET /index

description Home page for the HTML browsing

formats html

authentication public

output

example:

```
<HTML>
<HEAD>
<TITLE>OAR REST API</TITLE>
</HEAD>
<BODY>
<HR>
<A HREF=./resources.html>RESOURCES</A>&nbsp;&nbsp;&nbsp;
<A HREF=./jobs.html>JOBS</A>&nbsp;&nbsp;&nbsp;
<A HREF=./jobs/form.html>SUBMISSION</A>&nbsp;&nbsp;&nbsp;
<HR>
Welcome on the oar API
```

note Header of the HTML resources may be customized into the **/etc/oar/api_html_header.pl** file.

GET /version

description Gives version informations about OAR and OAR API. Also gives the timezone of the API server.

formats html , yaml , json

authentication public

output

structure: hash

yaml example:

```
---
api: 0.1.2
api_timestamp: 1245582255
api_timezone: CEST
apilib: 0.1.6
oar: 2.4.0
```

usage example

```
wget -q -O - http://localhost/oarapi/version.yaml
```

GET /whoami

description Gives the name of authenticated user seen by OAR API. The name for a not authenticated user is the null string.

formats html , yaml , json

authentication public

output

structure: hash

yaml example:

```
---
api_timestamp: 1245582255
authenticated_user: kameleon
```

usage example

```
wget -q -O - http://localhost/oarapi/whoami.yaml
```

GET /timezone

description Gives the timezone of the OAR API server. The api_timestamp given in each query is an UTC timestamp (epoch unix time). This timezone information allows you to re-construct the local time.

formats html , yaml , json

authentication public

output *structure*: hash

yaml example:

```
---
api_timestamp: 1245768107
timezone: CEST
```

usage example

```
wget -q -O - http://localhost/oarapi/timezone.yaml
```

GET /jobs

description List jobs (by default only the jobs in queue)

formats html , yaml , json

authentication public

parameters

- **state**: comma separated list of states for filtering the jobs. Possible values: Terminated, Running, Error, Waiting, Launching, Hold,. . .
- **array** (integer): to get the jobs belonging to an array
- **from** (timestamp): restrict the list to the jobs that are running or not yet started before this date. Using this parameters disables the default behavior of listing only the jobs that are in queue.
- **to** (timestamp): restrict the list to the jobs that are running or not yet finished at this date. Using this parameters disables the default behavior of listing only the jobs that are in queue.
- **user**: restrict the list to the jobs owned by this username
- **ids**: colon separated list of ids to get a set of jobs

output *structure*: collection

yaml example:

```
api_timestamp: 1286895857
items:
  - api_timestamp: 1286895857
    id: 58
    links:
      - href: /jobs/58
        rel: self
      - href: /jobs/58/resources
        rel: collection
        title: resources
      - href: /oarapi/jobs/58/nodes
        rel: collection
        title: nodes
    name: ~
    owner: kameleon
    queue: default
    state: Terminated
    submission: 1284109267
  - api_timestamp: 1286895857
    id: 59
    links:
      - href: /jobs/59
        rel: self
      - href: /jobs/59/resources
        rel: collection
        title: resources
      - href: /oarapi/jobs/59/nodes
        rel: collection
        title: nodes
    name: ~
    owner: kameleon
    queue: default
    state: Terminated
    submission: 1284109846
links:
  - href: /jobs.yaml?state=Terminated&limit=2&offset=48
    rel: self
  - href: /jobs.yaml?state=Terminated&limit=2&offset=50
    rel: next
  - href: /jobs.yaml?state=Terminated&limit=2&offset=46
    rel: previous
offset: 48
total: 206
```

note The “rel: resources” link of a job lists the assigned or reserved resources of a job.

usage example

```
wget -q -O - http://localhost/oarapi/jobs.yaml?state=Terminated,Running&
↪limit=2&offset=48"
```

GET /jobs/details

description Same as /jobs, but with more details and “resources” and “nodes” links developped.

formats html , yaml , json

authentication public

parameters

- **state**: comma separated list of states for filtering the jobs. Possible values: Terminated, Running, Error, Waiting, Launching, Hold,...

output *structure*: collection

yaml example:

```
api_timestamp: 1352707511
items:
  - api_timestamp: 1352707511
    array_id: 5540
    array_index: ~
    command: sleep 300
    cpuset_name: kameleon_5540
    dependencies: []
    events: []
    exit_code: ~
    id: 5540
    initial_request: oarsub sleep 300
    launching_directory: /home/kameleon
    links:
      - href: /oarapi/jobs/5540
        rel: self
      - href: /oarapi/jobs/5540/resources
        rel: collection
        title: resources
      - href: /oarapi/jobs/5540/nodes
        rel: collection
        title: nodes
    message: Karma = 0.000
    name: ~
    nodes:
      - api_timestamp: 1352707511
        links:
          - href: /oarapi/resources/nodes/node1
            rel: self
        network_address: node1
        status: assigned
    owner: kameleon
    project: default
    properties: desktop_computing = 'NO'
    queue: default
    reservation: None
    resources:
      - api_timestamp: 1352707511
        id: 1
        links:
          - href: /oarapi/resources/1
            rel: self
          - href: /oarapi/resources/1/jobs
            rel: collection
            title: jobs
        status: assigned
    resubmit_job_id: ~
    scheduled_start: 1352707488
    start_time: 1352707488
    state: Running
```

(continues on next page)

(continued from previous page)

```
    stderr_file: OAR.5540.stdout
    stdout_file: OAR.5540.stderr
    stop_time: 0
    submission_time: 1352707487
    type: PASSIVE
    types: []
    walltime: 7200
    wanted_resources: "-l \"{type = 'default'}/resource_id=1,
↪walltime=2:0:0\" "
  - api_timestamp: 1352707511
    array_id: 5542
    array_index: ~
    command: sleep 300
    cpuset_name: kameleon_5542
    dependencies: []
    events: []
    exit_code: ~
    id: 5542
    initial_request: oarsub -l /core=2 sleep 300
    launching_directory: /home/kameleon
    links:
      - href: /oarapi/jobs/5542
        rel: self
      - href: /oarapi/jobs/5542/resources
        rel: collection
        title: resources
      - href: /oarapi/jobs/5542/nodes
        rel: collection
        title: nodes
    message: Karma = 0.000
    name: ~
    nodes:
      - api_timestamp: 1352707511
        links:
          - href: /oarapi/resources/nodes/node1
            rel: self
        network_address: node1
        status: assigned
    owner: kameleon
    project: default
    properties: desktop_computing = 'NO'
    queue: default
    reservation: None
    resources:
      - api_timestamp: 1352707511
        id: 3
        links:
          - href: /oarapi/resources/3
            rel: self
          - href: /oarapi/resources/3/jobs
            rel: collection
            title: jobs
        status: assigned
      - api_timestamp: 1352707511
        id: 4
        links:
          - href: /oarapi/resources/4
```

(continues on next page)

(continued from previous page)

```
            rel: self
          - href: /oarapi/resources/4/jobs
            rel: collection
            title: jobs
        status: assigned
    resubmit_job_id: ~
    scheduled_start: 1352707510
    start_time: 1352707510
    state: Running
    stderr_file: OAR.5542.stdout
    stdout_file: OAR.5542.stderr
    stop_time: 0
    submission_time: 1352707509
    type: PASSIVE
    types: []
    walltime: 7200
    wanted_resources: "-l \"{type = 'default'}/core=2,walltime=2:0:0\
↪" "
links:
  - href: /oarapi/jobs/details.yaml?offset=0
    rel: self
offset: 0
total: 2
```

usage example:

```
wget -q -O - http://localhost/oarapi/jobs/details.yaml
```

GET /jobs/table

description Same as /jobs but outputs the data of the SQL job table

formats html , yaml , json

authentication public

parameters

- **state**: comma separated list of states for filtering the jobs. Possible values: Terminated, Running, Error, Waiting, Launching, Hold,…

output *structure*: collection

yaml example:

```
---
items:
 - accounted: NO
   api_timestamp: 1253017554
   array_id: 566
   assigned_moldable_job: 566
   checkpoint: 0
   checkpoint_signal: 12
   command: ''
   exit_code: ~
   file_id: ~
   info_type: bart:33033
```

(continues on next page)

(continued from previous page)

```
  initial_request: oarsub -I
  job_env: ~
  job_group: ''
  job_id: 566
  job_name: ~
  job_type: INTERACTIVE
  job_user: bzizou
  launching_directory: /home/bzizou/git/oar/git
  message: FIFO scheduling OK
  notify: ~
  project: default
  properties: desktop_computing = 'NO'
  queue_name: default
  reservation: None
  resubmit_job_id: 0
  scheduler_info: FIFO scheduling OK
  start_time: 1253017553
  state: Launching
  stderr_file: OAR.%jobid%.stderr
  stdout_file: OAR.%jobid%.stdout
  stop_time: 0
  submission_time: 1253017551
  suspended: NO
  uri: /jobs/566
- accounted: NO
  api_timestamp: 1253017554
  array_id: 560
  assigned_moldable_job: 0
  checkpoint: 0
  checkpoint_signal: 12
  command: /usr/bin/id
  exit_code: ~
  file_id: ~
  info_type: 'bart:'
  initial_request: oarsub --resource=/nodes=2/cpu=1 --use_job_key=1 /
↪usr/bin/id
  job_env: ~
  job_group: ''
  job_id: 560
  job_name: ~
  job_type: PASSIVE
  job_user: bzizou
  launching_directory: /home/bzizou
  message: Cannot find enough resources which fit for the job 560
  notify: ~
  project: default
  properties: desktop_computing = 'NO'
  queue_name: default
  reservation: None
  resubmit_job_id: 0
  scheduler_info: Cannot find enough resources which fit for the job␣
↪560
  start_time: 0
  state: Waiting
  stderr_file: OAR.%jobid%.stderr
  stdout_file: OAR.%jobid%.stdout
  stop_time: 0
```

(continues on next page)

(continued from previous page)

```
    submission_time: 1246948570
    suspended: NO
    uri: /jobs/560
links:
 - href: '/jobs/table.html?state=Terminated&limit=15&offset=0'
   rel: previous
 - href: '/jobs/table.html?state=Terminated&limit=15&offset=15'
   rel: self
 - href: '/jobs/table.html?state=Terminated&limit=15&offset=30'
   rel: next
offset: 15
total: 41
```

note: Field names may vary from the other job lists because this query results more like a dump of the jobs table.

usage example

```
wget -q -O - http://localhost/oarapi/jobs/table.yaml
```

GET /jobs/<id>[/details]

description Get infos about the given job. If /details is appended, it gives more informations, such as the expanded list of resources allocated to the job.

parameters

- **id**: the id of a job

formats html , yaml , json

authentication user

output *structure*: hash

yaml example:

```
api_timestamp: 1352707658
array_id: 5230
array_index: 3
command: /home/kameleon/cigri-3/tmp/test1.sh param48 48
cpuset_name: kameleon_5232
dependencies: []
events:
  - date: 1351087783
    description: Scheduler priority for job 5232 updated (network_
↪address/resource_id)
    event_id: 14454
    job_id: 5232
    to_check: NO
    type: SCHEDULER_PRIORITY_UPDATED_STOP
  - date: 1351087782
    description: '[bipbip 5232] Ask to change the job state'
    event_id: 14451
    job_id: 5232
    to_check: NO
    type: SWITCH_INTO_TERMINATE_STATE
```

(continues on next page)

(continued from previous page)

```
  - date: 1351087660
    description: Scheduler priority for job 5232 updated (network_
↪address/resource_id)
    event_id: 14446
    job_id: 5232
    to_check: NO
    type: SCHEDULER_PRIORITY_UPDATED_START
exit_code: 0
id: 5232
initial_request: oarsub --resource=core=1 --type=besteffort /home/
↪kameleon/cigri-3/tmp/test1.sh --array-param-file=/tmp/oarapi.
↪paramfile.7QPM0
launching_directory: /home/kameleon
links:
  - href: /oarapi/jobs/5232
    rel: self
  - href: /oarapi/jobs/5232/resources
    rel: collection
    title: resources
  - href: /oarapi/jobs/5232/nodes
    rel: collection
    title: nodes
message: Karma = 0.000
name: ~
owner: kameleon
project: default
properties: (besteffort = 'YES') AND desktop_computing = 'NO'
queue: besteffort
reservation: None
resubmit_job_id: 0
scheduled_start: ~
start_time: 1351087660
state: Terminated
stderr_file: OAR.5232.stderr
stdout_file: OAR.5232.stdout
stop_time: 1351087782
submission_time: 1351087659
type: PASSIVE
types:
  - besteffort
walltime: 7200
wanted_resources: "-l \"{type = 'default'}/core=1,walltime=2:0:0\" "
```

usage example

```
wget --user test --password test -q -O - http://localhost/oarapi/jobs/
↪547.yaml
```

GET /jobs/<id>/resources

description Get resources reserved or assigned to a job

parameters

- **id**: the id of a job

formats html , yaml , json

authentication public

output *structure*: hash

***yaml example*:**

```
api_timestamp: 1352707730
items:
  - api_timestamp: 1352707730
    id: 7
    links:
      - href: /oarapi/resources/7
        rel: self
      - href: /oarapi/resources/7/jobs
        rel: collection
        title: jobs
    status: assigned
links:
  - href: /oarapi/jobs/5232/resources.yaml
    rel: self
offset: 0
total: 1
```

usage example

```
wget -q -O - http://localhost/oarapi/jobs/547/resources.yaml
```

POST /jobs/<id>/deletions/new

description Deletes a job

parameters

- **id**: the id of a job

formats html , yaml , json

authentication user

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1253025331
cmd_output: |
  Deleting the job = 567 ...REGISTERED.
  The job(s) [ 567 ] will be deleted in a near future.
id: 567
status: Delete request registered
```

usage example

```
irb(main):148:0> puts post('/jobs/567/deletions/new.yaml','')
```

POST /jobs/<id>/checkpoints/new

description Send the checkpoint signal to a job

parameters

- **id**: the id of a job

formats html , yaml , json

authentication user

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1253025555
cmd_output: |
  Checkpointing the job 568 ...DONE.
  The job 568 was notified to checkpoint itself.
id: 568
status: Checkpoint request registered
```

usage example

```
irb(main):148:0> puts post('/jobs/568/checkpoints/new.yaml','')
```

POST /jobs/<id>/holds/new

description Asks to hold a waiting job

parameters

- **id**: the id of a job

formats html , yaml , json

authentication user

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1253025718
cmd_output: "[560] Hold request was sent to the OAR server.\n"
id: 560
status: Hold request registered
```

usage example

```
irb(main):148:0> puts post('/jobs/560/holds/new.yaml','')
```

POST /jobs/<id>/rholds/new

description Asks to hold a running job

parameters

- **id**: the id of a job

formats html , yaml , json

authentication oar

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1253025868
cmd_output: "[569] Hold request was sent to the OAR server.\n"
id: 569
status: Hold request registered
```

usage example

```
irb(main):148:0> puts post('/jobs/560/rholds/new.yaml','')
```

POST /jobs/<id>/resumptions/new

description Asks to resume a holded job

parameters

- **id**: the id of a job

formats html , yaml , json

authentication user

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1253026081
cmd_output: "[569] Resume request was sent to the OAR server.\n"
id: 569
status: Resume request registered
```

usage example

```
irb(main):148:0> puts post('/jobs/560/resumptions/new.yaml','')
```

POST /jobs/<id>/signals/<signal>

description Asks to resume a holded job

parameters

- **id**: the id of a job
- **signal**: the number of a signal (see kill -l)

formats html , yaml , json

authentication user

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1253102493
cmd_output: |
  Signaling the job 574 with 12 signal.
  DONE.
  The job 574 was notified to signal itself with 12.
id: 574
status: Signal sending request registered
```

usage example

```
irb(main):148:0> puts post('/jobs/560/signals/12.yaml','')
```

POST /jobs

description Creates (submit) a new job

formats html , yaml , json

authentication user

input Only [resource] and [command] are mandatory. Please, refer to the documentation of the *oarsub* command for the resource syntax which correspond to the -l (–resource) option.

structure: hash with possible arrays (for options that may be passed multiple times)

***fields*:**

- **resource** (*string*): the resources description as required by oar (example: "/nodes=1/cpu=2")
- **command** (*string*): a command name or a script that is executed when the job starts
- **workdir** (*string*): the path of the directory from where the job will be submited
- **param_file** (*string*): the content of a parameters file, for the submission of an array job. For example: {"resource":"/nodes=1, "command":"sleep", "param_file":"60n90n30"}
- **All other option accepted by the oarsub unix command**: every long option that may be passed to the oarsub command is known as a key of the input hash. If the option is a toggle (no value), you just have to set it to "1" (for example: 'use-job-key' => '1'). Some options may be arrays (for example if you want to specify several 'types' for a job)

***yaml example*:**

```
---
stdout: /tmp/outfile
command: /usr/bin/id;echo "OK"
resource: /nodes=2/cpu=1
workdir: ~bzizou/tmp
type:
- besteffort
- timesharing
use-job-key: 1
```

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1332323792
cmd_output: |
  [ADMISSION RULE] Modify resource description with type constraints
  OAR_JOB_ID=4
id: 4
links:
  - href: /oarapi-priv/jobs/4
    rel: self
```

note: more informations about the submited job may be obtained with a GET on the provided *uri*.

usage example

```
# Submitting a job using ruby rest client
irb(main):010:0> require 'json'
irb(main):012:0> j={ 'resource' => '/nodes=2/cpu=1', 'command' => '/usr/
↪bin/id' }
irb(main):015:0> job=post('/jobs' , j.to_json , :content_type =>
↪'application/json')

# Submitting a job with a provided inline script
irb(main):024:0> script="#!/bin/bash
irb(main):025:0" echo \"Hello world\"
irb(main):026:0" whoami
irb(main):027:0" sleep 300
irb(main):028:0" "
irb(main):029:0> j={ 'resource' => '/nodes=2/cpu=1', 'script' => script ,
↪ 'workdir' => '~bzizou/tmp'}
irb(main):030:0> job=post('/jobs' , j.to_json , :content_type =>
↪'application/json')
```

POST /jobs/<id>

description Updates a job. In fact, as some clients (www browsers) doesn't support the DELETE method, this POST resource has been created mainly to workaround this and provide another way to delete a job. It also provides *checkpoint*, *hold* and *resume* methods, but one should preferably use the /checkpoints, /holds and /resumptions resources.

formats html , yaml , json

authentication user

input *structure*: hash {"action" => "delete"}

***yaml example*:**

```
---
method: delete
```

output *structure*: hash

***yaml example*:**

```
---
api_timestamp: 1245944206
cmd_output: |
  Deleting the job = 554 ...REGISTERED.
```

(continues on next page)

(continued from previous page)

```
  The job(s) [ 554 ] will be deleted in a near future.
id: 554
status: Delete request registered
```

usage example

```
# Deleting a job in the ruby rest client
puts post('/jobs/554.yaml','{"method":"delete"}',:content_type =>
↪"application/json")
```

DELETE /jobs/<id>

description Delete or kill a job.

formats html , yaml , json

authentication user

output *structure*: hash returning the status

yaml example:

```
---
api_timestamp: 1245944206
cmd_output: |
  Deleting the job = 554 ...REGISTERED.
  The job(s) [ 554 ] will be deleted in a near future.
id: 554
status: Delete request registered
```

usage example

```
# Deleting a job in the ruby rest client
puts delete('/jobs/554.yaml')
```

note Not all clients support the DELETE method, especially some www browsers. So, you can do the same thing with a POST of a {"method":"delete"} hash on the /jobs/<id> resource.

GET /jobs/form

description HTML form for posting (submiting) new jobs from a browser

formats html

authentication user

output

example:

```
<HTML>
 <HEAD>
 <TITLE>OAR REST API</TITLE>
 </HEAD>
 <BODY>
 <HR>
 <A HREF=../resources.html>RESOURCES</A>&nbsp;&nbsp;&nbsp;
```

(continues on next page)

(continued from previous page)

```
<A HREF=../jobs.html>JOBS</A>&nbsp;&nbsp;&nbsp;
<A HREF=../jobs/form.html>SUBMISSION</A>&nbsp;&nbsp;&nbsp;
<HR>

<FORM METHOD=post ACTION=../jobs.html>
<TABLE>
<CAPTION>Job submission</CAPTION>
<TR>
  <TD>Resources</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=resource VALUE="/nodes=1/cpu=1,
↪walltime=00:30:00"></TD>
</TR><TR>
  <TD>Name</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=name VALUE="Test_job"></TD>
</TR><TR>
  <TD>Properties</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=property VALUE=""></TD>
</TR><TR>
  <TD>Program to run</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=command VALUE='"/bin/sleep 300"'>
↪</TD>
</TR><TR>
  <TD>Types</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=type></TD>
</TR><TR>
  <TD>Reservation dates</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=reservation></TD>
</TR><TR>
  <TD>Directory</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=directory></TD>
</TR><TR>
  <TD></TD><TD><INPUT TYPE=submit VALUE=SUBMIT></TD>
</TR>
</TABLE>
</FORM>
```

note This form may be customized in the **/etc/oar/api_html_postform.pl** file

GET /resources

description Get the list of resources and their state

formats html , yaml , json

authentication public

output *structure*: hash

***fields*:**

- **items** : list of resources
- **links** : links to previous, current and next resources
- **offset** : current offset
- **total** : resources total

***yaml example*:**

```
---
items:
 - api_timestamp: 1253201950
   jobs_uri: /resources/4/jobs
   network_address: liza-1
   node_uri: /resources/nodes/liza-1
   resource_id: 4
   state: Alive
   uri: /resources/4
 - api_timestamp: 1253201950
   jobs_uri: /resources/5/jobs
   network_address: liza-1
   node_uri: /resources/nodes/liza-1
   resource_id: 5
   state: Alive
   uri: /resources/5
 - api_timestamp: 1253201950
   jobs_uri: /resources/6/jobs
   network_address: liza-2
   node_uri: /resources/nodes/liza-2
   resource_id: 6
   state: Alive
   uri: /resources/6
 - api_timestamp: 1253201950
   jobs_uri: /resources/7/jobs
   network_address: liza-2
   node_uri: /resources/nodes/liza-2
   resource_id: 7
   state: Alive
   uri: /resources/7
links:
 - href: '/resources.yaml?limit=5&offset=2'
   rel: previous
 - href: '/resources.yaml?limit=5&offset=7'
   rel: self
 - href: '/resources.yaml?limit=5&offset=12'
   rel: next
offset: 2
total: 49
```

note: More details about a resource can be obtained with a GET on the provided *uri*. The list of all the resources of the same node may be obtained with a GET on *node_uri*. The list of running jobs on a resource can be obtained with a GET on the jobs_uri resource. *note*: The following parameters can be passed through the requested URL

- limit : limit of resources to be shown per page
- offset : the page result offset

usage example

```
wget -q -O - http://localhost/oarapi/resources.yaml
```

GET /resources/details

description Get the list of resources and all the details about them

formats html , yaml , json

authentication public

output *structure*: hash

***fields*:**

- **items** : list of resources
- **links** : links to previous, current and next resources
- **offset** : current offset
- **total** : total of resources

***yaml example*:**

```
---
    items:
           - api_timestamp: 1281967035
     available_upto: 0
     besteffort: YES
     core: ~
     cpu: 0
     cpufreq: ~
     cpuset: 0
     cputype: ~
     deploy: NO
     desktop_computing: NO
     expiry_date: 0
     finaud_decision: NO
     jobs_uri: '/resources/1/jobs.html'
     last_available_upto: 0
     last_job_date: 1278588052
     memnode: ~
     network_address: node1
         next_finaud_decision: NO
         next_state: UnChanged
         node_uri: '/resources/nodes/node1.html'
         resource_id: 1
         scheduler_priority: 0
         state: Suspected
         state_num: 3
         suspended_jobs: NO
         type: default
         uri: '/resources/1.html'
           - api_timestamp: 1281967035
         available_upto: 0
         besteffort: YES
         core: ~
         cpu: 0
         cpufreq: ~
         cpuset: 0
         cputype: ~
         deploy: NO
         desktop_computing: NO
         expiry_date: 0
         finaud_decision: NO
         jobs_uri: '/resources/2/jobs.html'
         last_available_upto: 0
         last_job_date: 1278588052
         memnode: ~
```

(continues on next page)

(continued from previous page)

```
          network_address: node1
          next_finaud_decision: NO
          next_state: UnChanged
          node_uri: '/resources/nodes/node1.html'
          resource_id: 2
          scheduler_priority: 0
          state: Suspected
          state_num: 3
          suspended_jobs: NO
          type: default
          uri: '/resources/2.html'
            - api_timestamp: 1281967035
          available_upto: 0
          besteffort: YES
          core: ~
          cpu: 1
          cpufreq: ~
          cpuset: 0
          cputype: ~
          deploy: NO
          desktop_computing: NO
          expiry_date: 0
          finaud_decision: NO
          jobs_uri: '/resources/3/jobs.html'
          last_available_upto: 0
          last_job_date: 1278588052
          memnode: ~
          network_address: node1
          next_finaud_decision: NO
          next_state: UnChanged
          node_uri: '/resources/nodes/node1.html'
          resource_id: 3
          scheduler_priority: 0
          state: Suspected
          state_num: 3
          suspended_jobs: NO
          type: default
          uri: '/resources/3.html'
      links:
        - href: '/resources/details.yaml?limit=5&offset=2'
          rel: previous
        - href: '/resources/details.yaml?limit=5&offset=7'
          rel: self
        - href: '/resources/details.yaml?limit=5&offset=12'
          rel: next
offset: 2
    total: 49
```

usage example

```
wget -q -O - http://localhost/oarapi/resources/details.yaml

*note*: The following parameters can be passed through the requested URL
       - limit : limit of resources to be shown per page
       - offset : the page result offset
```

GET /resources/<id>

description Get details about the resource identified by *id*

formats html , yaml , json

authentication public

output *structure*: 1 element array of hash

***yaml example*:**

```
---
api_timestamp: 1253202322
available_upto: 0
besteffort: YES
cluster: 0
cpu: 20
cpuset: 0
deploy: NO
desktop_computing: NO
expiry_date: 0
finaud_decision: NO
jobs_uri: /resources/1/jobs
last_available_upto: 0
last_job_date: 1253201845
licence: ~
network_address: bart-1
next_finaud_decision: NO
next_state: UnChanged
node_uri: /resources/nodes/bart-1
resource_id: 1
scheduler_priority: 0
state: Alive
state_num: 1
suspended_jobs: NO
test: ~
type: default
uri: /resources/1
```

usage example

```
wget -q -O - http://localhost/oarapi/resources/1.yaml
```

GET /resources/nodes/<network_address>

description Get details about the resources belonging to the node identified by *network_address*

formats html , yaml , json

authentication public

output *structure*: array of hashes

***yaml example*:**

```
---
- api_timestamp: 1253202379
  jobs_uri: /resources/4/jobs
```

(continues on next page)

(continued from previous page)

```
  network_address: liza-1
  node_uri: /resources/nodes/liza-1
  resource_id: 4
  state: Alive
  uri: /resources/4
- api_timestamp: 1253202379
  jobs_uri: /resources/5/jobs
  network_address: liza-1
  node_uri: /resources/nodes/liza-1
  resource_id: 5
  state: Alive
  uri: /resources/5
```

usage example

```
wget -q -O - http://localhost/oarapi/resources/nodes/liza-1.yaml
```

POST /resources/generate

description Generates (outputs) a set of resources using oaradmin. The result may then be directly sent to /resources for actual creation.

formats html , yaml , json

authentication oar

input [resources] and [properties] entries are mandatory

structure: hash describing the resources to generate

fields:

- **resources** (*string*): A string corresponding to the resources definition as it could have been passed to the "oaradmin resources -a" command (see man oaradmin).
- **properties** (*hash*): an optional hash defining some common properties for these new resources

yaml example:

```
---
ressources: /nodes=node{2}.test.generate/cpu={2}/core={2}
properties:
  memnode: 1050
  cpufreq: 5
```

output *structure*: an array of hashes containing the generated resources that may be pushed to POST /resources for actual creation

yaml example:

```
---
api_timestamp: 1321366378
items:
  - core: 1
    cpu: 1
    cpuset: 0
    network_address: node1.test.generate
```

(continues on next page)

(continued from previous page)

```
  - core: 2
    cpu: 1
    cpuset: 1
    network_address: node1.test.generate
  - core: 3
    cpu: 2
    cpuset: 2
    network_address: node1.test.generate
  - core: 4
    cpu: 2
    cpuset: 3
    network_address: node1.test.generate
  - core: 5
    cpu: 3
    cpuset: 0
    network_address: node2.test.generate
  - core: 6
    cpu: 3
    cpuset: 1
    network_address: node2.test.generate
  - core: 7
    cpu: 4
    cpuset: 2
    network_address: node2.test.generate
  - core: 8
    cpu: 4
    cpuset: 3
    network_address: node2.test.generate
links:
  - href: /oarapi-priv/resources/generate.yaml
    rel: self
offset: 0
total: 8
```

usage example

```
# Generating new resources with curl
curl -i -X POST http://oar:kameleon@localhost/oarapi-priv/resources/
↪generate -H'Content-Type: application/json' -d '{"resources":"/
↪nodes=node{2}.test.generate/cpu={2}/core={2}"}'
```

POST /resources

description Creates a new resource or a set of new resources

formats html , yaml , json

authentication oar

input A [hostname] or [network_address] entry is mandatory

structure: A hash describing the resource to be created. An array of hashes may be given for creating a set of new resources. The result of a /resources/generate query may be directly posted to /resources.

***fields*:**

- **hostname** alias **network_address** (*string*): the network address given to the resource

- **<properties>** : The hash may be appended with any other valid property

yaml example:

```
---
hostname: test2
besteffort: "NO"
cpu: "10"
```

output *structure*: hash returning the id of the newly created resource and status (or an array of hashes if a set of resources has been given on the input)

yaml example:

```
---
api_timestamp: 1245946199
id: 32
status: ok
uri: /resources/32
warnings: []
```

usage example

```
# Adding a new resource with the ruby rest client (oar user only)
irb(main):078:0> r={ 'hostname'=>'test2', 'properties'=> { 'besteffort'=>
↪'NO' , 'cpu' => '10' } }
irb(main):078:0> puts post('/resources', r.to_json , :content_type =>
↪'application/json')
```

POST /resources/<id>/state

description Change the state

formats html , yaml , json

authentication oar

input A [state] entry is mandatory and must be "Absent", "Alive" or "Dead"

structure: hash of state

fields:

- **state**: Alive, Absent or Dead

yaml example:

```
---
state: Dead
```

output *structure*:

yaml example:

```
---
api_timestamp: 1253283492
id: 34
status: Change state request registered
uri: /resources/34
```

usage example

```
irb
```

DELETE /resources/<id>

description Delete the resource identified by *id*

formats html , yaml , json

authentication oar

output *structure*: hash returning the status

***yaml example*:**

```
---
api_timestamp: 1245946801
status: deleted
```

usage example

```
# Deleting a resource with the ruby rest client
puts delete('/resources/32.yaml')
```

note If the resource could not be deleted, returns a 403 and the reason into the message body.

DELETE /resources/<node>/<cpuset_id>

description Delete the resource corresponding to *cpuset_id* on node *node*. It is useful when you don't know about the ids, but only the number of cpus on physical nodes.

formats html , yaml , json

authentication oar

output *structure*: hash returning the status

***yaml example*:**

```
---
api_timestamp: 1246459253
status: deleted
=> nil
```

usage example

```
# Deleting a resource with the ruby rest client
puts delete('/resources/test/0.yaml')
```

note If the resource could not be deleted, returns a 403 and the reason into the message body.

GET /admission_rules

description Get the list of admission rules

formats html , yaml , json

authentication oar

output *structure*: hash

fields:

- **items** : list of admission rules
- **links** : links to previous, current and next admission rules
- **offset** : current offset
- **total** : total of admission rules

yaml example:

```
---
items:
  - id: 1
    links:
       href: /admission_rules/1
       rel: self
    rule: 'if (not defined($queue_name)) {$queue_name="default";}'
  - id: 2
    links:
       href: /admission_rules/2
       rel: self
    rule: 'die ("[ADMISSION RULE] root and oar users are not allowed␣
↪to submit jobs.\n") if ( $user eq "root" or $user eq "oar" );'
  - id: 3
    links:
       href: /admission_rules/3
       rel: self
    rule: |2
           my $admin_group = "admin";
           if ($queue_name eq "admin") {
                   my $members;
                   (undef,undef,undef, $members) = getgrnam($admin_
↪group);
                   my %h = map { $_ => 1 } split(/\s+/,$members);
                   if ( $h{$user} ne 1 ) {
                   {die("[ADMISSION RULE] Only member of the group ".
↪$admin_group." can submit jobs in the admin queue\n");}
                   }
           }
links:
  - href: '/admission_rules.yaml?limit=5&offset=0'
    rel: previous
  - href: '/admission_rules.yaml?limit=5&offset=5'
    rel: self
  - href: '/admission_rules.yaml?limit=5&offset=10'
    rel: next
offset: 5
total: 5
```

usage example

```
wget -q -O - http://localhost/oarapi/admission_rules.yaml

*note*: The following parameters can be passed through the requested URL
       - limit : limit of admission rules to be shown per page
       - offset : the page result offset
```

GET /admission_rules/<id>

description Get details about the admission rule identified by *id*

formats html , yaml , json

authentication oar

output *structure*: 1 element array of hash

***yaml example*:**

```
---
- id: 1
  links:
     href: /admission_rules/1
     rel: self
  rule: 'if (not defined($queue_name)) {$queue_name="default";}'
```

usage example

```
wget -q -O - http://localhost/oarapi/admission_rules/1.yaml
```

DELETE /admission_rule/<id>

description Delete the admission rule identified by *id*

formats html , yaml , json

authentication oar

output *structure*: hash returning the status

***yaml example*:**

```
---
id: 32
api_timestamp: 1245946801
status: deleted
```

usage example

```
# Deleting an admisssion rule with the ruby rest client
puts delete('/admission_rules/32.yaml')
```

note

note Not all clients support the DELETE method, especially some www browsers. So, you can do the same thing with a POST of a {"method":"delete"} hash on the /admission_rule/<id> rule. If the admission rule could not be deleted, returns a 403 and the reason into the message body.

POST /admission_rules

description Add a new admission rule

formats html , yaml , json

authentication oar

input *structure*: hash

fields:

- **rule** (*text*): The admission rule to add

yaml example:

```
---
rule: |
  echo "This is a test rule"
```

output A 201 (created) header is returned if the rule is successfully created, with a location value.

yaml example:

```
---
api_timestamp: 1340180126
id: 19
rule: echo "This is a test rule"
uri: /oarapi-priv/admission_rules/19
```

POST /admission_rules/<id>

description Update or delete the admission rule given by *id*

formats html , yaml , json

authentication oar

input *structure*: hash

fields:

- **rule** (*text*): The content of the admission rule to update
- **method=delete** : If given, the admission rule is deleted

yaml example:

```
---
rule: |
  echo "This is a test rule"
```

output A 201 (created) header is returned if the rule is successfully updated, with a location value.

yaml example:

```
---
api_timestamp: 1340180126
id: 19
rule: echo"test rule"
uri: /oarapi-priv/admission_rules/19
```

GET /config

description Get the list of configured variables

formats html , yaml , json

authentication oar

output *structure*: array of hashes

yaml example:

```
---
- id: DB_BASE_NAME
  links:
     href: /config/DB_BASE_NAME
     rel: self
  value: oar
- id: OARSUB_FORCE_JOB_KEY
  links:
     href: /config/OARSUB_FORCE_JOB_KEY
     rel: self
  value: no
- id: SCHEDULER_GANTT_HOLE_MINIMUM_TIME
  links:
     href: /config/SCHEDULER_GANTT_HOLE_MINIMUM_TIME
     rel: self
  value: 300
- id: SCHEDULER_RESOURCE_ORDER
  links:
     href: /config/SCHEDULER_RESOURCE_ORDER
     rel: self
  value: 'scheduler_priority ASC, suspended_jobs ASC, network_address␣
↪DESC, resource_id ASC'
- id: SCHEDULER_PRIORITY_HIERARCHY_ORDER
  links:
     href: /config/SCHEDULER_PRIORITY_HIERARCHY_ORDER
     rel: self
  value: network_address/resource_id
- id: OARSUB_NODES_RESOURCES
  links:
     href: /config/OARSUB_NODES_RESOURCES
     rel: self
  value: network_address
- id: SCHEDULER_JOB_SECURITY_TIME
  links:
     href: /config/SCHEDULER_JOB_SECURITY_TIME
     rel: self
     value: 60
- id: DETACH_JOB_FROM_SERVER
  links:
     href: /config/DETACH_JOB_FROM_SERVER
     rel: self
  value: 0
- id: LOG_LEVEL
  links:
     href: /config/LOG_LEVEL
     rel: self
  value: 2
- id: OAREXEC_DEBUG_MODE
  links:
     href: /config/OAREXEC_DEBUG_MODE
     rel: self
  value: 0

    .....
    .....
```

usage example

```
curl -i -X GET http://login:password@localhost/oarapi-priv/config.yaml
```

GET /config/file

description Get the raw config file of OAR. It also output the path of the file used by the API.

formats html , yaml , json

authentication oar

output *structure*: hash

fields:

- **path** : The path of the config file
- **file** : The raw content of the config file (text)

usage example

```
curl -i -X GET http://kameleon:kameleon@localhost/oarapi-priv/config/
↪file.yaml
```

GET /config/<variable>

description Get details about the configuration variable identified by *variable*

formats html , yaml , json

authentication oar

output *structure*: 1 element array of hash

yaml example:

```
---
- id: DB_TYPE
  links:
     href: /config/DB_TYPE
     rel: self
  value: mysql
```

usage example

```
curl -i -X GET http://login:password@localhost/oarapi-priv/config/DB_
↪TYPE.yaml
```

POST /config/<variable>

description Change the value of the configuration variable identified by *variable*

formats html , yaml , json

authentication oar

input A [value] entry is mandatory

structure: hash describing the new value of the variable

fields:

- **value** (*string*): the value of the given variable

yaml example:

```
---
value: 'state=Finishing,Running,Resuming,Suspended,Launching,toLaunch,
↪Waiting,toAckReservation,Hold,Terminated'
```

output *structure*: hash returning the variable and his new value

yaml example:

```
---
    API_JOBS_URI_DEFAULT_PARAMS:
      value: 'state=Finishing,Running,Resuming,Suspended,Launching,
↪toLaunch,Waiting,toAckReservation,Hold,Terminated'
```

usage example

```
curl -i -X POST http://login:password@localhost/oarapi-priv/config/API_
↪JOBS_URI_DEFAULT_PARAMS.yaml -H'Content-Type: text/yaml' -T config.yaml
```

note config.yaml contains the value of the variable.

GET /media/ls/<file_path>

description Get a list of the directory from the path given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request.

formats html , yaml , json

authentication user

output *structure*: array of hashes giving for each listed file: the name, the mode, the size, the modification time and the type (*f* for a file or *d* for a directory)

yaml example:

```
---
api_timestamp: 1340095354
items:
  - mode: 33188
    mtime: 1339685040
    name: API.pm
    size: 58620
    type: f
  - mode: 16877
    mtime: 1340094660
    name: bart
    size: ~
    type: d
  - mode: 16877
    mtime: 1338993000
    name: cigri-3
    size: ~
    type: d
  - mode: 16877
```

(continues on next page)

(continued from previous page)

```
    mtime: 1340095200
    name: oar
    size: ~
    type: d
  - mode: 16877
    mtime: 1334132940
    name: oar_install
    size: ~
    type: d
  - mode: 33261
    mtime: 1339685040
    name: oarapi.pl
    size: 75939
    type: f
  - mode: 33261
    mtime: 1340027400
    name: test.sh
    size: 43
    type: f
links:
  - href: /oarapi-priv/media/ls/~/
    rel: self
offset: 0
total: 7
```

usage example

```
curl -i -X GET http://kameleon:kameleon@localhost/oarapi-priv/media/ls/~/
↪  -H'Content-Type: text/yaml'
```

note returns a 404 if the path does not exist, or a 403 if the path is not readable. Errors in debug mode (with ?debug=1) are formated into yaml.

GET /media/<file_path>

description Get a file located on the API host, into the path given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request.

parameters

- **tail**: specifies an optional number of lines for printing only the tail of a text file

formats application/octet-stream

authentication user

output octet-stream

usage example

```
curl -i -H'Content-Type: application/octet-stream'  http://
↪kameleon:kameleon@localhost/oarapi-priv/media/~/cigri-3/CHANGELOG
```

note returns a 404 if the file does not exist, or a 403 if the file is not readable. Errors in debug mode (with ?debug=1) are formated into yaml.

POST /media/<file_path>

description Upload or create a file on the API host, into the path given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request. If the path does not exists, the directories are automatically created. If no data is passed, an empty file is created. If binary data is sent as POSTDATA, then it is a file to upload.

formats application/octet-stream

authentication user

output 201 if ok

usage example

```
curl -i -X POST -H'Content-Type: application/octet-stream' --data-binary␣
↪@/etc/group http://kameleon:kameleon@localhost/oarapi-priv/media/~/
↪testdir/testfile
```

POST /media/chmod/<file_path>

description Changes the permissions on a file: do a chmod(1) on *file_path*. The special character "~" is expanded as the home of the user that makes the query.

formats html , yaml , json

authentication user

input A [mode] entry is mandatory

mode: A mode definition as passed to the "chmod" unix command.

output 202 if ok

usage example

```
curl -i -X POST http://kameleon:kameleon@localhost/oarapi-priv/media/
↪chmod/~/param9  -H'Content-Type: application/json' -d '{"mode":"755"}'
```

DELETE /media/<file_path>

description Delete the file or directory given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request. If the path is a directory, then it is deleted recursively.

formats application/octet-stream

authentication user

output 204 if ok

usage example

```
curl -i -X DELETE -H'Content-Type: application/octet-stream' http://
↪kameleon:kameleon@localhost/oarapi-priv/media/~/testdir
```

GET /colmet/job/<id>

description Extract colmet data for a given job. Colmet should be installed and the colmet-collector should dump data into hdf5 files located in the API_COLMET_HDF5_PATH_PREFIX specified into the oar.conf file. The served data is provided as a gzip compressed file containing a JSON hash with a key for each metric. The "hostname" and "timestamp" metrics are always appended, even if not specified.

parameters

- **from**: Optional timestamp to restrict the beginning of the time interval of data to get. If not specified, the start time of the job is used instead.
- **to**: Optional timestamp to restrict the end of the time interval of data to get. If not specified, the end of the job is used instead, or now if the job is still running.
- **metrics**: Coma separated list of metrics to get from colmet data files. The default is "ac_etime,cpu_run_real_total,coremem,read_bytes,write_bytes".

formats application/x-gzip

authentication user

output Gzip compressed JSON data

usage example

```
curl -H'Content-Type: application/x-gzip' "http://localhost/oarapi/
↪colmet/job/5767965?from=1427780621&to=1427899621" > 5767965.json.gz
```

1.6.5 Some equivalences with oar command line

OAR command	REST request
oarstat	GET /jobs.html
oarstat -Y	GET /jobs/details.yaml
oarstat -Y -j <id>	GET /jobs/<id>.yaml
oarstat -Y -fj <id>	GET /jobs/<id>/details.yaml
oardel <id>	DELETE /jobs/<id>.yaml
oardel <id> *(alternative way)*	POST /jobs/deletions/<id>/new.yaml
oarnodes -Y	GET /resources/details.yaml
oarnodes -Y -r1	GET /resources/1.yaml

1.7 FAQ - USER

1.7.1 How can I submit a moldable job?

You just have to use several "-l" *oarsub* option (one for each moldable description). By default the OAR scheduler will launch the moldable job which will end first.

So you can see some free resources but the scheduler can decide to start your job later because they will have more free resources and the job walltime will be smaller.

1.7.2 How can I submit a job with a non uniform description?

Example:

```
oarsub -I -l '{switch="sw1" or switch="sw5"}/switch=1+/node=1'
```

This example asks OAR to reserve all resources from the switch sw1 or the switch sw2 **and** a node on another switch.

You can see the "+" syntax as a sub-reservation directive.

1.7.3 Can I perform a fix scheduled reservation and then launch several jobs in it?

Yes. You have to use the OAR scheduler "timesharing" feature. To use it, the reservation and your further jobs must be of the type timesharing (only for you).

Example:

1. Make your reservation:

   ```
   oarsub -r "2006-09-12 8:00:00" -l /switch=1 -t 'timesharing=user,*'
   ```

   This command asks all resources from one switch at the given date for the default walltime. It also specifies that this job can be shared with himself and without a constraint on the job name.

2. Once your reservation has begun then you can launch:

   ```
   oarsub -I -l /node=2,walltime=0:50:00 -p 'switch="'scheduled_switch_name'\
   -t 'timesharing=user,*'
   ```

   So this job will be scheduled on nodes assigned from the previous reservation.

The "timesharing" *oarsub* command possibilities are enumerated in *Timesharing*.

1.7.4 How can a checkpointable job be resubmitted automatically?

You have to specify that your job is *idempotent* and exit from your script with the exit code 99. So, after a successful checkpoint, if the job is resubmitted then all will go right and there will have no problem (like file creation, deletion, . . .).

Example:

```
oarsub --checkpoint 600 --signal 2 -t idempotent /path/to/prog
```

So this job will send a signal *SIGINT* (see *man kill* to know signal numbers) 10 minutes before the walltime ends. Then if everything goes well and the exit code is 99 it will be resubmitted.

1.7.5 How to submit a non disturbing job for other users?

You can use the *besteffort* job type. Thus your job will be launched only if there is a hole and will be deleted if another job wants its resources.

Example:

```
oarsub -t besteffort /path/to/prog
```

CHAPTER

TWO

ADMIN DOCUMENTATION

2.1 Installation

2.1.1 Overview

There are currently 3 methods to install OAR:

- from the Debian packages
- from the RPM packages
- from sources

Before going further, please have in mind OAR's architecture. A common OAR installation is composed of:

- a **server** which will hold all of OAR "smartness". That host will run the OAR server daemon;
- one or more **frontends**, which users will have to login to, in order to reserve computing nodes (oarsub, oarstat, oarnodes, . . .);
- **computing nodes** (or basically *nodes*), where the jobs will execute;
- optionally a **visualisation server** which will host the visualisation webapps (monika, drawgantt, . . .);
- optionally an **API server**, which will host OAR restful API service.

Many OAR data are stored and archived in a database: you have the choice to use either PostgreSQL or MySQL. We recommend using **PostgreSQL**.

Beside this documentation, please have a look at **OAR website**: http://oar.imag.fr, which also provides a lot of information, espacially in the **Download** and **Contribs** sections.

2.1.2 Computing nodes

Installation from the packages

Instructions

For RedHat like systems:

```
# OAR provides a Yum repository.
# For more information see: http://oar.imag.fr/download#rpms

# Install OAR node
yum --enablerepo=OAR install oar-node
```

For the Debian like systems:

```
# OAR is shipped as part of Debian official distributions (newer versions can be␣
↪available in backports)
# For more info see: http://oar.imag.fr/download#debian

# Install OAR node
apt-get install oar-node
```

Installation from the tarball

Requirements

For RedHat like systems:

```
# Build dependencies
yum install gcc make tar python-docutils

# Common dependencies
yum install Perl Perl-base openssh
```

For Debian like system:

```
# Build dependencies
apt-get install gcc make tar python-docutils

# Common dependencies
apt-get install perl perl-base openssh-client openssh-server
```

Instructions

Get the sources:

```
OAR_VERSION=2.5.4
wget -O - http://oar-ftp.imag.fr/oar/2.5/sources/stable/oar-${OAR_VERSION}.tar.gz |␣
↪tar xzvf -
cd oar-${OAR_VERSION}/
```

build/install/setup:

```
# build
make node-build
# install
make node-install
# setup
make node-setup
```

Configuration

Init.d scripts

If you have installed OAR from sources, you need to become root user and install manually the {init.d,default,sysconfig} scripts present in the folders:

```
$PREFIX/share/doc/oar-node/examples/scripts/{init.d,default,sysconfig}
```

Then you just need to use the script `/etc/init.d/oar-node` to start the SSH daemon dedicated to oar-node.

SSH setup

OAR uses SSH to connect from machine to machine (e.g. from server or frontend to nodes or from nodes to nodes), using a dedicated SSH daemon usually running on port 6667.

Upon installtion of the OAR server on the server machine, a SSH key pair along with an authorized_keys file is created for the oar user in `/var/lib/oar/.ssh`. You need to copy that directory from the oar server to the nodes.

Please note that public key in the authorized_keys file must be prefixed with `environment="OAR_KEY=1"`, e.g.:

```
environment="OAR_KEY=1" ssh-rsa AAAAB3NzaC1yc2[...]6mIcqvcwG1K7V6CHLQKHKWo/␣
↪root@server
```

Also please make sure that the `/var/lib/oar/.ssh` directory and contained files have the right ownership (oar.oar) and permissions for SSH to function.

2.1.3 Server

Installation from the packages

Instructions

For RedHat like systems:

```
# OAR provides a Yum repository.
# For more information see: http://oar.imag.fr/download#rpms

# Install OAR server for the PostgreSQL backend
yum --enablerepo=OAR install oar-server oar-server-pgsql

# or Install OAR server for the MySQL backend
yum --enablerepo=OAR install oar-server oar-server-mysql
```

For the Debian like systems:

```
# OAR is shipped as part of Debian official distributions (newer versions can be␣
↪available in backports)
# For more info see: http://oar.imag.fr/download#debian

# Install OAR server for the PostgreSQL backend
apt-get install oar-server oar-server-pgsql

# or Install OAR server for the MySQL backend
apt-get install oar-server oar-server-mysql
```

Installation from the tarball

Requirements

For RedHat like systems:

```
# Add the epel repository (choose the right version depending on your
# operating system)
yum install epel-release

# Build dependencies
yum install gcc make tar python-docutils

# Common dependencies
yum install Perl Perl-base openssh Perl-DBI perl-Sort-Versions

# MySQL dependencies
yum install mysql-server mysql perl-DBD-MySQL

# PostgreSQL dependencies
yum install postgresql-server postgresql perl-DBD-Pg
```

For Debian like system:

```
# Build dependencies
apt-get install gcc make tar python-docutils

# Common dependencies
apt-get install perl perl-base openssh-client openssh-server libdbi-perl libsort-
↪versions-perl

# MySQL dependencies
apt-get install mysql-server mysql-client libdbd-mysql-perl

# PostgreSQL dependencies
apt-get install postgresql postgresql-client libdbd-pg-perl
```

Instructions

Get the sources:

```
OAR_VERSION=2.5.4
wget -O - http://oar-ftp.imag.fr/oar/2.5/sources/stable/oar-${OAR_VERSION}.tar.gz |␣
↪tar xzvf -
cd oar-${OAR_VERSION}/
```

Build/Install/Setup the OAR server:

```
# build
make server-build
# install
make server-install
# setup
make server-setup
```

Configuration

The oar database

Define the database configuration in /etc/oar/oar.conf. You need to set the variables `DB_TYPE`, `DB_HOSTNAME`, `DB_PORT`, `DB_BASE_NAME`, `DB_BASE_LOGIN`, `DB_BASE_PASSWD`, `DB_BASE_LOGIN_RO`, `DB_BASE_PASSWD_RO`:

```
vi /etc/oar/oar.conf
```

Create the database and the database users:

```
# General case
oar-database --create --db-admin-user <ADMIN_USER> --db-admin-pass <ADMIN_PASS>

# OR, for PostgreSQL, in case the database is installed locally
oar-database --create --db-is-local
```

Init.d scripts

If you have installed OAR from sources, you need to become root user and install manually the init.d/default/sysconfig scripts present in the folders:

```
$PREFIX/share/doc/oar-server/examples/scripts/{init.d,default,sysconfig}
```

Then use the script `/etc/init.d/oar-server` to start the OAR server daemon.

Adding resources to the system

To **automatically** initialize resources for your cluster, you can run the `oar_resources_init` command. It will detect the resources from nodes set in a file and give the OAR commands to initialize the database with the appropriate values for the memory and the cpuset properties.

Another tool is also available to create resources beforehand: that tool does not require nodes to be up and accessible by SSH. See `oar_resources_add`.

Otherwise:

To add resources to your system, you can use (as root) the `oarnodesetting` command. For a complete understanding of what that command does, see the manual page. For a basic usage, the main options are **-a** (means add a resource) and **-h** (defines the resource hostname or ip adress).

For instance, to add a computing resource for node <NODE_IP> to your setup, type:

```
oarnodesetting -a -h <NODE_IP>
```

This adds a resource with <NODE_IP> as host IP address (network_address property).

You can modify resources properties with **-p** option, for instance:

```
oarnodesetting -r 1 -p "besteffort=YES"
```

This allows the resource #1 to accept jobs of type *besteffort* (an admission rule forces besteffort jobs to execute on resources with the property "besteffort=YES").

Notes

Security issues

For security reasons it is hardly **recommended** to configure a read only account for the OAR database (like the above example). Thus you will be able to add it in DB_BASE_LOGIN_RO and DB_BASE_PASSWD_RO in *oar.conf*.

PostgreSQL: autovacuum

Be sure to activate the "autovacuum" feature in the "postgresql.conf" file (OAR creates and deletes a lot of records and this setting cleans the postgres database from unneeded records).

PostgreSQL: authentication

In case you've installed a PostgreSQL database remotely, if your PostgreSQL installation doesn't authorize the local connections by default, you need to enable the connections to this database for the oar users. Assuming the OAR server has the address <OAR_SERVER>, you can add the following lines in the `pg_hba.conf` file:

```
# in /etc/postgresql/8.1/main/pg_hba.conf or /var/lib/pgsql/data/pg_hba.conf
host    oar         oar_ro              <OAR_SERVER>/32    md5
host    oar         oar                 <OAR_SERVER>/32    md5
```

Using Taktuk

If you want to use taktuk to manage remote administration commands, you have to install it. You can find information about taktuk from its website: http://taktuk.gforge.inria.fr.

Then, you have to edit your oar configuration file and fill in the related parameters:

- `TAKTUK_CMD` (the path to the taktuk command)
- `PINGCHECKER_TAKTUK_ARG_COMMAND` (the command used to check resources states)
- `SCHEDULER_NODE_MANAGER_SLEEP_CMD` (command used for halting nodes)

CPUSET feature

OAR uses the CPUSET features provided by the Linux kernel >= 2.6. This enables to restrict user processes to reserved processors only and provides a powerful clean-up mechanism at the end of the jobs.

For more information, have a look at the CPUSET file.

Energy saving

Starting with version 2.4.3, OAR provides a module responsible of advanced management of wake-up/shut-down of nodes when they are not used. To activate this feature, you have to:

- provide 2 commands or scripts which will be executed on the oar server to shut-down (or set into standby) some nodes and to wake-up some nodes (configure the path of those commands into the `ENERGY_SAVING_NODE_MANAGER_WAKE_UP_CMD` and `ENERGY_SAVING_NODE_MANAGER_SHUT_DOWN_CMD` variables in oar.conf) Thes 2 commands are executed by the oar user.
- configure the `available_upto` property of all your nodes:
  - `available_upto=0` : to disable the wake-up and halt
  - `available_upto=1` : to disable the wake-up (but not the halt)
  - `available_upto=2147483647` : to disable the halt (but not the wake-up)
  - `available_upto=2147483646` : to enable wake-up/halt forever

   - `available_upto=<timestamp>` : to enable the halt, and the wake-up until the date given by <timestamp>

  Ex: to enable the feature on every nodes forever:

  ```
  oarnodesetting --sql true -p available_upto=2147483646
  ```

- activate the energy saving module by setting `ENERGY_SAVING_INTERNAL="yes"` and configuring the `ENERGY_*` variables into oar.conf
- configure the metascheduler time values into `SCHEDULER_NODE_MANAGER_IDLE_TIME`, `SCHEDULER_NODE_MANAGER_SLEEP_TIME` and `SCHEDULER_NODE_MANAGER_WAKEUP_TIME` variables of the oar.conf file.
- restart the oar server (you should see an “Almighty” process more).

You need to restart OAR each time you change an `ENERGY_*` variable. More informations are available inside the oar.conf file itself. For more details about the mechanism, take a look at the “Hulot” module documentation.

Disabling SELinux

On some distributions, SELinux is enabled by default. There is currently no OAR support for SELinux. So, you need to disable SELinux, if enabled.

Cpuset id issue

On some rare servers, the core ids are not persistent across reboot. So you need to update the cpuset ids in the resource database at startup for each computing node. You can do this by using the `/etc/oar/update_cpuset_id.sh` script. The following page give more informations on how configuring it:

  http://oar.imag.fr/wiki:old:customization_tips#start_stop_of_nodes_using_ssh_keys

2.1.4 Frontends

Installation from the packages

Instructions

For RedHat like systems:

```
# OAR provides a Yum repository.
# For more information see: http://oar.imag.fr/download#rpms

# Install OAR user for the PostgreSQL backend
yum --enablerepo=OAR install oar-user oar-user-pgsql

# or Install OAR user for the MySQL backend
yum --enablerepo=OAR install oar-user oar-user-mysql
```

For the Debian like systems:

```
# OAR is shipped as part of Debian official distributions (newer versions can be␣
↪available in backports)
# For more info see: http://oar.imag.fr/download#debian

```

(continues on next page)

(continued from previous page)

```
# Install OAR server for the PostgreSQL backend
apt-get install oar-user oar-user-pgsql

# or Install OAR server for the MySQL backend
apt-get install oar-user oar-user-mysql
```

Installation from the tarball

Requirements

For RedHat like systems:

```
# Build dependencies
yum install gcc make tar python-docutils

# Common dependencies
yum install Perl Perl-base openssh Perl-DBI

# MySQL dependencies
yum install mysql perl-DBD-MySQL

# PostgreSQL dependencies
yum install postgresql perl-DBD-Pg
```

For Debian like system:

```
# Build dependencies
apt-get install gcc make tar python-docutils

# Common dependencies
apt-get install perl perl-base openssh-client openssh-server libdbi-perl

# MySQL dependencies
apt-get install mysql-client libdbd-mysql-perl

# PostgreSQL dependencies
apt-get install postgresql-client libdbd-pg-perl
```

Instructions

Get the sources:

```
OAR_VERSION=2.5.4
wget -O - http://oar-ftp.imag.fr/oar/2.5/sources/stable/oar-${OAR_VERSION}.tar.gz |␣
↪tar xzvf -
cd oar-${OAR_VERSION}/
```

Build/Install/setup:

```
# build
make user-build
# install
make user-install
# setup
make user-setup
```

Configuration

SSH setup

OAR uses SSH to connect from machine to machine (e.g. from server or frontend to nodes or from nodes to nodes), using a dedicated SSH daemon usually running on port 6667.

Upon installtion of the OAR server on the server machine, a SSH key pair along with an authorized_keys file is created for the oar user in `/var/lib/oar/.ssh`. You need to copy that directory from the oar server to the frontend (if not the same machine).

Please note that public key in the authorized_keys file must be prefixed with `environment="OAR_KEY=1"`, e.g.:

```
environment="OAR_KEY=1" ssh-rsa AAAAB3NzaC1yc2[...]6mIcqvcwG1K7V6CHLQKHKWo/␣
↪root@server
```

Also please make sure that the `/var/lib/oar/.ssh` directory and contained files have the right ownership (oar.oar) and permissions for SSH to function.

Coherent configuration files between server node and user nodes

You need to have a coherent oar configuration between the server node and the user nodes. So you can just copy the /etc/oar/oar.conf directory from to server node to the user nodes.

About X11 usage in OAR

The easiest and scalable way to use X11 application on cluster nodes is to open X11 ports and set the right DISPLAY environment variable by hand. Otherwise users can use X11 forwarding via SSH to access cluster frontends. You must configure the SSH server on the frontends nodes with:

```
X11Forwarding yes
X11UseLocalhost no
```

With this configuration, users can launch X11 applications after a 'oarsub -I' on the given node or "oarsh -X node12".

2.1.5 API server

Description

Since the version 2.5.3, OAR offers an API for users and admins interactions. This api must be installed on a frontend node (with the user module installed).

Installation from the packages

Instructions

For RedHat like systems:

```
# OAR provides a Yum repository.
# For more information see: http://oar.imag.fr/download#rpms

# Install apache FastCGI and Suexec modules (optional but highly recommended)
```

(continues on next page)

(continued from previous page)

```
# Install OAR Restful api
yum --enablerepo=OAR install oar-restful-api
```

For the Debian like systems:

```
# OAR is shipped as part of Debian official distributions (newer versions can be␣
↪available in backports)
# For more info see: http://oar.imag.fr/download#debian

# Install apache FastCGI and Suexec modules (optional but highly recommended)

# Install OAR Restful api
apt-get install oar-restful-api
```

Installation from the tarball

Requirements

For RedHat like systems:

```
# Build dependencies
yum install gcc make tar python-docutils

# Common dependencies
yum install perl perl-base perl-DBI perl-CGI perl-JSON perl-YAML perl-libwww-perl␣
↪httpd

# Install apache FastCGI and Suexec modules (optional but highly recommended)

# MySQL dependencies
yum install perl-DBD-MySQL

# PostgreSQL dependencies
yum install perl-DBD-Pg
```

For Debian like system:

```
# Build dependencies
apt-get install gcc make tar python-docutils

# Common dependencies
apt-get install perl perl-base libdbi-perl libjson-perl libyaml-perl libwww-perl␣
↪apache2 libcgi-fast-perl

# Install apache FastCGI and Suexec modules (optional but highly recommended)

# MySQL dependencies
apt-get install libdbd-mysql-perl

# PostgreSQL dependencies
apt-get install libdbd-pg-perl
```

Instructions

Get the sources:

```
OAR_VERSION=2.5.4
wget -O - http://oar-ftp.imag.fr/oar/2.5/sources/stable/oar-${OAR_VERSION}.tar.gz |␣
↪tar xzvf -
cd oar-${OAR_VERSION}/
```

build/install/setup:

```
# build
make api-build
# install
make api-install
# setup
make api-setup
```

Configuration

Configuring OAR

For the moment, the API needs the user tools to be installed on the same host (‘`make user-install`’ or oar-user packages). A suitable `/etc/oar/oar.conf` should be present. For the API to work, you should have the oarstat/oarnodes/oarsub commands to work (on the same host you installed the API)

Configuring Apache

The api provides a default configuration file (`/etc/oar/apache-api.conf`) that is using an identd user identification enabled only from localhost. Edit the `/etc/oar/apache-api.conf` file and customize it to reflect the authentication mechanism you want to use. For ident, you may have to install a “identd” daemon on your distrib. The steps may be:

- Install and run an identd daemon on your server (like *pidentd*).
- Activate the ident auth mechanism into apache (`a2enmod ident`).
- Activate the headers apache module (`a2enmod headers`).
- Activate the rewrite apache module (`a2enmod rewrite`).
- Customize apache-api.conf to allow the hosts you trust for ident.

YAML, JSON, XML

You need at least one of the YAML or JSON perl module to be installed on the host running the API.

Test

You may test the API with a simple wget:

```
wget -O - http://localhost/oarapi/resources.html
```

It should give you the list of resources in the yaml format but enclosed in an html page. To test if the authentication works, you need to post a new job. See the example.txt file that gives you example queries with a ruby rest client.

2.1.6 Visualization server

Description

OAR provides two webapp tools for visualizing the resources utilization:

```
- monika which displays the current state of resources as well as all running and␣
↪waiting jobs
- drawgantt-svg which displays gantt chart of nodes and jobs for the past and future.
```

Installation from the packages

Instructions

For RedHat like systems:

```
# OAR provides a Yum repository.
# For more information see: http://oar.imag.fr/download#rpms

# Install OAR web status package
yum --enablerepo=OAR install oar-web-status
```

For the Debian like systems:

```
# OAR is shipped as part of Debian official distributions (newer versions can be␣
↪available in backports)
# For more info see: http://oar.imag.fr/download#debian

# Install OAR web status package
apt-get install oar-web-status
```

Installation from the tarball

Requirements

For RedHat like systems:

```
# Build dependencies
yum install gcc make tar python-docutils

# Common dependencies
yum install perl perl-base perl-DBI ruby-GD ruby-DBI perl-Tie-IxHash perl-Sort-
↪Naturally perl-AppConfig php

# MySQL dependencies
yum install mysql perl-DBD-MySQL ruby-mysql php-mysql

# PostgreSQL dependencies
yum install postgresql perl-DBD-Pg ruby-pg php-pgsql
```

For Debian like system:

```
# Build dependencies
apt-get install gcc make tar python-docutils

# Common dependencies
apt-get install perl perl-base ruby libgd-ruby1.8 libdbi-perl libtie-ixhash-perl␣
↪libappconfig-perl libsort-naturally-perl libapache2-mod-php5

# MySQL dependencies
apt-get install libdbd-mysql-perl libdbd-mysql-ruby php5-mysql
```

(continues on next page)

(continued from previous page)

```

# PostgreSQL dependencies
apt-get install libdbd-pg-perl libdbd-pg-ruby php5-pgsql
```

Instructions

Get the sources:

```
OAR_VERSION=2.5.4
wget -O - http://oar-ftp.imag.fr/oar/2.5/sources/stable/oar-${OAR_VERSION}.tar.gz |␣
↪tar xzvf -
cd oar-${OAR_VERSION}/
```

build/install/setup:

```
# build
make monika-build drawgantt-build drawgantt-svg-build www-conf-build
# install
make monika-install drawgantt-install drawgantt-svg-install www-conf-install
# setup
make monika-setup drawgantt-setup drawgantt-svg-setup www-conf-setup
```

Configuration

Monika configuration

- Edit `/etc/oar/monika.conf` to fit your configuration.

Drawgantt-SVG configuration

- Edit `/etc/oar/drawgantt-config.inc.php` to fit your configuration.

httpd configuration

- You need to edit `/etc/oar/apache.conf` to fit your needs and verify that you http server configured.

2.2 Configuration file

Be careful, the syntax of this file must be bash compliant(so after editing you must be able to launch in bash 'source /etc/oar.conf' and have variables assigned). Each configuration tag found in /etc/oar.conf is now described:

- Database type : you can use a MySQL or a PostgreSQL database (tags are "mysql" or "Pg"):

  ```
  DB_TYPE=Pg
  ```

- Database hostname:

  ```
  DB_HOSTNAME=127.0.0.1

    - Database port::

  DB_PORT=5432
  ```

- Database base name:

```
DB_BASE_NAME=oar
```

- DataBase user name:

```
DB_BASE_LOGIN=oar
```

- DataBase user password:

```
DB_BASE_PASSWD=oar
```

- DataBase read only user name:

```
DB_BASE_LOGIN_RO=oar_ro
```

- DataBase read only user password:

```
DB_BASE_PASSWD_RO=oar_ro
```

- OAR server hostname:

```
SERVER_HOSTNAME=localhost
```

- OAR server port:

```
SERVER_PORT=6666
```

- When the user does not specify a -l option then oar use this:

```
OARSUB_DEFAULT_RESOURCES="/resource_id=1"
```

- Force use of job key even if –use-job-key or -k is not set in oarsub:

```
OARSUB_FORCE_JOB_KEY="no"
```

- Specify where we are connected in the deploy queue(the node to connect to when the job is in the deploy queue):

```
DEPLOY_HOSTNAME="127.0.0.1"
```

- Specify where we are connected with a job of the cosystem type:

```
COSYSTEM_HOSTNAME="127.0.0.1"
```

- Set the directory where OAR will store its temporary files on each nodes of the cluster. This value MUST be the same in all oar.conf on all nodes:

```
OAR_RUNTIME_DIRECTORY="/tmp/oar_runtime"
```

- Specify the database field to use to fill the file on the first node of the job in $OAR_NODE_FILE (default is ‘network_address’). Only resources with type=default are displayed in this file:

```
NODE_FILE_DB_FIELD="network_address"
```

- Specify the database field that will be considered to fill the node file used by the user on the first node of the job. for each different value of this field then OAR will put 1 line in the node file(by default “cpu”):

```
NODE_FILE_DB_FIELD_DISTINCT_VALUES="core"
```

- By default OAR uses the ping command to detect if nodes are down or not. To enhance this diagnostic you can specify one of these other methods (give the complete command path):
  - OAR taktuk:

    ```
    PINGCHECKER_TAKTUK_ARG_COMMAND="-t 30 broadcast exec [ true ]"
    ```

    If you use sentinelle.pl then you must use this tag:

    ```
    PINGCHECKER_SENTINELLE_SCRIPT_COMMAND="/var/lib/oar/sentinelle.pl -t 30 -w 20"
    ```

  - OAR fping:

    ```
    PINGCHECKER_FPING_COMMAND="/usr/bin/fping -q"
    ```

  - OAR nmap : it will test to connect on the ssh port (22):

    ```
    PINGCHECKER_NMAP_COMMAND="/usr/bin/nmap -p 22 -n -T5"
    ```

  - OAR generic : a specific script may be used instead of ping to check aliveness of nodes. The script must return bad nodes on STDERR (1 line for a bad node and it must have exactly the same name that OAR has given in argument of the command):

    ```
    PINGCHECKER_GENERIC_COMMAND="/path/to/command arg1 arg2"
    ```

- OAR log level: 3(debug+warnings+errors), 2(warnings+errors), 1(errors):

  ```
  LOG_LEVEL=2
  ```

- OAR log file:

  ```
  LOG_FILE="/var/log/oar.log"
  ```

- If you want to debug oarexec on nodes then affect 1 (only effective if DETACH_JOB_FROM_SERVER = 1):

  ```
  OAREXEC_DEBUG_MODE=0
  ```

- Set the granularity of the OAR accounting feature (in seconds). Default is 1 day (86400s):

  ```
  ACCOUNTING_WINDOW="86400"
  ```

- OAR informations may be notified by email to the administror. Set accordingly to your configuration the next lines to activate this feature:

  ```
  MAIL_SMTP_SERVER="smtp.serveur.com"
  MAIL_RECIPIENT="user@domain.com"
  MAIL_SENDER="oar@domain.com"
  ```

- Set the timeout for the prologue and epilogue execution on computing nodes:

  ```
  PROLOGUE_EPILOGUE_TIMEOUT=60
  ```

- Files to execute before and after each job on the first computing node (by default nothing is executed):

  ```
  PROLOGUE_EXEC_FILE="/path/to/prog"
  EPILOGUE_EXEC_FILE="/path/to/prog"
  ```

- Set the timeout for the prologue and epilogue execution on the OAR server:

```
SERVER_PROLOGUE_EPILOGUE_TIMEOUT=60
```

- Files to execute before and after each job on the OAR server (by default nothing is executed):

```
SERVER_PROLOGUE_EXEC_FILE="/path/to/prog"
SERVER_EPILOGUE_EXEC_FILE="/path/to/prog"
```

- Set the frequency for checking Alive and Suspected resources:

```
FINAUD_FREQUENCY=300
```

- Set time after which resources become Dead (default is 0 and it means never):

```
DEAD_SWITCH_TIME=600
```

- Maximum of seconds used by a scheduler:

```
SCHEDULER_TIMEOUT=20
```

- Time to wait when a reservation has not got all resources that it has reserved (some resources could have become Suspected or Absent since the job submission) before to launch the job in the remaining resources:

```
RESERVATION_WAITING_RESOURCES_TIMEOUT=300
```

- Time to add between each jobs (time for administration tasks or time to let computers to reboot):

```
SCHEDULER_JOB_SECURITY_TIME=1
```

- Minimum time in seconds that can be considered like a hole where a job could be scheduled in:

```
SCHEDULER_GANTT_HOLE_MINIMUM_TIME=300
```

- You can add an order preference on resource assigned by the system(SQL ORDER syntax):

```
SCHEDULER_RESOURCE_ORDER="switch ASC, network_address DESC, resource_id ASC"
```

- You can specify resources from a resource type that will be always assigned for each job (for example: enable all jobs to be able to log on the cluster frontales). For more information, see the FAQ:

```
SCHEDULER_RESOURCES_ALWAYS_ASSIGNED_TYPE="42 54 12 34"
```

- This says to the scheduler to treate resources of these types, where there is a suspended job, like free ones. So some other jobs can be scheduled on these resources. (list resource types separate with spaces; Default value is nothing so no other job can be scheduled on suspended job resources):

```
SCHEDULER_AVAILABLE_SUSPENDED_RESOURCE_TYPE="default licence vlan"
```

- Name of the perl script that manages suspend/resume. You have to install your script in $OARDIR and give only the name of the file without the entire path. (default is suspend_resume_manager.pl):

```
SUSPEND_RESUME_FILE="suspend_resume_manager.pl"
```

- Files to execute just after a job was suspended and just before a job was resumed:

```
JUST_AFTER_SUSPEND_EXEC_FILE="/path/to/prog"
JUST_BEFORE_RESUME_EXEC_FILE="/path/to/prog"
```

- Timeout for the two previous scripts:

```
SUSPEND_RESUME_SCRIPT_TIMEOUT=60
```

- Indicate the name of the database field that contains the cpu number of the node. If this option is set then users must use oarsh instead of ssh to walk on each nodes that they have reserved via oarsub.

```
JOB_RESOURCE_MANAGER_PROPERTY_DB_FIELD=cpuset
```

- Name of the perl script that manages cpuset. You have to install your script in $OARDIR and give only the name of the file without the entire path. (default is cpuset_manager.pl which handles the linux kernel cpuset)

```
JOB_RESOURCE_MANAGER_FILE="cpuset_manager.pl"
```

- Resource "type" DB field to use if you want to enable the job uid feature. (create a unique user id per job on each nodes of the job)

```
JOB_RESOURCE_MANAGER_JOB_UID_TYPE="userid"
```

- If you have installed taktuk and want to use it to manage cpusets then give the full command path (with your options except "-m" and "-o" and "-c"). You don't also have to give any taktuk command.(taktuk version must be >= 3.6)

```
TAKTUK_CMD="/usr/bin/taktuk -s"
```

- If you want to manage nodes to be started and stoped. OAR gives you this API:
- When OAR scheduler wants some nodes to wake up then it launches this command and puts on its STDIN the list of nodes to wake up (one hostname by line).The scheduler looks at *available_upto* field in the *resources* table to know if the node will be started for enough time:

```
SCHEDULER_NODE_MANAGER_WAKE_UP_CMD="/path/to/the/command with your args"
```

- When OAR considers that some nodes can be shut down, it launches this command and puts the node list on its STDIN(one hostname by line):

```
SCHEDULER_NODE_MANAGER_SLEEP_CMD="/path/to/the/command args"
```

- Parameters for the scheduler to decide when a node is idle(number of seconds since the last job was terminated on the nodes):

```
SCHEDULER_NODE_MANAGER_IDLE_TIME=600
```

- Parameters for the scheduler to decide if a node will have enough time to sleep(number of seconds before the next job):

```
SCHEDULER_NODE_MANAGER_SLEEP_TIME=600
```

- Command to use to connect to other nodes (default is "ssh" in the PATH)

```
OPENSSH_CMD="/usr/bin/ssh"
```

- These are configuration tags for OAR in the desktop-computing mode:

```
DESKTOP_COMPUTING_ALLOW_CREATE_NODE=0
DESKTOP_COMPUTING_EXPIRY=10
STAGEOUT_DIR="/var/lib/oar/stageouts/"
```

(continues on next page)

(continued from previous page)

```
STAGEIN_DIR="/var/lib/oar/stageins"
STAGEIN_CACHE_EXPIRY=144
```

- This variable must be set to enable the use of oarsh from a frontale node. Otherwise you must not set this variable if you are not on a frontale:

```
OARSH_OARSTAT_CMD="/usr/bin/oarstat"
```

- The following variable adds options to ssh. If one option is not handled by your ssh version just remove it BUT be careful because these options are there for security reasons:

```
OARSH_OPENSSH_DEFAULT_OPTIONS="-oProxyCommand=none -oPermitLocalCommand=no"
```

2.3 Admin commands

2.3.1 oarproperty

This command manages OAR resource properties stored in the database.

Options are:

```
-l : list properties
-a NAME : add a property
  -c : sql new field of type VARCHAR(255) (default is integer)
-d NAME : delete a property
-r "OLD_NAME,NEW_NAME" : rename property OLD_NAME into NEW_NAME
```

Examples:

```
# oarproperty -a cpu_freq
# oarproperty -r "cpu_freq,freq"
```

2.3.2 oarnodesetting

This command allows to change the state or a property of a node or of several resources resources.

By default the node name used by *oarnodesetting* is the result of the command *hostname*.

Options are:

```
-r, --resource [resource_id]         Resource id of the resource to modify
-h, --hostname [hostname]            Hostname for the resources to modify
-f, --file [file]                    Get a hostname list from a file (1
                                     hostname by line) for resources to modify
    --sql [SQL]                      Select resources to modify from database
                                     using a SQL where clause on the resource
                                     table (e.g.: "type = 'default'")
-a, --add                            Add a new resource
-s, --state=state                    Set the new state of the node
-m, --maintenance [on|off]           Set/unset maintenance mode for resources,
                                     this is equivalent to setting its state
                                     to Absent and its available_upto to 0
-d, --drain [on|off]                 Prevent new job to be scheduled on
```

(continues on next page)

(continued from previous page)

```
                                     resources, this is equivalent to setting
                                     the drain property to YES
-p, --property ["property=value"]    Set the property of the resource to the
                                     given value
-n, --no-wait                        Do not wait for job end when the node
                                     switches to Absent or Dead
    --last-property-value [property] Get the last value used for a property (as
                                     sorted by SQL's ORDER BY DESC)
```

2.3.3 oaradmissionrules

This command is used to add, edit or remove the admission rules.

The admission rules are a piece of Perl code that is executed in the oarsub command just before to submit the job to the system.

Options are:

```
-S, --show-all
  -Y, --enabled             show enabled admission rules only
  -N, --disabled            show disabled admission rules only

-S, --show-all
-s, --show [rule-id]
  -I, --id                  show #rule-id only
  -f, --full                show full script
  -H, --no-header           show script only
  -w, --to-file [filename]  write script to file (%% replaced by #rule-id)

-n, --new
-m, --modify <rule-id>
  -e, --edit [cmd]          edit script using editor or cmd if provided
  -r, --from-file <filename> read script from file instead of running editor
  -P, --priority <priority> set priority for rule
  -Y, --enabled             enable admission rule
  -N, --disabled            disable admission rule

-d, --delete <rule-id>
  no option
```

2.3.4 oarremoveresource

This command permits to remove a resource from the database.

The node must be in the state "Dead" (use *oarnodesetting* to do this) and then you can use this command to delete it.

Be aware that it also removes the history of all the jobs that have run on this resource.

2.3.5 oaraccounting

This command permits to update the *accounting* table for jobs ended since the last launch.

Option `--reinitialize` removes everything in the *accounting* table and switches the "accounted" field of the table *jobs* into "NO". So when you will launch the *oaraccounting* command again, it will take the whole jobs.

Option `--delete_before` removes records from the *accounting* table that are older than the amount of time specified. So if the table becomes too big you can shrink old data; for example:

```
oaraccounting --delete_before 2678400
```

(Remove everything older than 31 days)

2.3.6 oarnotify

This command sends commands to the *Almighty* module and manages scheduling queues.

Option are:

```
    Almighty_tag    send this tag to the Almighty (default is TERM)
-e                  active an existing queue
-d                  inactive an existing queue
-E                  active all queues
-D                  inactive all queues
--add_queue         add a new queue; syntax is name,priority,scheduler
                    (ex: "name,3,oar_sched_gantt_with_timesharing"
--remove_queue      remove an existing queue
-l                  list all queues and there status
-h                  show this help screen
-v                  print OAR version number
```

2.3.7 oar-database

This command create, initialize, upgrade, reset and drop the oar database.

2.3.8 oar_resource_init

Connect to a list of hosts to gather system information and create the corresponding OAR resources.

Hosts are read one per line from a file or STDIN.

The command either generates a script which could be executed afterward, or directly executes the OAR commands (oarnodesetting and oarproperty).

The following OAR resource hierarchy is assumed:

host > cpu > core

Or if the *-T* option is set:

host > cpu > core > thread

The mem property is set along with the hierarchy.

Other properties are not set, however the generated script can be modified to do so, or the oarnodesetting command can be used to set them afterward.

2.3.9 oar_resource_add

Yet another helper script to define OAR resources

This tool generates the oarproperty and oarnodesetting commands to create OAR resources following the host / cpu / core (/ thread) hierarchy, possibly with GPU alongside.

REMINDER: Each physical element (each cpu, each core, each thread, each gpu) must have a unique identifier in the OAR resources database. If some resources already exists in the database (e.g. from a previously installed cluster), offsets can be given in the command line or guessed with the auto-offset option, so that identifiers for newly created resources are unique.

This tool is also a good example of how one can create OAR resources using script loops and the oarnodesetting command. If it does not exactly fit your needs, feel free to read the script code and adapt it.

The oar_resource_add tool does not look at the actual hardware topology of the target machines. Core and GPU device affinity to CPU may not be correct. See the *hwloc* commands for instance to find out the correct topology and affinity, and use the –cputopo and –gputopo options accordingly.

For more details, read the manual pages of the commands.

2.4 Admin REST - API

The OAR REST API is currently a cgi script being served by an http server (we recommend Apache) that allows the programming of interfaces to OAR using a REST library. Most of the operations usually done with the oar Unix commands may be done using this API from the favourite language of the users. But this API may also be used as an administrator portal as it provides you a convenient way to create resources, edit configuration variables or admission rules.

2.4.1 Installation

. . . To be written. . .

2.4.2 Authentication setup

The API authentication relies on the authentication mechanism of the http server used to serve the CGI script. The API may be configured to use the IDENT protocol for authentication from trusted hosts, like a cluster frontend. In this case, a unix login is automatically used by the API. This only works for hosts that have been correctly configured (for which the security rules are trusted by the admistrator). If IDENT is not used or not trusted, the API can use the basic HTTP authentication. You may also want to set-up https certificates.

In summary, the API authentication is based on the http server's configuration. The API uses the **X_REMOTE_IDENT** http header variable, so the administrator has to set up this variable inside the http server configuration. Look at the provided apache sample configuration files (api/apache2.conf of the OAR sources or the installed /etc/oar/apache-api.conf of packages) for more details.

2.5 Security aspects

In OAR, security and user switching is managed by the "oardodo" command. It is a suid binary which can be executed only by root and the oar group members that is used to launch a command, a terminal or a script with the privileges of a particular user. When "oardodo" is called, it checks the value of an environment variable: `OARDO_BECOME_USER`.

- If this variable is empty, "oardodo" will execute the command with the privileges of the superuser (root).
- Else, this variable contains the name of the user that will be used to execute the command.

Here are the scripts/modules where "oardodo" is called and which user is used during this call:

- OAR::Modules::Judas: this module is used for logging and notification.
  - user notification: email or command execution. OARDO_BECOME_USER = user
- oarsub: this script is used for submitting jobs or reservations.
  - read user script
  - connection to the job and the remote shell
  - keys management
  - job key export

  for all these functions, the user used in the OARDO_BECOME_USER variable is the user that submits the job.
- pingchecker: this module is used to check resources health. Here, the user is root.
- oarexec: executed on the first reserved node, oarexec executes the job prologue and initiate the job.
  - the "clean" method kills every oarsub connection process in superuser mode
  - "kill_children" method kills every child of the process in superuser mode
  - execution of a passive job in user mode
  - getting of the user shell in user mode
  - checkpointing in superuser mode
- job_resource_manager: The job_resource_manager script is a perl script that oar server deploys on nodes to manage cpusets, users, job keys...
  - cpuset creation and clean is executed in superuser mode
- oarsh_shell: shell program used with the oarsh script. It adds its own process in the cpuset and launches the shell or the script of the user.
  - cpuset filling, "nice" and display management are executed as root.
  - TTY login is executed as user.
- oarsh: oar's ssh wrapper to connect from node to node. It contains all the context variables usefull for this connection.
  - **display management and connection with a user job key file are executed** as user.

2.6 Modules descriptions

OAR can be decomposed into several modules which perform different tasks.

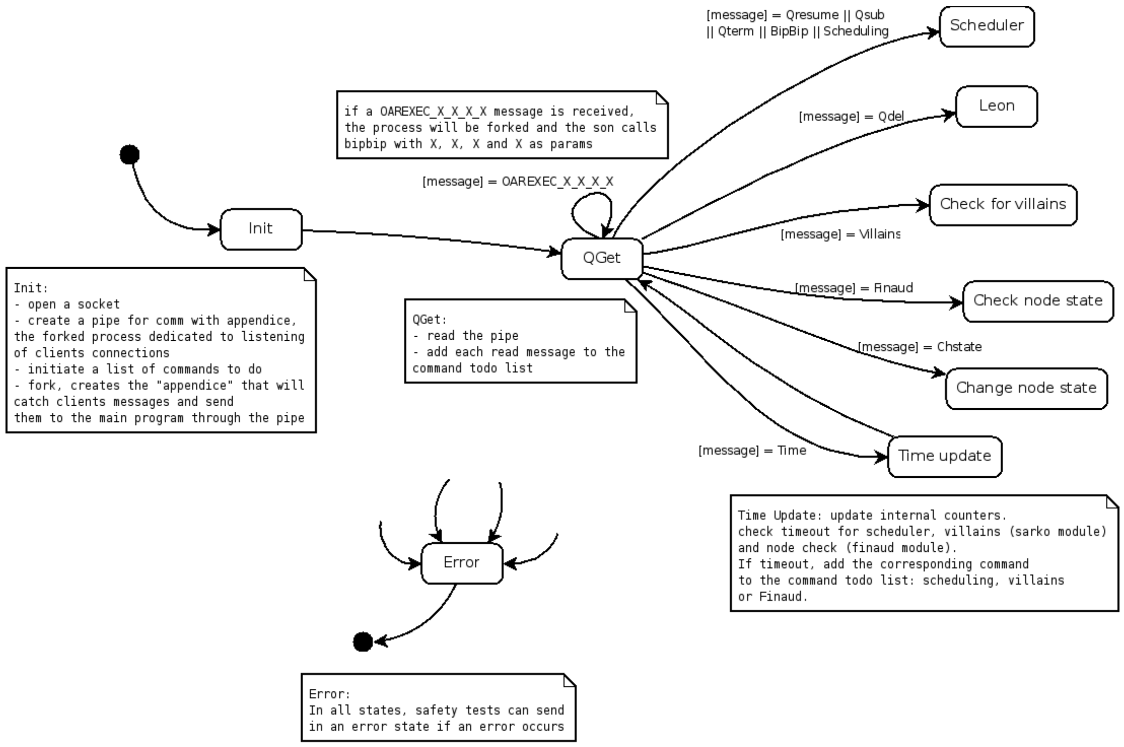
2.6.1 Almighty

This module is the OAR server. It decides what actions must be performed. It is divided into 3 processes:

- One listens to a TCP/IP socket. It waits informations or commands from OAR user program or from the other modules.
- Another one deals with commands thanks to an automaton and launch right modules one after one.
- The third one handles a pool of forked processes that are used to launch and stop the jobs.

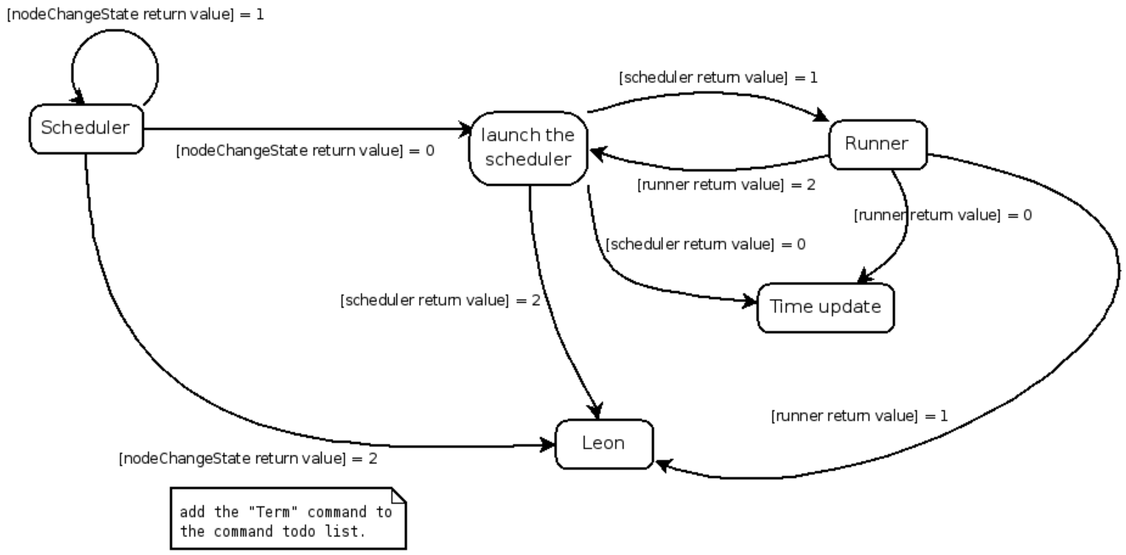
It's behaviour is represented in these schemes.

- General schema:

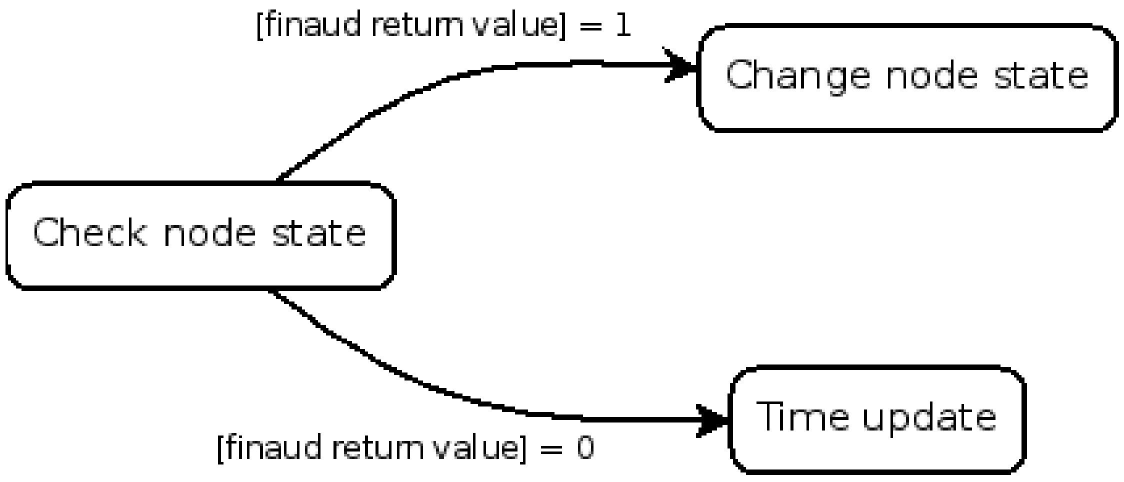

[message] = Qresume || Qsub || Qterm || BipBip || Scheduling
Scheduler
if a OAREXEC_X_X_X_X message is received, the process will be forked and the son calls bipbip with X, X, X and X as params
[message] = Qdel
Leon
[message] = OAREXEC_X_X_X_X
Check for villains
Init
QGet
[message] = Villains
Init:
- open a socket
- create a pipe for comm with appendice, the forked process dedicated to listening of clients connections
- initiate a list of commands to do
- fork, creates the "appendice" that will catch clients messages and send them to the main program through the pipe
[message] = Finaud
Check node state
QGet:
- read the pipe
- add each read message to the command todo list
[message] = Chstate
Change node state
[message] = Time
Time update
Time Update: update internal counters. check timeout for scheduler, villains (sarko module) and node check (finaud module). If timeout, add the corresponding command to the command todo list: scheduling, villains or Finaud.
Error
Error: In all states, safety tests can send in an error state if an error occurs

When the Almighty automaton starts it will first open a socket and creates a pipe for the process communication with it's forked son. Then, Almighty will fork itself in a process called "appendice" which role is to listen to incoming connections on the socket and catch clients messages. These messages will be thereafter piped to Almighty. Then, the automaton will change it's state according to what message has been received.

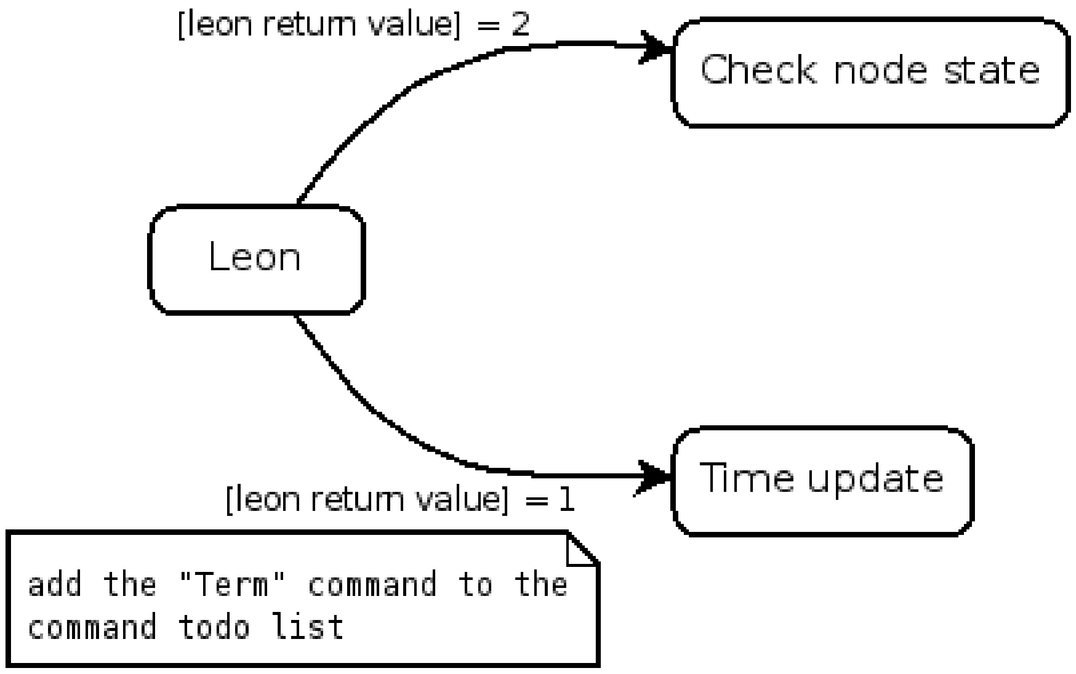
- Scheduler schema:


[nodeChangeState return value] = 1
Scheduler
[nodeChangeState return value] = 0
launch the scheduler
[scheduler return value] = 1
Runner
[runner return value] = 2
[runner return value] = 0
[scheduler return value] = 0
[scheduler return value] = 2
Time update
[runner return value] = 1
Leon
[nodeChangeState return value] = 2
add the "Term" command to the command todo list.

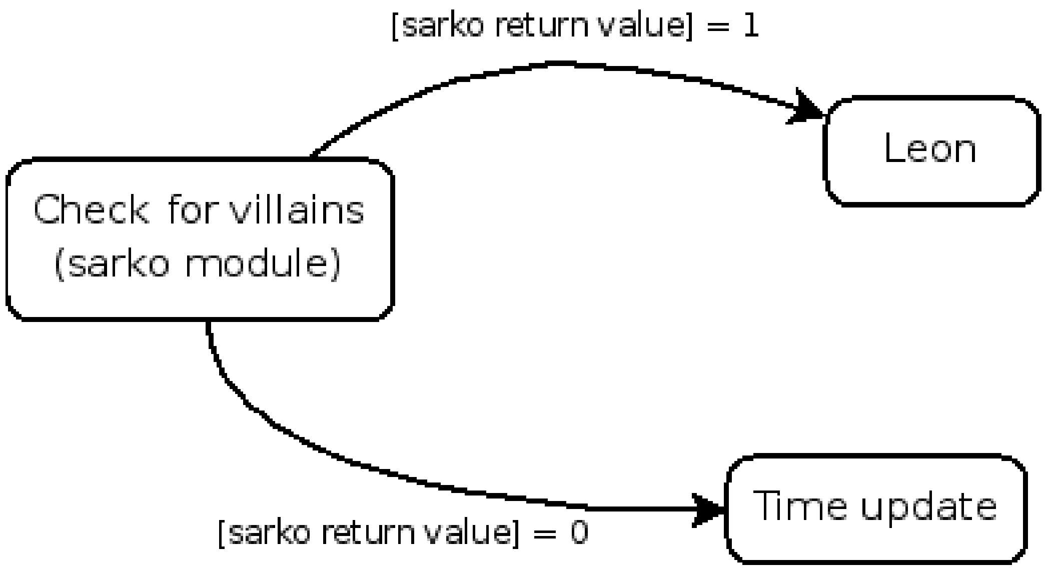
- Finaud schema:



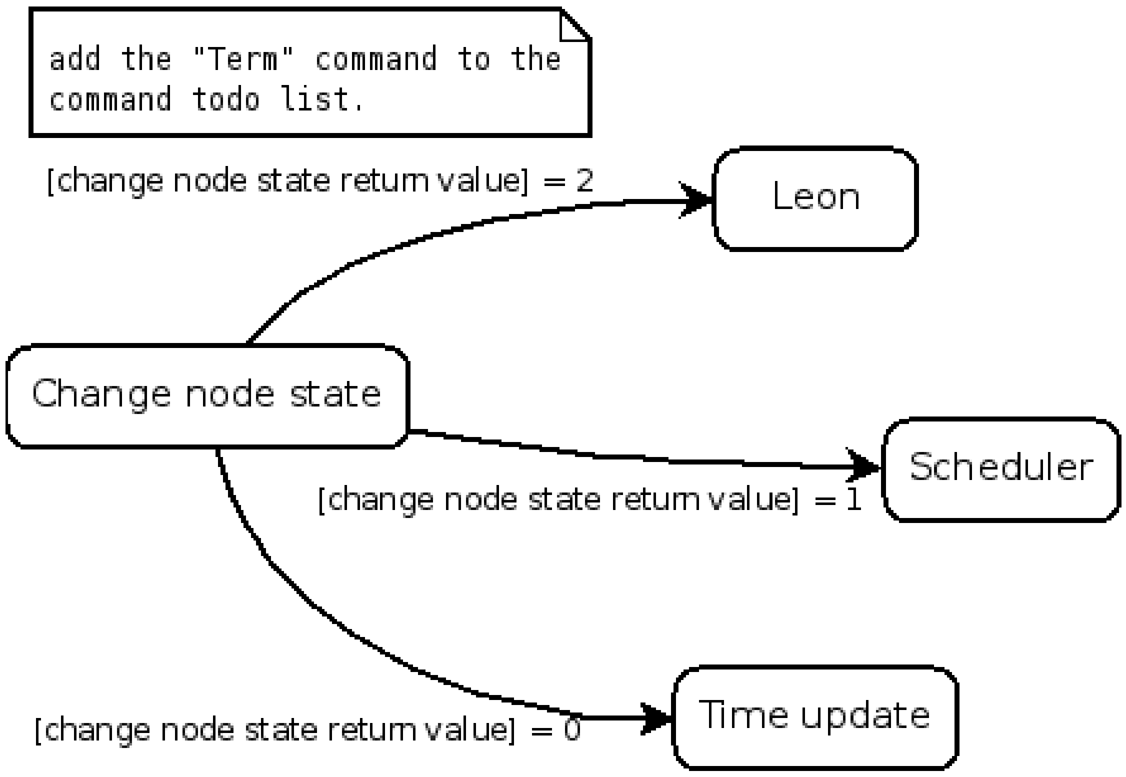
* Leon schema:



* Sarko schema:



- ChangeNode schema:


add the "Term" command to the command todo list.
[change node state return value] = 2
Leon
Change node state
Scheduler
[change node state return value] = 1
Time update
[change node state return value] = 0

2.6.2 Sarko

This module is executed periodically by the Almighty (default is every 30 seconds).

The jobs of Sarko are :

- Look at running job walltimes and ask to frag them if they had expired.

- Detect if fragged jobs are really fragged otherwise asks to exterminate them.
- In "Desktop Computing" mode, it detects if a node date has expired and asks to change its state into "Suspected".
- Can change "Suspected" resources into "Dead" after *DEAD_SWITCH_TIME* seconds.

2.6.3 Judas

This is the module dedicated to print and log every debugging, warning and error messages.

The notification functions are the following:

- send_mail(mail_recipient_address, object, body, job_id) that sends emails to the OAR admin
- notify_user(base, method, host, user, job_id, job_name, tag, comments) that parses the notify method. This method can be a user script or a mail to send. If the "method" field begins with "mail:", notify_user will send an email to the user. If the beginning is "exec:", it will execute the script as the "user".

The main logging functions are the following:

- redirect_everything() this function redirects STDOUT and STDERR into the log file
- oar_debug(message)
- oar_warn(message)
- oar_error(message)

The three last functions are used to set the log level of the message.

2.6.4 Leon

This module is in charge to delete the jobs. Other OAR modules or commands can ask to kill a job and this is Leon which performs that.

There are 2 frag types :

- *normal* : Leon tries to connect to the first node allocated for the job and terminates the job.
- *exterminate* : after a timeout if the *normal* method did not succeed then Leon notifies this case and clean up the database for these jobs. So OAR doesn't know what occured on the node and Suspects it.

2.6.5 NodeChangeState

This module is in charge of changing resource states and checking if there are jobs on these.

It also checks all pending events in the table *event_logs*.

2.6.6 Scheduler

This module checks for each reservation jobs if it is valid and launches them at the right time.

Scheduler launches all gantt scheduler in the order of the priority specified in the database and update all visualization tables (*gantt_jobs_predictions_visu* and *gantt_jobs_resources_visu*).

It also trigger if a job has to be launched.

oar_sched_gantt_with_timesharing

This is a OAR scheduler. It implements functionalities like timesharing, moldable jobs, *besteffort jobs*, ...

We have implemented the FIFO with backfilling algorithm. Some parameters can be changed in the *configuration file* (see *SCHEDULER_TIMEOUT*, *SCHEDULER_JOB_SECURITY_TIME*, *SCHEDULER_GANTT_HOLE_MINIMUM_TIME*, *SCHEDULER_RESOURCE_ORDER*).

oar_sched_gantt_with_timesharing_and_fairsharing

This scheduler is the same than *oar_sched_gantt_with_timesharing* but it looks at the consumption past and try to order waiting jobs with fairsharing in mind.

Some parameters can be changed directly in the file

```
###############################################################################
# Fairsharing parameters #
##########################
# Avoid problems if there are too many waiting jobs
my $Karma_max_number_of_jobs_treated = 1000;
# number of seconds to consider for the fairsharing
my $Karma_window_size = 3600 * 30;
# specify the target percentages for project names (0 if not specified)
my $Karma_project_targets = {
    first => 75,
    default => 25
};

# specify the target percentages for users (0 if not specified)
my $Karma_user_targets = {
    oar => 100
};
# weight given to each criteria
my $Karma_coeff_project_consumption = 3;
my $Karma_coeff_user_consumption = 2;
my $Karma_coeff_user_asked_consumption = 1;
###############################################################################
```

This scheduler takes its historical data in the *accounting* table. To fill this, the command *oaraccounting* has to be run periodically (in a cron job for example). Otherwise the scheduler cannot be aware of new user consumptions.

oar_sched_gantt_with_timesharing_and_fairsharing_and_quotas

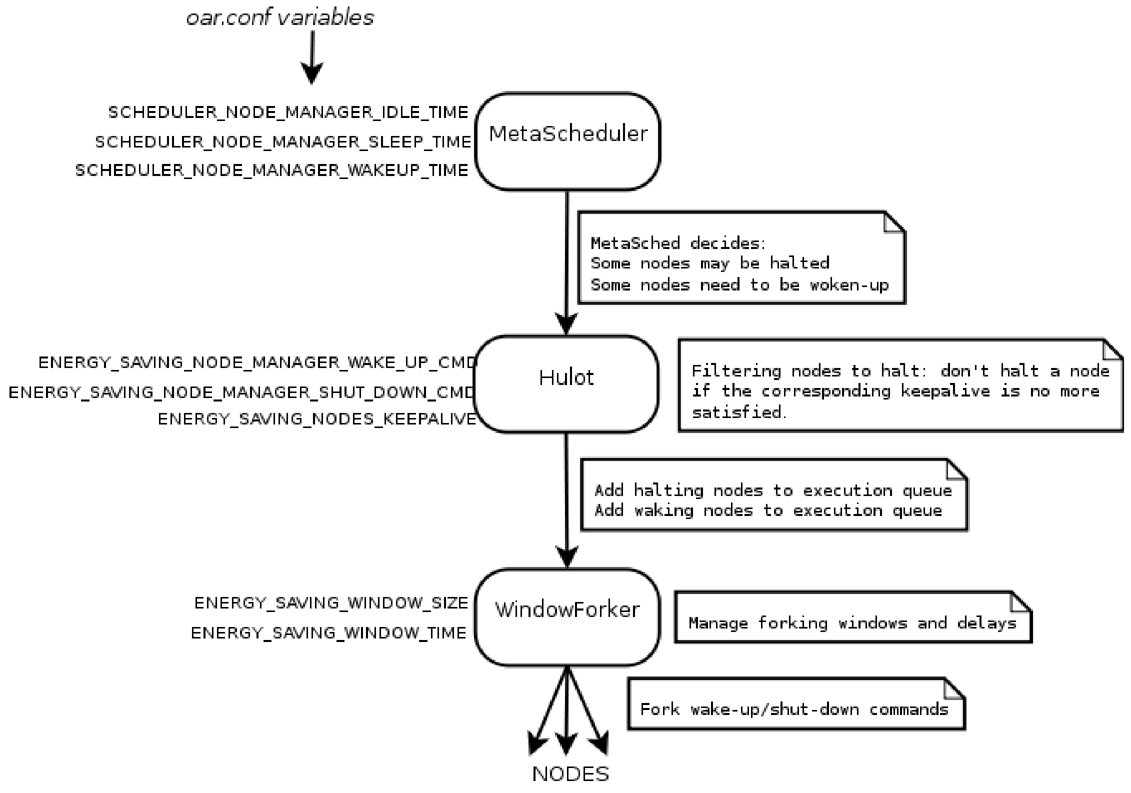
This scheduler is the same than oar_sched_gantt_with_timesharingand_fairsharing but it implements quotas which are configured in "/etc/oar/scheduler_quotas.conf".

2.6.7 Hulot

This module is responsible of the advanced management of the standby mode of the nodes. It's related to the energy saving features of OAR. It is an optional module activated with the ENERGY_SAVING_INTERNAL=yes configuration variable.

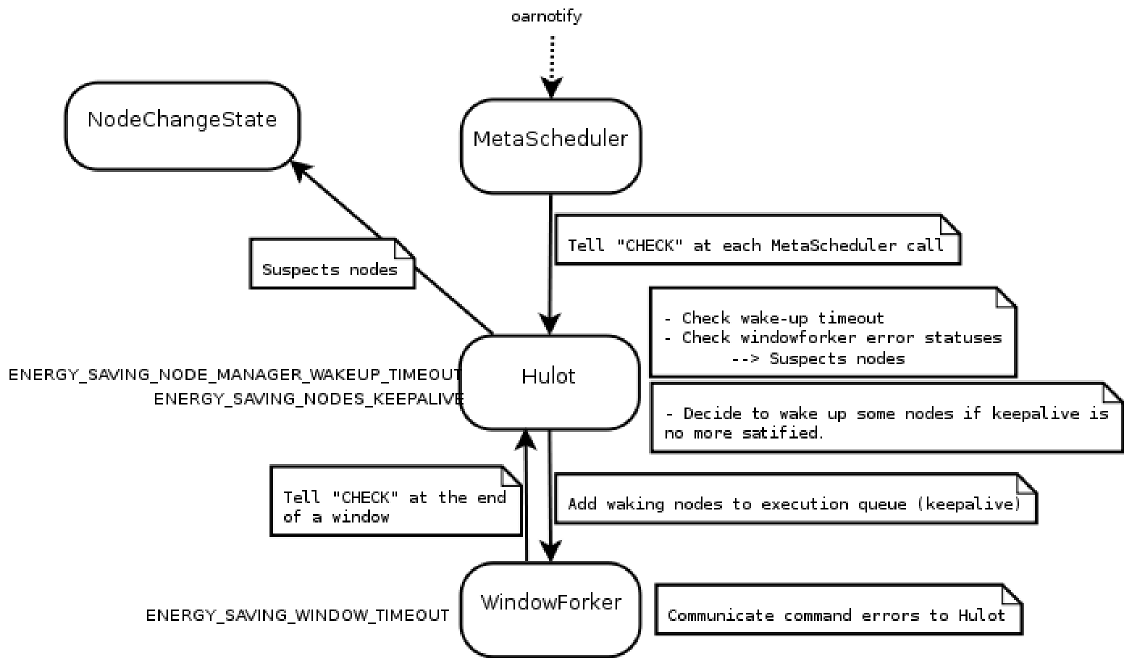
It runs as a fourth "Almighty" daemon and opens a pipe on which it receives commands from the MetaScheduler. It also communicates with a library called "WindowForker" that is responsible of forking shut-down/wake-up commands in a way that not too much commands are started at a time.

- Hulot general commands process schema:


oar.conf variables
SCHEDULER_NODE_MANAGER_IDLE_TIME
SCHEDULER_NODE_MANAGER_SLEEP_TIME
SCHEDULER_NODE_MANAGER_WAKEUP_TIME
MetaScheduler
MetaSched decides:
Some nodes may be halted
Some nodes need to be woken-up
ENERGY_SAVING_NODE_MANAGER_WAKE_UP_CMD
ENERGY_SAVING_NODE_MANAGER_SHUT_DOWN_CMD
ENERGY_SAVING_NODES_KEEPALIVE
Hulot
Filtering nodes to halt: don't halt a node if the corresponding keepalive is no more satisfied.
Add halting nodes to execution queue
Add waking nodes to execution queue
ENERGY_SAVING_WINDOW_SIZE
ENERGY_SAVING_WINDOW_TIME
WindowForker
Manage forking windows and delays
Fork wake-up/shut-down commands
NODES

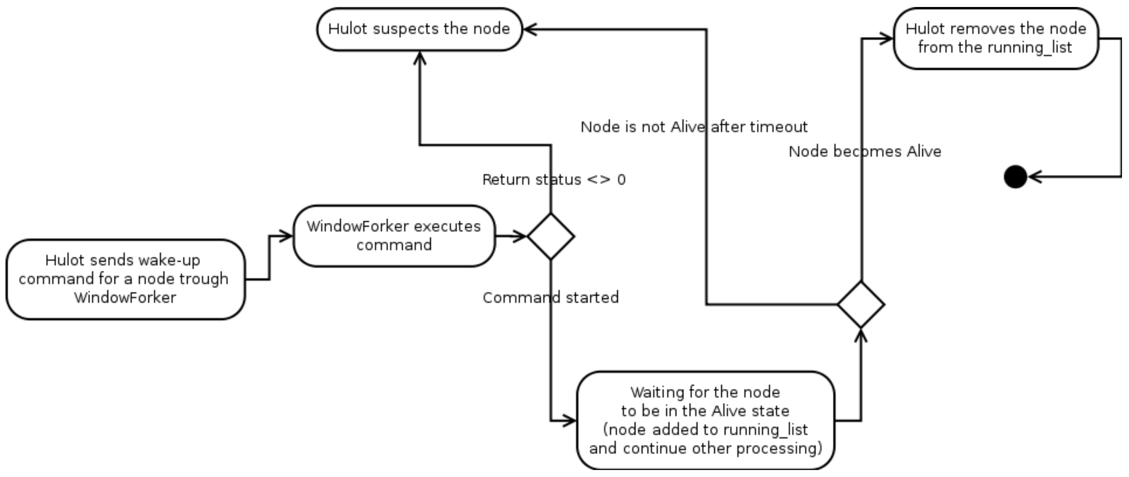
When Hulot is activated, the metascheduler sends, each time it is executed, a list of nodes that need to be woken-up or may be halted. Hulot maintains a list of commands that have already been sent to the nodes and asks to the windowforker to actually execute the commands only when it is appropriate. A special feature is the “keepalive” of nodes depending on some properties: even if the metascheduler asks to shut-down some nodes, it’s up to Hulot to check if the keepalive constraints are still satisfied. If not, Hulot refuses to halt the corresponding nodes.

- Hulot checking process schema:

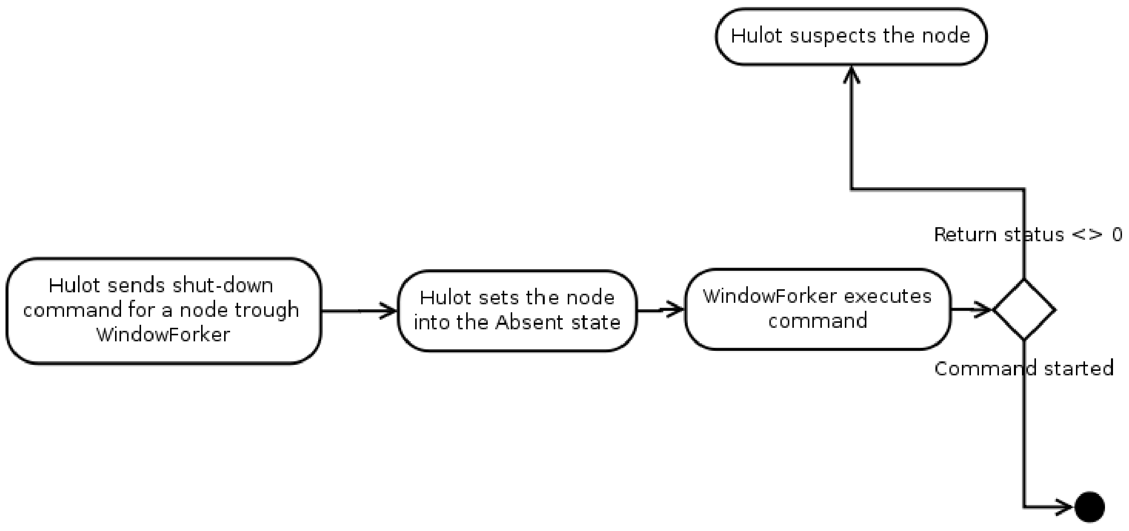

oarnotify
NodeChangeState
MetaScheduler
Tell "CHECK" at each MetaScheduler call
Suspects nodes
- Check wake-up timeout
- Check windowforker error statuses
--> Suspects nodes
ENERGY_SAVING_NODE_MANAGER_WAKEUP_TIMEOUT
ENERGY_SAVING_NODES_KEEPALIVE
Hulot
- Decide to wake up some nodes if keepalive is no more satified.
Tell "CHECK" at the end of a window
Add waking nodes to execution queue (keepalive)
ENERGY_SAVING_WINDOW_TIMEOUT
WindowForker
Communicate command errors to Hulot

Hulot is called each time the metascheduler is called, to do all the checking process. This process is also executed when Hulot receives normal halt or wake-up commands from the scheduler. Hulot checks if waking-up nodes are actually Alive or not and suspects the nodes if they haven't woken-up before the timeout. It also checks keepalive constraints and decides to wake-up nodes if a constraint is no more satisfied (for example because new jobs are running on nodes that are now busy, and no more idle). Hulot also checks the results of the commands sent by the windowforker and may also suspect a node if the command exited with non-zero status.

- Hulot wake-up process schema


Hulot suspects the node
Hulot removes the node from the running_list
Node is not Alive after timeout
Node becomes Alive
Return status <> 0
WindowForker executes command
Hulot sends wake-up command for a node trough WindowForker
Command started
Waiting for the node to be in the Alive state (node added to running_list and continue other processing)

- Hulot shutdown process schema


Hulot sends shut-down command for a node trough WindowForker
Hulot sets the node into the Absent state
WindowForker executes command
Return status <> 0
Hulot suspects the node
Command started

2.7 Internal mechanisms

2.7.1 Job execution

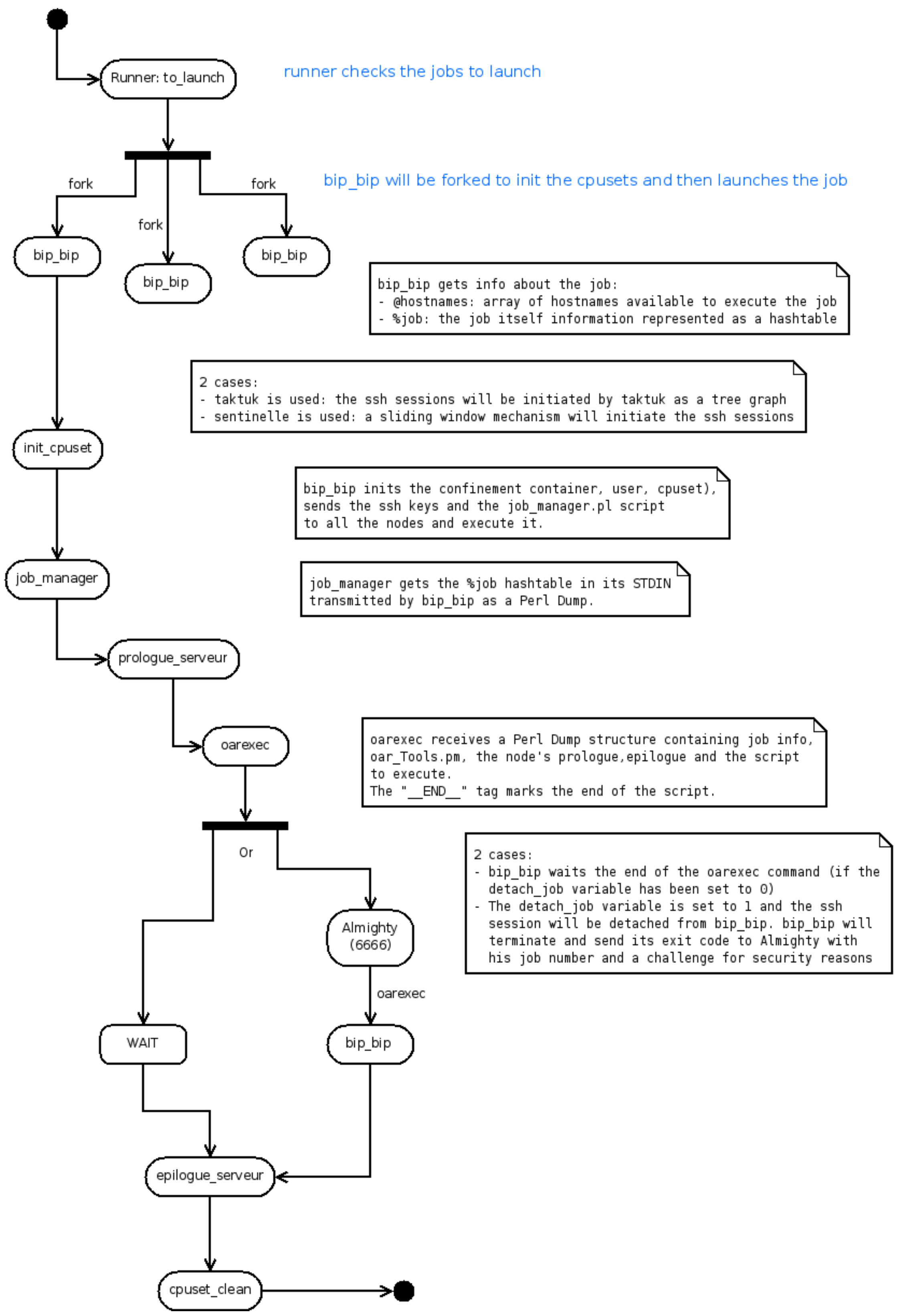
2.7.2 Scheduling

2.8 Database scheme

Note : all dates and duration are stored in an integer manner (number of seconds since the EPOCH).

2.8.1 accounting

Fields	Types	Descriptions
window_start	INT UNSIGNED	start date of the accounting interval
window_stop	INT UNSIGNED	stop date of the accounting interval
accounting_user	VARCHAR(20)	user name
accounting_project	VARCHAR(255)	name of the related project
queue_name	VARCHAR(100)	queue name
consumption_type	ENUM("ASKED", "USED")	"ASKED" corresponds to the walltimes specified by the user. "USED" corresponds to the effective time used by the user.
consumption	INT UNSIGNED	number of seconds used



```
bip_bip gets info about the job:
- @hostnames: array of hostnames available to execute the job
- %job: the job itself information represented as a hashtable
```

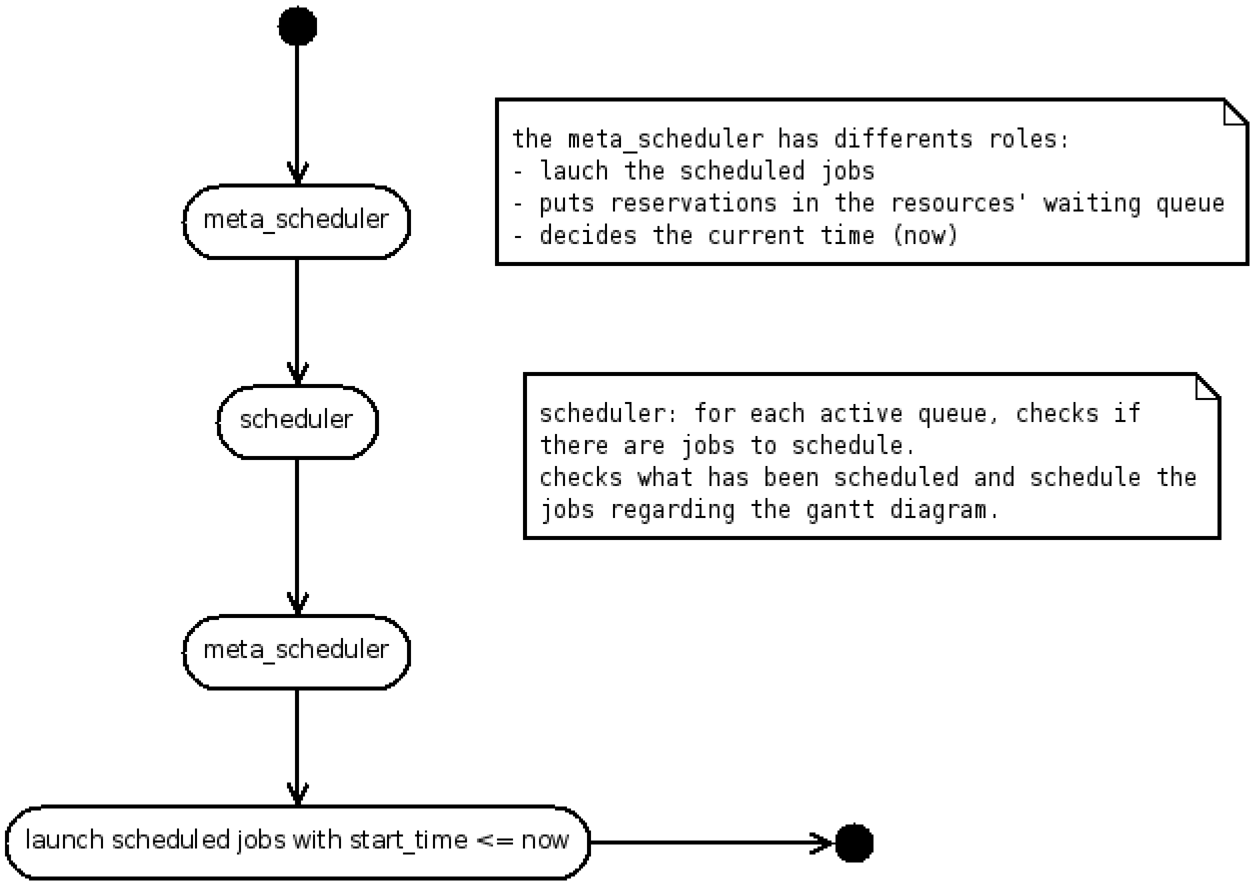
```
2 cases:
- taktuk is used: the ssh sessions will be initiated by taktuk as a tree graph
- sentinelle is used: a sliding window mechanism will initiate the ssh sessions
```

```
bip_bip inits the confinement container, user, cpuset),
sends the ssh keys and the job_manager.pl script
to all the nodes and execute it.
```

```
job_manager gets the %job hashtable in its STDIN
transmitted by bip_bip as a Perl Dump.
```

```
oarexec receives a Perl Dump structure containing job info,
oar_Tools.pm, the node's prologue,epilogue and the script
to execute.
The "__END__" tag marks the end of the script.
```

```
2 cases:
- bip_bip waits the end of the oarexec command (if the
  detach_job variable has been set to 0)
- The detach_job variable is set to 1 and the ssh
  session will be detached from bip_bip. bip_bip will
  terminate and send its exit code to Almighty with
  his job number and a challenge for security reasons
```


meta_scheduler
scheduler
meta_scheduler
launch scheduled jobs with start_time <= now
the meta_scheduler has differents roles:
- lauch the scheduled jobs
- puts reservations in the resources' waiting queue
- decides the current time (now)
scheduler: for each active queue, checks if there are jobs to schedule.
checks what has been scheduled and schedule the jobs regarding the gantt diagram.

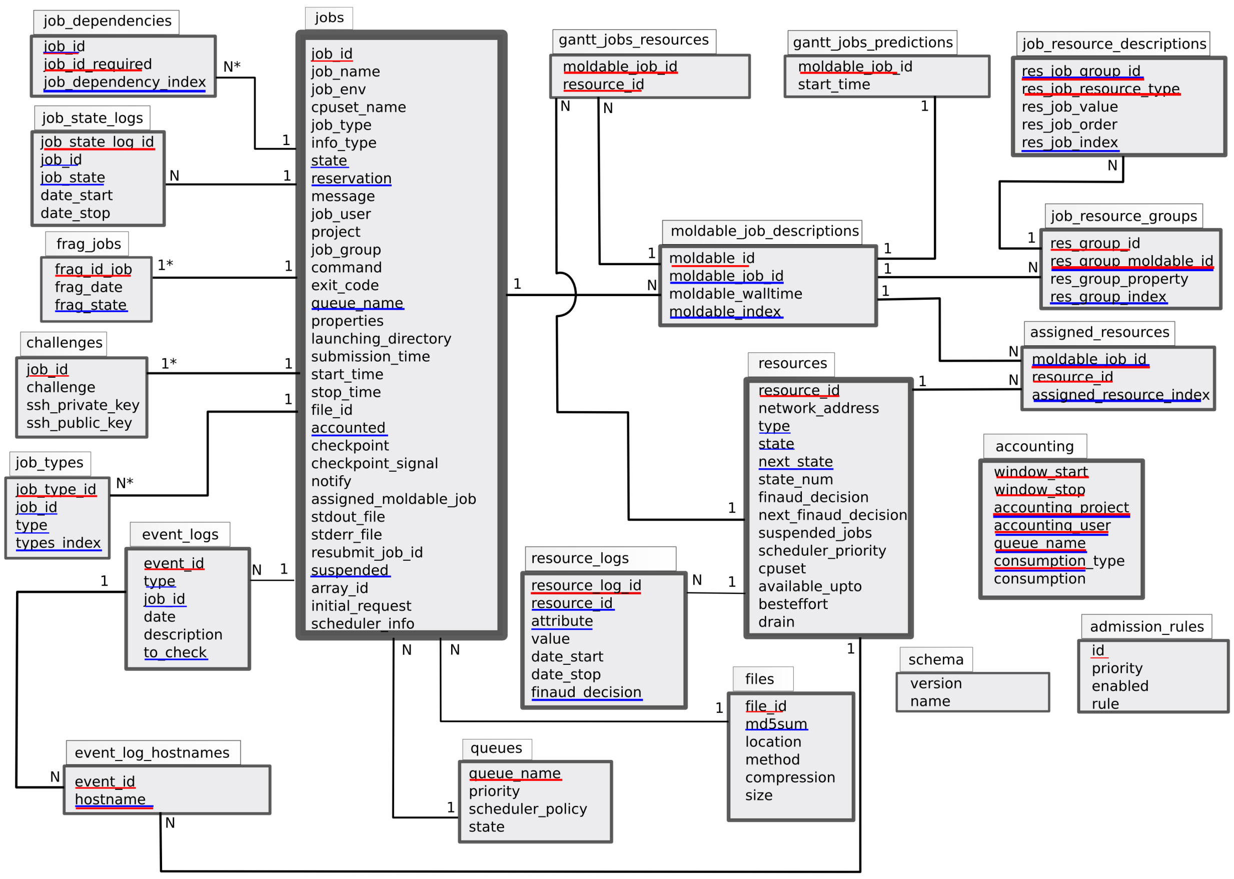


Fig. 1: Database scheme (red lines seem PRIMARY KEY, blue lines seem INDEX)

Primary key window_start, window_stop, accounting_user, queue_name, accounting_project, consumption_type

Index fields window_start, window_stop, accounting_user, queue_name, accounting_project, consumption_type

This table is a summary of the consumption for each user on each queue. This increases the speed of queries about user consumptions and statistic generation.

Data are inserted through the command *oaraccounting* (when a job is treated the field *accounted* in table jobs is passed into "YES"). So it is possible to regenerate this table completely in this way :

- Delete all data of the table:

```
DELETE FROM accounting;
```

- Set the field *accounted* in the table jobs to "NO" for each row:

```
UPDATE jobs SET accounted = "NO";
```

- Run the *oaraccounting* command.

You can change the amount of time for each window : edit the oar configuration file and change the value of the tag *ACCOUNTING_WINDOW*.

2.8.2 schema

Fields	Types	Descriptions
version	VARCHAR(255)	database schema version number
name	VARCHAR(255)	optional name

This table is used to store the version of the database schema.

So the oar-database command be used to automatically upgrade the schema from any version with:

```
oar-database --setup
```

2.8.3 admission_rules

Fields	Types	Descriptions
id	INT UNSIGNED	id number
rule	TEXT	rule written in Perl applied when a job is going to be registered

Primary key id

Index fields *None*

You can use these rules to change some values of some properties when a job is submitted. So each admission rule is executed in the order of the id field and it can set several variables. If one of them exits then the others will not be evaluated and oarsub returns an error.

The rules can be added with the following command:

```
oaradmissionrules -n
```

Some examples are better than a long description:

- Specify the default value for queue parameter

```
if (not defined($queue_name)) {
    $queue_name="default";
}
```

- Avoid users except oar to go in the admin queue

```
if (($queue_name eq "admin") && ($user ne "oar")) {
  die("[ADMISSION RULE] Only oar user can submit jobs in the admin queue\n");
}
```

- Restrict the maximum of the walltime for interactive jobs

```
my $max_walltime = OAR::IO::sql_to_duration("12:00:00");
if ($jobType eq "INTERACTIVE"){
  foreach my $mold (@{$ref_resource_list}){
    if (
      (defined($mold->[1])) and
      ($max_walltime < $mold->[1])
    ){
      print("[ADMISSION RULE] Walltime to big for an INTERACTIVE job so it is set␣
↪to $max_walltime.\n");
      $mold->[1] = $max_walltime;
    }
  }
}
```

- Specify the default walltime

```
my $default_wall = OAR::IO::sql_to_duration("2:00:00");
foreach my $mold (@{$ref_resource_list}){
  if (!defined($mold->[1])){
    print("[ADMISSION RULE] Set default walltime to $default_wall.\n");
    $mold->[1] = $default_wall;
  }
}
```

- How to perform actions if the user name is in a file

```
open(FILE, "/tmp/users.txt");
while (($queue_name ne "admin") and ($_ = <FILE>)){
  if ($_ =~ m/^\\s*$user\\s*$/m){
    print("[ADMISSION RULE] Change assigned queue into admin\n");
    $queue_name = "admin";
  }
}
close(FILE);
```

- How to automatically add a job type depending of the walltime and an estimation of the number of resources of the job

```
foreach my $e (estimate_job_nb_resources($dbh_ro, $ref_resource_list,
↪$jobproperties)){
  #print("AREA: $e->{nbresources} x $e->{walltime} = ".$e->{nbresources} * $e->
↪{walltime}."\n");
  if ($e->{nbresources} * $e->{walltime} > 24*3600*1){
    print("[ADMISSION RULE] Your job is of the 'big' type\n");
```

(continues on next page)

(continued from previous page)

```
    push(@{$type_list},"big");
    last;
  }
}
```

You can print all the admission rules with:

```
oaradmissionrules -S -f
```

2.8.4 event_logs

Fields	Types	Descriptions
event_id	INT UNSIGNED	event identifier
type	VARCHAR(50)	event type
job_id	INT UNSIGNED	job related of the event
date	INT UNSIGNED	event date
descrip-tion	VARCHAR(255)	textual description of the event
to_check	ENUM('YES', 'NO')	specify if the module *NodeChangeState* must check this event to Suspect or not some nodes

Primary key event_id

Index fields type, to_check

The different event types are:

- "PING_CHECKER_NODE_SUSPECTED" : the system detected via the module "finaud" that a node is not responding.
- "PROLOGUE_ERROR" : an error occurred during the execution of the job prologue (exit code != 0).
- "EPILOGUE_ERROR" : an error occurred during the execution of the job epilogue (exit code != 0).
- "CANNOT_CREATE_TMP_DIRECTORY" : OAR cannot create the directory where all information files will be stored.
- "CAN_NOT_WRITE_NODE_FILE" : the system was not able to write file which had to contain the node list on the first node (*/tmp/OAR_job_id*).
- "CAN_NOT_WRITE_PID_FILE" : the system was not able to write the file which had to contain the pid of oarexec process on the first node (*/tmp/pid_of_oarexec_for_job_id*).
- "USER_SHELL" : the system was not able to get informations about the user shell on the first node.
- "EXIT_VALUE_OAREXEC" : the oarexec process terminated with an unknown exit code.
- "SEND_KILL_JOB" : signal that OAR has transmitted a kill signal to the oarexec of the specified job.
- "LEON_KILL_BIPBIP_TIMEOUT" : Leon module has detected that something wrong occurred during the kill of a job and so kill the local *bipbip* process.
- "EXTERMINATE_JOB" : Leon module has detected that something wrong occurred during the kill of a job and so clean the database and terminate the job artificially.
- "WORKING_DIRECTORY" : the directory from which the job was submitted does not exist on the node assigned by the system.
- "OUTPUT_FILES" : OAR cannot write the output files (stdout and stderr) in the working directory.

- "CANNOT_NOTIFY_OARSUB" : OAR cannot notify the *oarsub* process for an interactive job (maybe the user has killed this process).
- "WALLTIME" : the job has reached its walltime.
- "SCHEDULER_REDUCE_NB_NODES_FOR_RESERVATION" : this means that there is not enough nodes for the reservation and so the scheduler do the best and gives less nodes than the user wanted (this occurres when nodes become Suspected or Absent).
- "BESTEFFORT_KILL" : the job is of the type *besteffort* and was killed because a normal job wanted the nodes.
- "FRAG_JOB_REQUEST" : someone wants to delete a job.
- "CHECKPOINT" : the checkpoint signal was sent to the job.
- "CHECKPOINT_ERROR" : OAR cannot send the signal to the job.
- "CHECKPOINT_SUCCESS" : system has sent the signal correctly.
- "SERVER_EPILOGUE_TIMEOUT" : epilogue server script has time outed.
- "SERVER_EPILOGUE_EXIT_CODE_ERROR" : epilogue server script did not return 0.
- "SERVER_EPILOGUE_ERROR" : cannot find epilogue server script file.
- "SERVER_PROLOGUE_TIMEOUT" : prologue server script has time outed.
- "SERVER_PROLOGUE_EXIT_CODE_ERROR" : prologue server script did not return 0.
- "SERVER_PROLOGUE_ERROR" : cannot find prologue server script file.
- "CPUSET_CLEAN_ERROR" : OAR cannot clean correctly cpuset files for a job on the remote node.
- "MAIL_NOTIFICATION_ERROR" : a mail cannot be sent.
- "USER_MAIL_NOTIFICATION" : user mail notification cannot be performed.
- "USER_EXEC_NOTIFICATION_ERROR" : user script execution notification cannot be performed.
- "BIPBIP_BAD_JOBID" : error when retrieving informations about a running job.
- "BIPBIP_CHALLENGE" : OAR is configured to detach jobs when they are launched on compute nodes and the job return a bad challenge number.
- "RESUBMIT_JOB_AUTOMATICALLY" : the job was automatically resubmitted.
- "WALLTIME" : the job reached its walltime.
- "REDUCE_RESERVATION_WALLTIME" : the reservation job was shrunk.
- "SSH_TRANSFER_TIMEOUT" : node OAR part script was too long to transfer.
- "BAD_HASHTABLE_DUMP" : OAR transfered a bad hashtable.
- "LAUNCHING_OAREXEC_TIMEOUT" : oarexec was too long to initialize itself.
- "RESERVATION_NO_NODE" : All nodes were detected as bad for the reservation job.

2.8.5 event_log_hostnames

Fields	Types	Descriptions
event_id	INT UNSIGNED	event identifier
hostname	VARCHAR(255)	name of the node where the event has occured

Primary key event_id

Index fields hostname

This table stores hostnames related to events like “PING_CHECKER_NODE_SUSPECTED”.

2.8.6 files

Fields	Types	Descriptions
idFile	INT UNSIGNED	
md5sum	VARCHAR(255)	
location	VARCHAR(255)	
method	VARCHAR(255)	
compression	VARCHAR(255)	
size	INT UNSIGNED	

Primary key idFile

Index fields md5sum

2.8.7 frag_jobs

Fields	Types	Descriptions
frag_id_job	INT UNSIGNED	job id
frag_date	INT UNSIGNED	kill job decision date
frag_state	ENUM(‘LEON’, ‘TIMER_ARMED’ , ‘LEON_EXTERMINATE’, ‘FRAGGED’) DEFAULT ‘LEON’	state to tell Leon what to do

Primary key frag_id_job

Index fields frag_state

What do these states mean:

- “LEON” : the Leon module must try to kill the job and change the state into “TIMER_ARMED”.
- “TIMER_ARMED” : the Sarko module must wait a response from the job during a timeout (default is 60s)
- “LEON_EXTERMINATE” : the Sarko module has decided that the job time outed and asked Leon to clean up the database.
- “FRAGGED” : job is fragged.

2.8.8 gantt_jobs_resources

Fields	Types	Descriptions
moldable_job_id	INT UNSIGNED	moldable job id
resource_id	INT UNSIGNED	resource assigned to the job

Primary key moldable_job_id, resource_id

Index fields *None*

This table specifies which resources are attributed to which jobs.

2.8.9 gantt_jobs_resources_visu

Fields	Types	Descriptions
moldable_job_id	INT UNSIGNED	moldable job id
resource_id	INT UNSIGNED	resource assigned to the job

Primary key moldable_job_id, resource_id

Index fields *None*

This table is the same as *gantt_jobs_resources* and is used by visualisation tools. It is updated atomically (a lock is used).

2.8.10 gantt_jobs_predictions

Fields	Types	Descriptions
moldable_job_id	INT UNSIGNED	job id
start_time	INT UNSIGNED	date when the job is scheduled to start

Primary key moldable_job_id

Index fields *None*

With this table and *gantt_jobs_resources* you can know exactly what are the decisions taken by the schedulers for each waiting jobs.

note The special job id “0” is used to store the scheduling reference date.

2.8.11 gantt_jobs_predictions_visu

Fields	Types	Descriptions
moldable_job_id	INT UNSIGNED	job id
start_time	INT UNSIGNED	date when the job is scheduled to start

Primary key job_id

Index fields *None*

This table is the same as *gantt_jobs_predictions* and is used by visualisation tools. It is made up to date in an atomic action (with a lock).

2.8.12 jobs

Fields	Types
job_id	INT UNSIGNED
array_id	INT
array_index	INT
initial_request	TEXT
job_name	VARCHAR(100)
cpuset_name	VARCHAR(255)
job_type	ENUM(‘INTERACTIVE’, ‘PASSIVE’) DEFAULT ‘PASSIVE’

Table 1 – continued f

Fields	Types
info_type	VARCHAR(255)
state	ENUM('Waiting','Hold', 'toLaunch', 'toError', 'toAckReservation', 'Launching', 'Running' 'Suspended',
reservation	ENUM('None', 'toSchedule', 'Scheduled') DEFAULT 'None'
message	VARCHAR(255)
job_user	VARCHAR(255)
command	TEXT
queue_name	VARCHAR(100)
properties	TEXT
launching_directory	TEXT
submission_time	INT UNSIGNED
start_time	INT UNSIGNED
stop_time	INT UNSIGNED
file_id	INT UNSIGNED
accounted	ENUM("YES", "NO") DEFAULT "NO"
notify	VARCHAR(255)
assigned_moldable_job	INT UNSIGNED
checkpoint	INT UNSIGNED
checkpoint_signal	INT UNSIGNED
stdout_file	TEXT
stderr_file	TEXT
resubmit_job_id	INT UNSIGNED
project	VARCHAR(255)
suspended	ENUM("YES","NO")
job_env	TEXT
exit_code	INT DEFAULT 0
job_group	VARCHAR(255)

Primary key job_id

Index fields state, reservation, queue_name, accounted, suspended

Explications about the "state" field:

- "Waiting" : the job is waiting OAR scheduler decision.
- "Hold" : user or administrator wants to hold the job (*oarhold* command). So it will not be scheduled by the system.
- "toLaunch" : the OAR scheduler has attributed some nodes to the job. So it will be launched.
- "toError" : something wrong occurred and the job is going into the error state.
- "toAckReservation" : the OAR scheduler must say "YES" or "NO" to the waiting *oarsub* command because it requested a reservation.
- "Launching" : OAR has launched the job and will execute the user command on the first node.
- "Running" : the user command is executing on the first node.
- "Suspended" : the job was in Running state and there was a request (*oarhold* with "-r" option) to suspend this job. In this state other jobs can be scheduled on the same resources (these resources has the "suspended_jobs" field to "YES").
- "Finishing" : the user command has terminated and OAR is doing work internally
- "Terminated" : the job has terminated normally.

- "Error" : a problem has occurred.

Explications about the "reservation" field:

- "None" : the job is not a reservation.
- "toSchedule" : the job is a reservation and must be approved by the scheduler.
- "Scheduled" : the job is a reservation and is scheduled by OAR.

2.8.13 job_dependencies

Fields	Types	Descriptions
job_id	INT UNSIGNED	job identifier
job_id_required	INT UNSIGNED	job needed to be completed before launching job_id

Primary key job_id, job_id_required

Index fields job_id, job_id_required

This table is feeded by *oarsub* command with the "-a" option.

2.8.14 moldable_job_descriptions

Fields	Types	Descriptions
moldable_id	INT UNSIGNED	moldable job identifier
moldable_job_id	INT UNSIGNED	corresponding job identifier
moldable_walltime	INT UNSIGNED	instance duration

Primary key moldable_id

Index fields moldable_job_id

A job can be described with several instances. Thus OAR scheduler can choose one of them. For example it can calculate which instance will finish first. So this table stores all instances for all jobs.

2.8.15 job_resource_groups

Fields	Types	Descriptions
res_group_id	INT UNSIGNED	group identifier
res_group_moldable_id	INT UNSIGNED	corresponding moldable job identifier
res_group_property	TEXT	SQL constraint properties

Primary key res_group_id

Index fields res_group_moldable_id

As you can specify job global properties with *oarsub* and the "-p" option, you can do the same thing for each resource groups that you define with the "-l" option.

2.8.16 job_resource_descriptions

Fields	Types	Descriptions
res_job_group_id	INT UNSIGNED	corresponding group identifier
res_job_resource_type	VARCHAR(255)	resource type (name of a field in resources)
res_job_value	INT	wanted resource number
res_job_order	INT UNSIGNED	order of the request

Primary key res_job_group_id, res_job_resource_type, res_job_order

Index fields res_job_group_id

This table store the hierarchical resource description given with *oarsub* and the "-l" option.

2.8.17 job_state_logs

Fields	Types	Descriptions
job_state_log_id	INT UNSIGNED	identifier
job_id	INT UNSIGNED	corresponding job identifier
job_state	ENUM('Waiting', 'Hold', 'toLaunch', 'toError', 'toAckReservation', 'Launching', 'Finishing', 'Running', 'Suspended', 'Resuming', 'Terminated', 'Error')	job state during the interval
date_start	INT UNSIGNED	start date of the interval
date_stop	INT UNSIGNED	end date of the interval

Primary key job_state_log_id

Index fields job_id, job_state

This table keeps informations about state changes of jobs.

2.8.18 job_types

Fields	Types	Descriptions
job_type_id	INT UNSIGNED	identifier
job_id	INT UNSIGNED	corresponding job identifier
type	VARCHAR(255)	job type like "deploy", "timesharing", …
type_index	ENUM('CURRENT', 'LOG')	index field

Primary key job_type_id

Index fields job_id, type

This table stores job types given with the *oarsub* command and "-t" options.

2.8.19 resources

Fields	Types	Descriptions
resource_id	INT UNSIGNED	resource identifier
type	VARCHAR(100) DEFAULT “default”	resource type (used for licence resources for example)
network_address	VARCHAR(100)	node name (used to connect via SSH)
state	ENUM(‘Alive’, ‘Dead’ , ‘Suspected’, ‘Absent’)	resource state
next_state	ENUM(‘UnChanged’, ‘Alive’, ‘Dead’, ‘Absent’, ‘Suspected’) DEFAULT ‘UnChanged’	state for the resource to switch
finaud_decision	ENUM(‘YES’, ‘NO’) DEFAULT ‘NO’	tell if the actual state results in a “finaud” module decision
next_finaud_decision	ENUM(‘YES’, ‘NO’) DEFAULT ‘NO’	tell if the next node state results in a “finaud” module decision
state_num	INT	corresponding state number (useful with the SQL “ORDER” query)
suspended_jobs	ENUM(‘YES’,’NO’)	specify if there is at least one suspended job on the resource
scheduler_priority	INT UNSIGNED	arbitrary number given by the system to select resources with more intelligence
switch	VARCHAR(50)	name of the switch
cpu	INT UNSIGNED	global cluster cpu number
cpuset	INT UNSIGNED	field used with the *JOB_RESOURCE_MANAGER_PROPERTY_DB_FIELD*
besteffort	ENUM(‘YES’,’NO’)	accept or not besteffort jobs
deploy	ENUM(‘YES’,’NO’)	specify if the resource is deployable
expiry_date	INT UNSIGNED	field used for the desktop computing feature
desktop_computing	ENUM(‘YES’,’NO’)	tell if it is a desktop computing resource (with an agent)
last_job_date	INT UNSIGNED	store the date when the resource was used for the last time
available_upto	INT UNSIGNED	used with compute mode features to know if an Absent resource can be switch on

Primary key resource_id

Index fields state, next_state, type, suspended_jobs

State explications:

- “Alive” : the resource is ready to accept a job.
- “Absent” : the oar administrator has decided to pull out the resource. This computer can come back.
- “Suspected” : OAR system has detected a problem on this resource and so has suspected it (you can look in the *event_logs* table to know what has happened). This computer can come back (automatically if this is a “finaud” module decision).
- “Dead” : The oar administrator considers that the resource will not come back and will be removed from the pool.

This table permits to specify different properties for each resources. These can be used with the *oarsub* command (“-p” and “-l” options).

You can add your own properties with *oarproperty* command.

These properties can be updated with the *oarnodesetting* command ("-p" option).

Several properties are added by default:

- switch : you have to register the name of the switch where the node is plugged.
- cpu : this is a unique name given to each cpus. This enables OAR scheduler to distinguish all cpus.
- cpuset : this is the name of the cpu on the node. The Linux kernel sets this to an integer beginning at 0. This field is linked to the configuration tag *JOB_RESOURCE_MANAGER_PROPERTY_DB_FIELD*.

2.8.20 resource_logs

Fields	Types	Descriptions
resource_log_id	INT UNSIGNED	unique id
resource_id	INT UNSIGNED	resource identifier
attribute	VARCHAR(255)	name of corresponding field in resources
value	VARCHAR(255)	value of the field
date_start	INT UNSIGNED	interval start date
date_stop	INT UNSIGNED	interval stop date
finaud_decision	ENUM('YES','NO')	store if this is a system change or a human one

Primary key *None*

Index fields resource_id, attribute

This table permits to keep a trace of every property changes (consequence of the *oarnodesetting* command with the "-p" option).

2.8.21 assigned_resources

Fields	Types	Descriptions
moldable_job_id	INT UNSIGNED	job id
resource_id	INT UNSIGNED	resource assigned to the job

Primary key moldable_job_id, resource_id

Index fields moldable_job_id

This table keeps informations for jobs on which resources they were scheduled.

2.8.22 queues

Fields	Types	Descriptions
queue_name	VARCHAR(100)	queue name
priority	INT UNSIGNED	the scheduling priority
sched-uler_policy	VARCHAR(100)	path of the associated scheduler
state	ENUM('Active', 'notActive') DEFAULT 'Active'	permits to stop the scheduling for a queue

Primary key queue_name

Index fields *None*

This table contains the schedulers executed by the *oar_meta_scheduler* module. Executables are launched one after one in the specified priority.

2.8.23 challenges

Fields	Types	Descriptions
job_id	INT UNSIGNED	job identifier
challenge	VARCHAR(255)	challenge string
ssh_private_key	TEXT DEFAULT NULL	ssh private key given by the user (in grid usage it enables to connect onto all nodes of the job of all clusers with oarsh)
ssh_public_key	TEXT DEFAULT NULL	ssh public key

Primary key job_id

Index fields *None*

This table is used to share a secret between OAR server and oarexec process on computing nodes (avoid a job id being stolen/forged by malicious user).

For security reasons, this table **must not be readable** for a database account given to users who want to access OAR internal informations(like statistics).

2.9 Admin FAQ

2.9.1 Release policy

Since the version 2.2, release numbers are divided into 3 parts:

- The first represents the design and the implementation used.
- The second represents a set of OAR functionalities.
- The third is incremented after bug fixes.

2.9.2 What means the error "Bad configuration option: PermitLocalCommand" when I am using oarsh?

For security reasons, on the latest OpenSSH releases you are able to execute a local command when you are connecting to the remote host and we must deactivate this option because the oarsh wrapper executes the *ssh* command with oar user privileges.

So if you encounter this error message it means that your OpenSSH does not know this option and you have to remove it from the oar.conf. There is a variable named *OARSH_OPENSSH_DEFAULT_OPTIONS* in oar.conf used by oarsh. So you have just to remove the not yet implemented option.

2.9.3 How to manage start/stop of the nodes?

You have to add a script in /etc/init.d which switches resources of the node into the "Alive" or "Absent" state. So when this script is called at boot time, it will change the state into "Alive". And when it is called at halt time, it will change into "Absent".

There are two ways to perform this action:

1. Install OAR "oar-libs" part on all nodes. Thus you will be able to launch the command *oarnodesetting* (be careful to right configure "oar.conf" with database login and password AND to allow network connections on this database). So you can execute:

   ```
   oarnodesetting -s Alive -h node_hostname
       or
   oarnodesetting -s Absent -h node_hostname
   ```

2. You do not want to install anything else on each node. So you have to enable oar user to connect to the server via ssh (for security you can use another SSH key with restrictions on the command that oar can launch with this one). Thus you will have in your init script something like:

   ```
   sudo -u oar ssh oar-server "oarnodesetting -s Alive -h node_hostname"
       or
   sudo -u oar ssh oar-server "oarnodesetting -s Absent -h node_hostname"
   ```

   In this case, further OAR software upgrade will be more painless.

Take a look in "/etc/default/oar-node" for Debian packaging and in "/etc/sysconfig/oar-node" for redhat.

2.9.4 How can I manage scheduling queues?

see *oarnotify*.

2.9.5 How can I handle licence tokens?

There are 2 ways to handle licence tokens:

- by defining licence resources into OAR (like cores).
- by calling external scripts that gives the number of free licences.

Defining licence resources into OAR

This approach is useful when everything is done inside the cluster (no interaction with the outside).

OAR does not manage resources with an empty "network_address". So you can define resources that are not linked with a real node.

So the steps to configure OAR with the possibility to reserve licences (or whatever you want that are other notions):

1. Add a new field in the table *resources* to specify the licence name.

   ```
   oarproperty -a licence -c
   ```

2. Add your licence name resources with *oarnodesetting*:

```
oarnodesetting -a -h "" -p type=mathlab -p licence=l1
oarnodesetting -a -h "" -p type=mathlab -p licence=l2
oarnodesetting -a -h "" -p type=fluent -p licence=l1
...
```

After this configuration, users can perform submissions like

```
oarsub -I -l "/switch=2/nodes=10+{type='mathlab'}/licence=2"
```

So users ask OAR to give them some other resource types but nothing blocks their program to take more licences than they asked. So the users have to really take care to define the right amount of licences that theirs jobs will use.

Calling an external script

This approach is useful when the cluster processes will use the same licence servers than other clusters or other computers. So you can't know in advance when another computer outside the cluster will use the tokens (like the slots for a proprietary software).

So the only way to handle this situation is to tell the OAR scheduler how many tokens are free each times. And so it can try to schedule the job that asked some tokens.

This is not a perfect solution but it works most of the time.

To configure this feature, you have to:

1. Write a script that displays on the STDOUT the number free tokens.
2. Edit /etc/oar/oar.conf on the OAR server and change the value of SCHEDULER_TOKEN_SCRIPTS; for example:

   ```
   SCHEDULER_TOKEN_SCRIPTS="{ fluent => '/usr/local/bin/check_fluent.sh' }"
   ```

Then the users will be able to submit jobs like:

```
oarsub -l nodes=1/core=12 -t token:fluent=12 ./script.sh
```

2.9.6 How can I handle multiple clusters with one OAR?

These are the steps to follow:

1. create a resource property to identify the corresponding cluster (like "cluster"):

   ```
   oarproperty -a cluster
   ```

   (you can see this new property when you use oarnodes)
2. with *oarnodesetting* you have to fill this field for all resources; for example:

   ```
   oarnodesetting -h node42.cluster1.com -p cluster=1
   oarnodesetting -h node43.cluster1.com -p cluster=1
   oarnodesetting -h node2.cluster2.com -p cluster=2
   ...
   ```
3. Then you have to restrict properties for new job type. So an admission rule performs this job (you can insert this new rule with the *oaradmissionrules* command):

```
my $cluster_constraint = 0;
if (grep(/^cluster1$/, @{$type_list})){
    $cluster_constraint = 1;
}elsif (grep(/^cluster2$/, @{$type_list})){
    $cluster_constraint = 2;
}
if ($cluster_constraint > 0){
    if ($jobproperties ne ""){
        $jobproperties = "($jobproperties) AND cluster = $cluster_constraint";
    }else{
        $jobproperties = "cluster = $cluster_constraint";
    }
    print("[ADMISSION RULE] Added automatically cluster resource constraint\n");
}
```

4. Edit the admission rule which checks the right job types and add “cluster1” and “cluster2” in.

So when you will use oarsub to submit a “cluster2” job type only resources with the property “cluster=2” is used. This is the same when you will use the “cluster1” type. For example:

```
oarsub -I -t cluster2
#is equivalent to
oarsub -I -p "cluster = 2"
```

2.9.7 How to configure a more ecological cluster (or how to make some power consumption economies)?

This feature can be performed with the *Dynamic nodes coupling features*.

First you have to make sure that you have a command to wake up your nodes. . For example you can use the ipmitool tool to communicate with the management boards of the computers.

If you want to enable a node to be woke up the next 12 hours:

```
((DATE=$(date +%s)+3600*12))
oarnodesetting -h host_name -p available_upto=$DATE
```

Otherwise you can disable the wake up of nodes (but not the halt) by:

```
oarnodesetting -h host_name -p available_upto=1
```

If you want to disable the halt on a node (but not the wakeup):

```
oarnodesetting -h host_name -p available_upto=2147483647
```

2147483647 = 2^31 - 1 : we take this value as infinite and it is used to disable the halt mechanism.

And if you want to disable the halt and the wakeup:

```
oarnodesetting -h host_name -p available_upto=0
```

Your *SCHEDULER_NODE_MANAGER_WAKE_UP_CMD* must be a script that reads node names and translate them into the right wake up commands.

So with the right OAR and node configurations you can optimize the power consumption of your cluster (and your air conditioning infrastructure) without drawback for the users.

Take a look at your cluster occupation and your electricity bill to know if it could be interesting for you ;-)

2.9.8 How to enable jobs to connect to the frontales from the nodes using oarsh?

First you have to install the node part of OAR on the wanted nodes.

After that you have to register the frontales into the database using oarnodesetting with the "frontal" (for example) type and assigned the desired cpus into the cpuset field; for example:

```
oarnodesetting -a -h frontal1 -p type=frontal -p cpuset=0
oarnodesetting -a -h frontal1 -p type=frontal -p cpuset=1
oarnodesetting -a -h frontal2 -p type=frontal -p cpuset=0
...
```

Thus you will be able to see resources identifier of these resources with oarnodes; try to type:

```
oarnodes --sql "type='frontal'"
```

Then put this type name (here "frontal") into the *oar.conf* file on the OAR server into the tag *SCHEDULER_NODE_MANAGER_WAKE_UP_CMD*.

Notes:

- if one of these resources become "Suspected" then the scheduling will stop.
- you can disable this feature with *oarnodesetting* and put these resources into the "Absent" state.

2.9.9 A job remains in the "Finishing" state, what can I do?

If you have waited more than a couple of minutes (30mn for example) then something wrong occurred (frontal has crashed, out of memory, . . .).

So you are able to turn manually a job into the "Error" state by typing in the OAR install directory with the root user (example with a bash shell):

```
export OARCONFFILE=/etc/oar/oar.conf
perl -e 'use OAR::IO; $db = OAR::IO::connect(); OAR::IO::set_job_state($db,42,"Error")
↪'
```

(Replace 42 by your job identifier)

Since OAR 2.5.3, you can directly use the command:

```
oardel --force-terminate-finishing-job 42
```

2.9.10 How to activate the memory management on nodes ?

OAR job resources manager is reponsible for setting up the nodes of a job. Among all required steps for that setup, one is to configure the cgroups of the job, in particular the memory cgroup if it is enabled.

To enable the memory cgroup on some Linux distributions (e.g. Debian), it is necessary to pass a parameter to the kernel command line (in the boot loader configuration).

For grub on a Debian system, edit /etc/default/grub, and set:

```
GRUB_CMDLINE_LINUX_DEFAULT="cgroup_enable=memory"
```

Once the node is rebooted, check that is indeed appear in

```
cat /proc/cmdline
```

and “memory” should appear in

```
cat /proc/cgroups
```

Then you just need to set in /etc/oar/job_resource_manager.pl (or the file set in oar.conf for the job resource manager):

```
my $ENABLE_MEMCG = "YES";
```

The the memory usage of the job is given by the the files:

```
/dev/oar_cgroups_links/memory/oar/USERNAME_JOBID/memory.max_usage_in_bytes

/dev/oar_cgroups_links/memory/oar/USERNAME_JOBID/memory.limit_in_bytes
```

The first file tells the maximum amount of memory used by all the processes of the job. So if “max_usage_in_bytes > limit_in_bytes” then swap was used.

CHAPTER

THREE

OTHER DOCUMENTATIONS

See the documentation section of OAR website: http://oar.imag.fr/documentation

Note: For information about OAR release versions with changelogs along with errata and known bugs, see https://oar.imag.fr/oar_versions. This page only shows changelogs.

APPENDIX

A

OAR CHANGELOG

A.1 version 2.5.9:

- [scheduler] add the SCHEDULER_RESOURCE_ORDER_ADV_RESERVATIONS option, so that the scheduling of advance reservations is not impacted by the current state of the resources (e.g. nodes in standby, current besteffort jobs)
- [admission rules] add an admission rule to restrict advance reservation inner jobs to use container jobs that are advance reservations as well
- [schedulers] fix issues with the scheduling of a inner job before its container
- [schedulers] waiting inner job in container that vanished are set to error
- [oarexec] add an option to have inner jobs killed along with their container
- [oarexec] do not run inner jobs before their container is already running
- [oarwalltime] make walltime change respect a possibly defined job deadline
- [oarwalltime] add an option to disable the walltime reduction
- [oarwalltime] fix oarwalltime the per queue configuration
- [oarsub/scheduler] fix a bug with the recent Perl max recursion depth limit
- [drawgantt] show the timezone in the dates
- [oarexec] fix oarsub shell termination when the job is killed
- [database] add an index to the resource_log table
- [oar_resource_add] add support for reparing the resource properties
- [oarexec] add support to disable the auto-repair of suspected nodes
- [job_resource_manager/oarsh] add the COMPUTE_THREAD_SIBLINGS option, to let OAR automatically set the HT thread siblings if not set in the resources hierarchy with a thread resource, or in the resource cpuset field
- [job_resource_manager] rework code, support more cgroup subsystems
- [oarsh] add support to let oarsh create a sub cgroup with either a subset of the cpuset or of the devices in the shell opened on the node. See an example of usage with GNU Parallel in the website documentation
- [oarsub] add the OARSUB_NODE_EXEC_FILE configuration to run a custom command on the head node of the job before the job shell
- [oarsub] make oarsub accept the submission of a noop job with no script
- [oarstat] fix JSON/YAML/XML output when no job to display
- [oarstat] oarstat -j can now use the OAR_JOB_ID environment variable

- [oarstat] fix YAML display with the YAML::Syck library

A.2 version 2.5.8:

- [job_resource_manager] manage nvidia gpu with the cgroup devices
- [oarwalltime] add functionality to allow changing the walltime of a running job. See the oarwalltime command and oar.conf
- [scheduler] fix the besteffort + deploy VS adv. reservation case
- [scheduler] add the state=permissive job type, allowing jobs to be scheduled and run (if noop or cosystem as well only) regardless of the aliveness of resources
- [oarsub/scheduler] fix warning "Use of uninitialized value $resource_value"
- [oarsub] fix unknown error message in case of job termination + typos
- [oarnodesetting] do not kill noop jobs using by resources changed to Dead or Absent
- [finaud] fix: make pingchecker run only on resources of type default
- [oar-database] fix the privilegies of oar's read only user in PostgreSQL in new installation. For existing database, the following command apply the fix: *oar-database –fix-ro-user-priv …*
- [api] some improvement in the Apache configuration and tests
- [api] added POST /media/force to overwrite a file
- [finaud] bugfix: make pingchecker run only on resources of type default
- [api] hardening on the syntax of the URIs (should not impact good URIs!)
- [drawgantt-svg] add a mark next to the label of the resources pointed by the mouse
- [drawgantt-svg] fix possible SQL injection with the filters
- [drawgantt-svg] improve the label_display_regex text replacement mechanism
- [drawgantt-svg/oarstat] fix past and current moldable jobs display
- [drawgantt-svg] fix drain display
- [drawgantt-svg] fix nav_filter with only one option
- [oar.conf] update SSH options to the one of OpenSSH 7.6p1
- [oar-database] support –db-is-local (UNIX socket) for MySQL (MariaDB)
- [oar-node] fix warnings with OAR's sshd configuration
- [oar-resource-add] fix the auto-offset option
- [oar-resource-add] add support for creating GPU resources
- [oar-resource-add] add support to handle the CPU and GPU topologies

A.3 version 2.5.7:

This version mainly brings a security fix for the oarsh command. It is highly recommended to upgrade (server, frontend(s) and nodes), since all previous versions of OAR are affected.

- [oarsh] fix a security hole when passing option to OpenSSH. See oar.conf to adapt settings to your setup, if required (OARSH_* variables)
- [oarsh] dropped the mechanism to select whether to use oarsh or fall back to ssh, given a list of hostname patterns
- [oarsub] fix the job-key information of the manual page
- [oarsub] handle cases where trailing spaces were breaking oarsub script directives
- [api] added an example of Apache configuration for the authentication
- [documentation] improve the SSH keys setup explanations for OAR installation

A.4 version 2.5.6:

- [oar.conf] add the SCHEDULER_MIN_TIME_BETWEEN_2_CALLS option
- **[metascheduler] fix a bug with advance reservations when predicted resources** must be recomputed
- **[metascheduler] fix a bug with advance reservations with standby start job** types
  (noop/cosystem/deploy=standby)
- [oar-node init] create /var/run/sshd if needed
- [oarsub] fix several bugs with the array job submission
- [oarstat] allow using Perl's YAML::Syck for a quicker YAML output
- [oarstat] improve performance and information for the –gantt option
- [oarstat] prettier print of job events
- **[oarnodesetting] optimize grouped operations on resources and add a lock** around property changes
- [oaradmissionrules] fix bug: changing a rule priority does not enable it
- [oar_resources_init] fix node read from standard input
- [oarnodecheck] use /var/lib/oar instead of /etc/oar for working files
- [logs] several cosmetic fixes
- [api] add colmet extraction function
- [api] proposed apache configuration now uses a virtual host on port 6668
- [drawgantt-svg] fix the possibly very long delay when zooming
- [drawgantt-svg] add forecast buttons + relative start/stop url arguments
- [drawgantt-svg] rework configuration for the default display
- [drawgantt-svg] allow displaying resources of type != default
- **[drawgantt-svg] improve support for use as a widget in custom HTML pages** (multisite, etc)
- [monika] fix bugs with recent Perl/Perl CGI versions
- [monika] fix harmless bug in configuration
- **[visualization] remove overlib.js (license issue), this breaks the legacy** drawgantt (which is not supported
  anymore)
- [misc] remove some old development codes from sources
- [misc] fix inconsistent copyrights and licenses

- [doc] update the installation documentation

A.5 version 2.5.5:

- [iolib] fix deadlock with TRUNCATE in postgresql
- **[almighty] add SCHEDULER_MIN_TIME_BETWEEN_2_CALLS:** the scheduler is launched at max every t seconds (t=5 by default), this avoids the scheduler to cause starvation with regard to the other modules
- [scheduler] fix some memory leaks.
- **[scheduler] add a cache to the resources tree computation: improve** the scheduler speed by reducing the number of SQL queries.
- [scheduler] backport the expire/postpone/deadline job types.
- [scheduler] rename the placeholder job types: placeholder/allowed.
- [scheduler] fix timesharing (adv reservation and *_placeholder schedulers).
- **[scheduler] allows noop/cosystem/deploy jobs to start on resources in** standby, no wake-up is triggered (requires activating energy saving).
- [oarsub] use jobkey (-k) if the OAR_JOB_KEY_FILE env variable is set.
- [oarstat] fix accounting display
- [oar_resources_init] fix HyperThreading bug + improve CLI
- **[oar_resources_add] make HyperThreading optional + fix long options + make** nicer warning outputs for auto-offset
- [admission rules] rewrite the job type check rule
- [admission rules] fix oaradmissionrules bug with MySQL when modifying a rule
- [oar-node] fix pid in init script.
- [api] some optimizations + rework authentication configuration (apache).
- [api][drawgantt-svg][monika] fix apache config (apache 2.4).
- [drawgantt-svg] new version with aggreation of resources and more.
- [monika] add thread to the hidden properties.
- [api] fastcgi config now using suexec
- [api] now using apache environment variables when headers are not available
- **[api] optimization of /jobs query response time (especially efficient for** mysql based installations)
- [api] security fix: HTML outputs which did not break on errors

A.6 version 2.5.4:

- **[api] Implemented GET /resources/<property>/jobs to get jobs running on** resources, grouped by a given property.
- **[api] Implemented HTTP_X_API_PATH_PREFIX header variable to prefix all** returned URIs.
- [api] Added GET /jobs/<id>/details support.

- [api] Implemented the ability to get a set of jobs at once with GET /jobs?ids=<id1>:<id2>:<id3>:. . .
- [api] BUGFIX: stderr and stdout where reversed.
- [api] BUGFIX: memory leak in the API when used with FastCGI.
- [api] Rewritten/commented apache config file.
- [kamelot] BUGFIX: fix hierarchies manipulation (remove toplevel resource).
- [accounting] Fixed a memory leak and a rare case of bad consumption count.
- **[oar.conf] Replace the MAX_CONCURRENT_JOB_TERMINATIONS option by** MAX_CONCURRENT_JOBS_STARTING_OR_TERMINATING
- **[almighty] Rewrote the handling of starting and finishing jobs: limit** bipbip processes to MAX_CONCURRENT_JOBS_STARTING_OR_TERMINATING to avoid overloading the server.
- **[oarexec] Introduced BASH_ENV=~oar/.batch_job_bashrc for batch jobs** Batch jobs with bash shell have some difficulties to source the right bash scripts when launching. Now we set BASH_ENV=~oar/.batch_job_bashrc before launching the user bash process so we can handle which script must be sourced. By default we source ~/.bashrc.
- [commands] Exit immediately on wrong arguments.
- **[oarsh] Propagate OAR shell environment variables:** The users have access to the same OAR environment variables when connecting on all the job nodes with oarsh
- [job_uid] Removed job uid feature (not used).
- [job_resource_manager] Use fstrim (for SSD) when cleaning files.
- **[deploy] Do not check the nodes when ACTIVATE_PINGCHECKER_AT_JOB_END is on** and the job is of the deploy type (bug #17128).
- **[judas] Disabled sending log by email on errors as this could generate too** many mails.
- **[noop] Added the 'noop' job type. If specified, nothing is done on computing** nodes. The job just reserves the resources for the specified walltime.
- **[quotas] Added the possibility to make quotas on:**
  - the number of used resources
  - the number of running jobs
  - the result of resources X hours of running jobs
- [runner] Added runner bipbip processes in the bipbip_laucher in Almighty.
- **[database] Replaced field "maintenance" by "drain".** The administrator can disable resources without killing current jobs by:

  ```
  oarnodesetting -h n12 -p drain=YES
  ```

  or:

  ```
  oarnodesetting --drain -h n12
  ```

  WARNING any admission rule using the "maintenance" keyword must be adapted to use the "drain" keyword.
- [oar_resources_init] Added support for SMT (hyperthreading)
- **[cpuset] The cpuset resources filed is now a varchar.** It is now possible to specify several cpu id in the cpuset field as needed in some case where SMT is enabled on nodes, e.g.:

```
1+4+8
```

- **[oarsub] Added a filter for notifications**

  It now is possible to specify which TAGs must trigger motifications:: oarsub –notify "[END,ERROR]mail:name@domain.com" -I

- **[admission rules] Added priority to rules that allows to manage more easily** the rules execution order.
- **[admission rules] Added a enable/disable flag to rules to allow activating** or deactivating rules without having to comment the code.
- **[oaradmin] The oaradmin rules command is now disabled since it does not** handle priority and enable flags.
- [oaradmin] The oaradmin conf command is disabled.
- **[oar_resources_add] Added the oar_resources_add command to help adding** resources and replace the oaradmin resources command.
- **[oaradmissionrules] oaradminssionrules is a new command to manage the** oaradmission rules.
- [oarnodesetting] Removed dependnency to oarnodes.
- [drawgantt-svg] Various bugfixes and improvements
- **[metasched] If a besteffort job has a checkpoint duration defined** (oarsub –checkpoint) then OAR tries to checkpoint it before killing it. It is possible to define a limit of the checkpoint duration with an admission rule ($checkpoint variable).
- [drawgantt] Drawgantt is not now deprecated (and not shipped with packages)
- [misc] OAR packaged components do not require Ruby anymore.
- [oaraccounting] Fix bug reported in Debian tracker #678976
- [sources] Clean-up some used or unrelevant files/codes
- **[scheduler] change default schedulers to quota** The default scheduler of the queues default, admin and besteffort is now oar_sched_gantt_with_timesharing_and_fairsharing_and_quotas. The configuration file /etc/oar/scheduler_quotas.conf contains no quota enforcement so the behaviour remains the same as before.

A.7 version 2.5.3:

- Add the "Name" field on the main Monika page. This is easier for the users to find there jobs.
- Add MAX_CONCURRENT_JOB_TERMINATIONS into the oar.conf ofthe master. This limits the number of concurrent processes launched by the Almighty when the the jobs finish.
- Bug fix in ssh key feature in oarsub.
- Added –compact, -c option to oarstat (compact view or array jobs).
- Improvements of the API: media upload from html forms, listing of files, security fixes, add of new configuration options, listing of the scheduled nodes into jobs, fixed bad reinitialization of the limit parameter, stress_factor, accounting… See OAR-DOCUMENTATION-API-USER for more informations.
- CGROUP: handle cgroup hierarchy already mounted by the OS like in Fedora 18 (by systemd in /sys/fs/cgroup) in job_resource_manager_cgroups.pl.
- Bug fix oar-database: fix the reset function for mysql.

- SVG version of drawgantt: all features are now implemented to replace the legacy drawgantt. Both can be installed.
- Bug fix schedulers: rewrite schedulers with placeholders.
- Rework default admission rules.
- Add support to the oar_resource_init command to generate resources with a “thread” property (useful if Hyper-Threading is activated/used on nodes).
- Fix stdout/stderr bug: check the allowed characters in the path given by the users.
- Fix: the user shell (bash) didn’t source /etc/bash.bashrc in batch jobs.
- Add quota which limits the number of used resources at a time depending of the job attributes: queue, project, types, user (available with the scheduler “oar_sched_gantt_with_timesharing_and_fairsharing_and_quotas”).
- Add comments in user job STDERR files to know if a job was killed or checkpointed.
- Add the variable $jobproperties_applied_after_validation. It can be used in an admission rule to add a constraint after the validation of the job. Ex:

  > $jobproperties_applied_after_validation = “maintenance=’off’”;

  So, even if all the ressources have “maintenance=’on’”, the new jobs will be accepted but not scheduled now.
- Add the oardel option –force-terminate-finishing-job: to use when a job is stuck in the Finishing state.
- Bug #15911: Energy saving now waits SCHEDULER_NODE_MANAGER_IDLE_TIME for nodes that have been woken up, even if they didn’t run any job.
- Simplify job dependencies: do not check the exit code of the jobs in dependencies.
- Admission rules: add the “estimate_job_nb_resources” function that is useful to know the number of resources that will be used by a job.
- oarstat: add another output format that can be used by using “–format 2” or by setting “OARSTAT_DEFAULT_OUTPUT_FORMAT=2” in oar.conf.
- oarsub: Add the capability to use the tag %jobname% in the STDOUT (-O) and/or STDERR (-E) filenames (like %jobid%).
- bug #14935: fix timesharing jobs within a container issue
- add schedulers with the placeholder feature.

A.8 version 2.5.2:

- Bugfix: /var/lib/oar/.bash_oar was empty due to an error in the common setup script.
- Bugfix: the PINGCHECKER_COMMAND in oar.conf depends now on %%OARDIR%%.
- **Bug #13939: the job_resource_manager.pl and job_resource_manager_cgroups.pl** now deletes the user

  files in /tmp, /var/tmp and /dev/shm at the end of the jobs.
- Bugfix: in oardodo.c, the preprocessed variables was not defined correclty.
- Finaud: fix race condition when there was a PINGCHECKER error jsut before another problem. The node became Alive again when the PINGCHECKER said OK BUT there was another error to resolve.
- Bugfix: The feature CHECK_NODES_WITH_RUNNING_JOB=yes never worked before.
- Speedup monika (X5).
- Monika: Add the conf max_cores_per_line to have several lines if the number of cores are too big.

- **Minor changes into API:**
    - added cmd_output into POST /jobs.
- API: Added GET /select_all?query=<query> (read only mode).
- Add the field "array_index" into the jobs table. So that resubmit a job from an array will have the right array_index anvironment variable.
- oarstat: order the output by job_id.
- Speedup oarnodes.
- Fix a spelling error in the oaradmin manpage.
- Bugfix #14122 : the oar-node init.d script wasn't executing start_oar_node/stop_oar_node during the 'restart' action.
- Allow the dash character into the –notify "exec:. . . " oarsub option.
- **Remove some old stuffs from the tarball:**
    - visualization_interfaces/{tgoar,accounting,poar};
    - scheduler/moldable;
    - pbs-oar-lib.
- Fix some licence issues.

A.9 version 2.5.1:

- Sources directories reorganized
- New "Phoenix" tool to try to reboot automatically broken nodes (to setup into /etc/oar/oar_phoenix.pl)
- New (experimental!) scheduler written in Ocaml
- Cpusets are activated by default
- Bugfix #11065: oar_resource_init fix (add a space)
- Bug 10999: memory leak into Hulot when used with postgresql. The leak has been minimized, but it is still there (DBD::Pg bug)
- Almighty cleans ipcs used by oar on exit
- Bugfix #10641 and #10999 : Hulot is automatically and periodically restarted
- Feature request #10565: add the possibility to check the aliveness of the nodes of a job at the end of this one (pingchecker)
- REST API heavily updated: new data structures with paginated results, desktop computing functions, rspec tests, oaradmin resources management, admission rules edition, relative/absolutes uris fixed
- New ruby desktop computing agent using REST API (experimental)
- Experimental testsuite
- Poar: web portal using the REST API (experimental)
- Oaradmin YAML export support for resources creation (for the REST API)
- Bugfix #10567: enabling to bypass window mechanism of hulot.
- Bugfix #10568: Wake up timeout changing with the number of nodes

- Add in oar.conf the tag "RUNNER_SLIDING_WINDOW_SIZE": it allows the runner to use a sliding window to launch the bipbip processes if "DETACH_JOB_FROM_SERVER=1". This feature avoids the overload of the server if plenty of jobs have to be launched at the same time.
- Fix problem when deleting a job in the Suspended state (oarexec was stopped by a SIGSTOP so it was not able to handle the delete operation)
- Make the USER_SIGNAL feature of oardel multi job independant and remove the temporary file at the end of the job
- **Monika: display if the job is of timesharing type or not** add in the job listing the initial_request (is there a reason to not display it?)
- **IoLib: update scheduler_priority resources property for timesharing jobs.** So the scheduler will be able to avoid to launch every timesharing jobs on the same resources (they can be dispatched)
- OAREXEC: unmask SIGHUP and SIGPIPE for user script
- node_change_state: do not Suspect the first node of a job which was EXTERMINATED by Leon if the cpuset feature is configured (let do the job by the cpuset)
- OAREXEC: ESRF detected that sometime oarexec think that he notified the Almighty with it exit code but nothing was seen on the server. So try to resend the exit code until oarexec is killed.
- oar_Tools: add in notify_almighty a check on the print and on the close of the socket connected to Almighty.
- oaraccounting: –sql is now possible into a "oarstat –accounting" query
- Add more logs to the command "oarnodes -e host" when a node turns into Suspected
- Execute user commands with /proc/self/oom_adj to 15. So the first processes that will be killed when there is no more memory available is the user ones. Hence the system will remain up and running and the user job will finished. Drawback: this file can be changed manually by the user so if someone knows a method to do the same thing but only managed by root, we take???
- Bugfix API: quotes where badly escaped into job submission
- Add the possibility to automatically resubmit idempotent job which ends with an exit code of 99: oarsub -t idempotent "sleep 5; exit 99"
- Bugfix API: Some informations where missing into jobs/details, especially the scheduled resources.
- API: added support of "param_file" value for array job submissions. This value is a string representing the content of a parameters file. Sample submission:

  ```
  {"resource":"/cpu=1", "command":"sleep", "param_file":"60\n90\n30"}
  ```

  This submits 3 sleep jobs with differents sleep values.
- Remove any reference to gridlibs and gridapi as these components are obselete
- Add stdout and stderr files of each job in oarstat output.
- API now supports fastcgi (big performance raise!)
- Add "-f" option to oarnodesetting to read hostnames from a file.
- API can get/upload files (GET or POST /media/<file_path>)
- Make "X11 forwarding" working even if the user XAUTHORITY environment variable does not contain ~/.Xauthority (GDM issue).
- Add job_resource_manager_cgroups which handles cpuset + other cgroup features like network packet tagging, IO disk shares, . . .
- Bugfix #13351: now oar_psql_db_init is executed with root privileges

- Bugfix #13434: reservation were not handled correctly with the energy saving feature
- Add cgroups FREEZER feature to the suspend/resume script (better than kill SIGSTOP/SIGCONT). This is doable thanks to the new job_resource_manager_cgroups.
- Implement a new script 'oar-database' to manage the oar database. oar_mysql_init & oar_psql_init are dropped.
- Huge code reorganisation to allow a better packaging and system integration
- Drop the oarsub/oarstat 2.3 version that was kept for compatiblity issues during the 2.4.x branch.
- By default the oar scheduler is now 'oar_sched_gantt_with_timesharing_and_fairsharing' and the following values has been set in oar.conf: SCHEDULER_TIMEOUT to 30, SCHEDULER_NB_PROCESSES to 4 and SCHEDULER_FAIRSHARING_MAX_JOB_PER_USER to 30
- Add a limitation on the number of concurrent bipbip processes on the server (for detached jobs).
- Add IPC cleaning to the job_resource_manager* when there is no other job of the same user on the nodes.
- make better scheduling behaviour for dependency jobs
- API: added missing stop_time into /jobs/details

A.10 version 2.4.4:

- oar_resource_init: bad awk delimiter. There's a space and if the property is the first one then there is not a ','.
- job suspend: oardo does not exist anymore (long long time ago). Replace it with oardodo.
- oarsub: when an admission rule died micheline returns an integer and not an array ref. Now oarsub ends nicely.
- Monika: add a link on each jobid on the node display area.
- sshd_config: with nodes with a lot of core, 10 // connections could be too few

A.11 version 2.4.3:

- Hulot module now has customizable keepalive feature
- Added a hook to launch a healing command when nodes are suspected (activate the SUSPECTED_HEALING_EXEC_FILE variable)
- Bugfix #9995: oaraccouting script doesn't freeze anymore when db is unreachable.
- Bugfix #9990: prevent from inserting jobs with invalid username (like an empty username)
- Oarnodecheck improvements: node is not checked if a job is already running
- New oaradmin option: –auto-offset
- Feature request #10565: add the possibility to check the aliveness of the nodes of a job at the end of this one (pingchecker)

A.12 version 2.4.2:

- New "Hulot" module for intelligent and configurable energy saving
- Bug #9906: fix bad optimization in the gantt lib (so bad scheduling

A.13 version 2.4.1:

- Bug #9038: Security flaw in oarsub –notify option
- Bug #9601: Cosystem jobs are no more killed when a resource is set to Absent
- Fixed some packaging bugs
- API bug fixes in job submission parsing
- Added standby info into *oarnodes -s* and available_upto info into /resources uri of the API
- Bug Grid‘5000 #2687 Fix possible crashes of the scheduler.
- Bug fix: with MySQL DB Finaud suspected resources which are not of the “default” type.
- Signed debian packages (install oar-keyring package)

A.14 version 2.4.0:

- Bug #8791: added CHECK_NODES_WITH_RUNNING_JOB=no to prevent from checking occupied nodes
- Fix bug in oarnodesetting command generated by oar_resources_init (detect_resources)
- Added a –state option to oarstat to only get the status of specified jobs (optimized query, to allow scripting)
- Added a REST API for OAR and OARGRID
- Added JSON support into oarnodes, oarstat and oarsub
- New Makefile adapted to build packages as non-root user
- add the command “oar_resources_init” to easily detect and initialize the whole resources of a cluster.
- “oaradmin version” : now retrieve the most recent database schema number
- Fix rights on the “schema” table in postgresql.
- Bug #7509: fix bug in add_micheline_subjob for array jobs + jobtypes
- Ctrl-C was not working anymore in oarsub. It seems that the signal handler does not handle the previous syntax ($SIG = ‘qdel’)
- Fix bug in oarsh with the “-l” option
- Bug #7487: bad initialisation of the gnatt for the container jobs.
- Scheduler: move the “delete_unnecessary_subtrees” directly into “find_first_hole”. Thus this is possible to query a job like:

  ```
  oarsub -I -l nodes=1/core=1+nodes=4/core=2
  (no hard separation between each group)
  ```

  For the same behaviour as before, you can query: oarsub -I -l {prop=1}/nodes=1/core=1+{prop=2}/nodes=4/core=2
- Bug #7634: test if the resource property value is effectively defined otherwise print a ‘’
- Optional script to take into account cpu/core topology of the nodes at boot time (to activate inside oarnodesetting_ssh)
- Bug #7174: Cleaned default PATH from “./” into oardodo
- Bug #7674: remove the computation of the scheduler_priority field for besteffort jobs from the asynchronous OAR part. Now the value is set when the jobs are turned into toLaunch state and in Error/Terminated.

- Bug #7691: add –array and –array-param-file options parsing into the submitted script. Fix also some parsing errors.
- Bug #7962: enable resource property "cm_availability" to be manipulated by the oarnodesetting command
- **Added the (standby) information to a node state in oarnodes when it's state** is Absent and cm_availability
  != 0
- Changed the name of cm_availability to available_upto which is more relevant
- add a –maintenance option to oarnodesetting that sets the state of a resource to Absent and its available_upto to 0 if maintenance is on and resets previous values if maintenance is off.
- added a –signal option to oardel that allow a user to send a signal to one of his jobs
- added a name field in the schema table that will refer to the OAR version name
- added a table containing scheduler name, script and description
- Bug #8559: Almighty: Moved OAREXEC_XXXX management code out of the queue for immediate action, to prevent potential problems in case of scheduler timeouts.
- oarnodes, oarstat and the REST API are no more making retry connections to the database in case of failure, but exit with an error instead. The retry behavior is left for daemons.
- improved packaging (try to install files in more standard places)
- improved init script for Almighty (into deb and rpm packages)
- fixed performance issue on oarstat (array_id index missing)
- fixed performance issue (job_id index missing in event_log table)
- fixed a performance issue at job submission (optimized a query and added an index on challenges table) decisions).

A.15 version 2.3.5:

- Bug #8139: Drawgantt nil error (Add condition to test the presence of nil value in resources table.)
- Bug #8416: When a the automatic halt/wakeup feature is enabled then there was a problem to determine idle nodes.
- Debug a mis-initialization of the Gantt with running jobs in the metascheduler (concurrency access to PG database)

A.16 version 2.3.4:

- add the command "oar_resources_init" to easily detect and initialize the whole resources of a cluster.
- "oaradmin version" : now retrieve the most recent database schema number
- Fix rights on the "schema" table in postgresql.
- Bug #7509: fix bug in add_micheline_subjob for array jobs + jobtypes
- Ctrl-C was not working anymore in oarsub. It seems that the signal handler does not handle the previous syntax ($SIG = 'qdel')
- Bug #7487: bad initialisation of the gnatt for the container jobs.
- Fix bug in oarsh with the "-l" option

- Bug #7634: test if the resource property value is effectively defined otherwise print a ‘’
- Bug #7674: remove the computation of the scheduler_priority field for besteffort jobs from the asynchronous OAR part. Now the value is set when the jobs are turned into toLaunch state and in Error/Terminated.
- Bug #7691: add –array and –array-param-file options parsing into the submitted script. Fix also some parsing errors.
- Bug #7962: enable resource property “cm_availability” to be manipulated by the oarnodesetting command

A.17 version 2.3.3:

- Fix default admission rules: case unsensitive check for properties used in oarsub
- Add new oaradmin subcommand : oaradmin conf. Useful to edit conf files and keep changes in a Subversion repository.
- Kill correctly each taktuk command children in case of a timeout.
- New feature: array jobs (option –array) (on oarsub, oarstat oardel, oarhold and oarresume) and file-based parametric array jobs (oarsub –array-param-file) /!in this version the DB scheme has changed. If you want to upgrade your installation from a previous 2.3 release then you have to execute in your database one of these SQL script (stop OAR before):

```
mysql:
    DB/mysql_structure_upgrade_2.3.1-2.3.3.sql

postgres:
    DB/pg_structure_upgrade_2.3.1-2.3.3.sql
```

A.18 version 2.3.2:

- Change scheduler timeout implementation to schedule the maximum of jobs.
- Bug #5879: do not show initial_request in oarstat when it is not a job of the user who launched the oarstat command (oar or root).
- Add a –event option to oarnodes and oarstat to display events recorded for a job or node
- Display reserved resources for a validated waiting reservation, with a hint in their state
- Fix oarproperty: property names are lowercase
- Fix OAR_JOB_PROPERTIES_FILE: do not display system properties
- Add a new user command: oarprint which allow to pretty print resource properties of a job
- Debug temporary job UID feature
- Add ‘kill -9’ on subprocesses that reached a timeout (avoid Perl to wait something)
- desktop computing feature is now available again. (ex: oarsub -t desktop_computing date)
- Add versioning feature for admission rules with Subversion

A.19 version 2.3.1:

- Add new oarmonitor command. This will permit to monitor OAR jobs on compute nodes.
- Remove sudo dependency and replace it by the commands "oardo" and "oardodo".
- Add possibility to create a temporary user for each jobs on compute nodes. So you can perform very strong restrictions for each job (ex: bandwidth restrictions with iptable, memory management, . . . everything that can be handled with a user id)
- Debian packaging: Run OAR specific sshd with root privileges (under heavy load, kernel may be more responsive for root processes. . .)
- Remove ALLOWED_NETWORKS tag in oar.conf (added more complexeity than resolving problems)
- /!change database scheme for the field *exit_code* in the table *jobs*. Now *oarstat exit_code* line reflects the right exit code of the user passive job (before, even when the user script was not launched the *exit_code* was 0 which was BAD)
- /!add DB field *initial_request* in the table *jobs* that stores the oarsub line of the user
- Feature Request #4868: Add a parameter to specify what the "nodes" resource is a synomym for. Network_address must be seen as an internal data and not used.
- Scheduler: add timeout for each job == 1/4 of the remaining scheduler timeout.
- Bug #4866: now the whole node is Suspected instead of just the par where there is no job onto. So it is possible to have a job on Suspected nodes.
- Add job walltime (in seconds) in parameter of prologue and epilogue on compute nodes.
- oarnodes does not show system properties anymore.
- New feature: container job type now allows to submit inner jobs for a scheduling within the container job
- Monika refactoring and now in the oar packaging.
- Added a table schema in the db with the field version, reprensenting the version of the db schema.
- Added a field DB_PORT in the oar config file.
- Bug #5518: add right initialization of the job user name.
- Add new oaradmin command. This will permit to create resources and manage admission rules more easily.
- Bug #5692: change source code into a right Perl 5.10 syntax.

A.20 version 2.2.12:

- Bug #5239: fix the bug if there are spaces into job name or project
- Fix the bug in Iolib if DEAD_SWITCH_TIME >0
- Fix a bug in bipbip when calling the cpuset_manager to clean jobs in error
- Bug #5469: fix the bug with reservations and Dead resources
- Bug #5535: checks for reservations made at a same time was wrong.
- New feature: local checks on nodes can be plugged in the oarnodecheck mechanism. Results can be asynchronously checked from the server (taktuk ping checker)

- Add 2 new tables to keep track of the scheduling decisions (gantt_jobs_predictions_log and gantt_jobs_resources_log). This will help debugging scheduling troubles (see SCHEDULER_LOG_DECISIONS in oar.conf)
- Now reservations are scheduled only once (at submission time). Resources allocated to a reservations are definitively set once the validated is done and won't change in next scheduler's pass.
- Fix DrawGantt to not display besteffort jobs in the future which is meaningless.

A.21 version 2.2.11:

- Fix Debian package dependency on a CGI web server.
- Fix little bug: remove notification (scheduled start time) for Interactive reservation.
- Fix bug in reservation: take care of the SCHEDULER_JOB_SECURITY_TIME for reservations to check.
- Fix bug: add a lock around the section which creates and feed the OAR cpuset.
- Taktuk command line API has changed (we need taktuk >= 3.6).
- Fix extra ' in the name of output files when using a job name.
- Bug #4740: open the file in oarsub with user privileges (-S option)
- Bug #4787: check if the remote socket is defined (problem of timing with nmap)
- Feature Request #4874: check system names when renaming properties
- DrawGantt can export charts to be reused to build a global multi-OAR view (e.g. DrawGridGantt).
- Bug #4990: DrawGantt now uses the database localtime as its time reference.

A.22 version 2.2.10:

- Job dependencies: if the required jobs do not have an exit code == 0 and in the state Terminated then the schedulers refuse to schedule this job.
- Add the possibility to disable the halt command on nodes with cm_availability value.
- Enhance oarsub "-S" option (more #OAR parsed).
- Add the possibility to use oarsh without configuring the CPUSETs (can be useful for users that don't want to configure there ssh keys)

A.23 version 2.2.9:

- Bug 4225: Dump only 1 data structure when using -X or -Y or -D.
- Bug fix in Finishing sequence (Suspect right nodes).

A.24 version 2.2.8:

- Bug 4159: remove unneeded Dump print from oarstat.
- Bug 4158: replace XML::Simple module by XML::Dumper one.

- Bug fix for reservation (recalculate the right walltime).
- Print job dependencies in oarstat.

A.25 version 2.2.7:

A.26 version 2.2.11:

- Fix Debian package dependency on a CGI web server.
- Fix little bug: remove notification (scheduled start time) for Interactive reservation.
- Fix bug in reservation: take care of the SCHEDULER_JOB_SECURITY_TIME for reservations to check.
- Fix bug: add a lock around the section which creates and feed the OAR cpuset.
- Taktuk command line API has changed (we need taktuk >= 3.6).
- Fix extra ' in the name of output files when using a job name.
- Bug #4740: open the file in oarsub with user privileges (-S option)
- Bug #4787: check if the remote socket is defined (problem of timing with nmap)
- Feature Request #4874: check system names when renaming properties
- DrawGantt can export charts to be reused to build a global multi-OAR view (e.g. DrawGridGantt).
- Bug #4990: DrawGantt now uses the database localtime as its time reference.

A.27 version 2.2.10:

- Job dependencies: if the required jobs do not have an exit code == 0 and in the state Terminated then the schedulers refuse to schedule this job.
- Add the possibility to disable the halt command on nodes with cm_availability value.
- Enhance oarsub "-S" option (more #OAR parsed).
- Add the possibility to use oarsh without configuring the CPUSETs (can be useful for users that don't want to configure there ssh keys)

A.28 version 2.2.9:

- Bug 4225: Dump only 1 data structure when using -X or -Y or -D.
- Bug fix in Finishing sequence (Suspect right nodes).

A.29 version 2.2.8:

- Bug 4159: remove unneeded Dump print from oarstat.
- Bug 4158: replace XML::Simple module by XML::Dumper one.
- Bug fix for reservation (recalculate the right walltime).

- Print job dependencies in oarstat.

A.30 version 2.2.7:

- Bug 4106: fix oarsh and oarcp issue with some options (erroneous leading space).
- Bug 4125: remove exit_code data when it is not relevant.
- Fix potential bug when changing asynchronously the state of the jobs into "Terminated" or "Error".

A.31 version 2.2.6:

- **Bug fix: job types was not sent to cpuset manager script anymore.** (border effect from bug 4069 resolution)

A.32 version 2.2.5:

- Bug fix: remove user command when oar execute the epilogue script on the nodes.
- Clean debug and mail messages format.
- Remove bad oarsub syntax from oarsub doc.
- Debug xauth path.
- bug 3995: set project correctly when resubmitting a job
- debug 'bash -c' on Fedora
- bug 4069: reservations with CPUSET_ERROR (remove bad hosts and continue with a right integrity in the database)
- bug 4044: fix free resources query for reservation (get the nearest hole from the beginning of the reservation)
- bug 4013: now Dead, Suspected and Absent resources have different colors in drawgantt with a popup on them.

A.33 version 2.2.4:

- Redirect third party commands into oar.log (easier to debug).
- Add user info into drawgantt interface.
- Some bug fixes.

A.34 version 2.2.3:

- Debug prologue and epilogue when oarexec receives a signal.

A.35 version 2.2.2:

- Switch nice value of the user processes into 0 in oarsh_shell (in case of sshd was launched with a different priority).
- debug taktuk zombies in pingchecker and oar_Tools

A.36 version 2.2.1:

- install the "allow_clasic_ssh" feature by default
- debug DB installer

A.37 version 2.2:

- oar_server_proepilogue.pl: can be used for server prologue and epilogue to authorize users to access to nodes that are completely allocated by OAR. If the whole node is assigned then it kills all jobs from the user if all cpus are assigned.
- the same thing can be done with cpuset_manager_PAM.pl as the script used to configure the cpuset. More efficent if cpusets are configured.
- debug cm_availability feature to switch on and off nodes automatically depending on waiting jobs.
- reservations now take care of cm_availability field

A.38 version 2.1.0:

- add "oarcp" command to help the users to copy files using oarsh.
- add sudo configuration to deal with bash. Now oarsub and oarsh have the same behaviour as ssh (the bash configuration files are loaded correctly)
- bug fix in drawgantt (loose jobs after submission of a moldable one)
- add SCHEDULER_RESOURCES_ALWAYS_ASSIGNED_TYPE into oar.conf. Thus admin can add some resources for each jobs (like frontale node)
- add possibility to use taktuk to check the aliveness of the nodes
- %jobid% is now replaced in stdout and stderr file names by the effective job id
- change interface to shu down or wake up nodes automatically (now the node list is read on STDIN)
- add OARSUB_FORCE_JOB_KEY in oar.conf. It says to create a job ssh key by default for each job.
- %jobid% is now replaced in the ssh job key name (oarsub -k . . .).
- add NODE_FILE_DB_FIELD_DISTINCT_VALUES in oar.conf that enables the admin to configure the generated containt of the OAR_NODE_FILE
- change ssh job key oarsub options behaviour
- add options "–reinitialize" and "–delete-before" to the oaraccounting command
- cpuset are now stored in /dev/cpuset/oar

- debian packaging: configure and launch a specific sshd for the user oar
- use a file descriptor to send the node list –> able to handle a very large amount of nodes
- every config files are now in /etc/oar/
- oardel can add a besteffort type to jobs and vis versa

A.39 version 2.0.2:

- add warnings and exit code to oarnodesetting when there is a bad node name or resource number
- change package version
- change default behaviour for the cpuset_manager.pl (more portable)
- enable a user to use the same ssh key for several jobs (at his own risk!)
- add node hostnames in oarstat -f
- add –accounting and -u options in oarstat
- bug fix on index fields in the database (syncro): bug 2020
- bug fix about server pro/epilogue: bug 2022
- change the default output of oarstat. Now it is usable: bug 1875
- remove keys in authorized_keys of oar (on the nodes) that do not correspond to an active cpuset (clean after a reboot)
- reread oar.conf after each database connection tries
- add support for X11 forwarding in oarsub -I and -C
- debug mysql initialization script in debian package
- add a variable in oarsh for the default options of ssh to use (more useful to change if the ssh version installed does not handle one of these options)
- read oar.conf in oarsh (so admin can more easily change options in this script)
- add support for X11 forwarding via oarsh
- change variable for oarsh: OARSH_JOB_ID –> OAR_JOB_ID

A.40 version 2.0.0:

- Now, with the ability to declare any type of resources like licences, VLAN, IP range, computing resources must have the type *default* and a network_address not null.
- Possibility to declare associated resources like licences, IP ranges, … and to reserve them like others.
- Now you can connect to your jobs (not only for reservations).
- Add “cosystem” job type (execute and do nothing for these jobs).
- New scheduler : “oar_sched_gantt_with_timesharing”. You can specify jobs with the type “timesharing” that indicates that this scheduler can launch more than 1 job on a resource at a time. It is possible to restrict this feature with words “user and name”. For example, ‘-t timesharing=user,name’ indicates that only a job from the same user with the same name can be launched in the same time than it.
- Add PostGresSQL support. So there is a choice to make between MySQL and PostgresSQL.

- New approach for the scheduling : administrators have to insert into the databases descriptions about resources and not nodes. Resources have a network address (physical node) and properties. For example, if you have dual-processor, then you can create 2 different resources with the same natwork address but with 2 different processor names.
- The scheduler can now handle resource properties in a hierarchical manner. Thus, for example, you can do "oarsub -l /switch=1/cpu=5" which submit a job on 5 processors on the same switch.
- Add a signal handler in oarexec and propagate this signal to the user process.
- Support '#OAR -p ... ' options in user script.
- **Add in oar.conf:**
  - DB_BASE_PASSWD_RO : for security issues, it is possible to execute request with parts specified by users with a read only account (like "-p" option).
  - OARSUB_DEFAULT_RESOURCES : when nothing is specified with the oarsub command then OAR takes this default resource description.
  - OAREXEC_DEBUG_MODE : turn on or off debug mode in oarexec (create /tmp/oar/oar.log on nodes).
  - FINAUD_FREQUENCY : indicates the frequency when OAR launchs Finaud (search dead nodes).
  - SCHEDULER_TIMEOUT : indicates to the scheduler the amount of time after what it must end itself.
  - SCHEDULER_JOB_SECURITY_TIME : time between each job.
  - DEAD_SWITCH_TIME : after this time Absent and Suspected resources are turned on the Dead state.
  - PROLOGUE_EPILOGUE_TIMEOUT : the possibility to specify a different timeout for prologue and epilogue (PROLOGUE_EPILOGUE_TIMEOUT).
  - PROLOGUE_EXEC_FILE : you can specify the path of the prologue script executed on nodes.
  - EPILOGUE_EXEC_FILE : you can specify the path of the epilogue script executed on nodes.
  - GENERIC_COMMAND : a specific script may be used instead of ping to check aliveness of nodes. The script must return bad nodes on STDERR (1 line for a bad node and it must have exactly the same name that OAR has given in argument of the command).
  - JOBDEL_SOFTWALLTIME : time after a normal frag that the system waits to retry to frag the job.
  - JOBDEL_WALLTIME : time after a normal frag that the system waits before to delete the job arbitrary and suspects nodes.
  - LOG_FILE : specify the path of OAR log file (default : /var/log/oar.log).
- Add wait() in pingchecker to avoid zombies.
- Better code modularization.
- Remove node install part to launch jobs. So it is easier to upgrade from one version to an other (oarnodesetting must already be installed on each nodes if we want to use it).
- Users can specify a method to be notified (mail or script).
- Add cpuset support
- Add prologue and epilogue script to be executed on the OAR server before and after launching a job.
- Add dependancy support between jobs ("-a" option in oarsub).
- In oarsub you can specify the launching directory ("-d" option).

- In oarsub you can specify a job name ("-n" option).
- In oarsub you can specify stdout and stderr file names.
- User can resubmit a job (option "–resubmit" in oarsub).
- It is possible to specify a read only database account and it will be used to evaluate SQL properties given by the user with the oarsub command (more scecure).
- Add possibility to order assigned resources with their properties by the scheduler. So you can privilege some resources than others (SCHEDULER_RESOURCE_ORDER tag in oar.conf file)
- a command can be specified to switch off idle nodes (SCHEDULER_NODE_MANAGER_SLEEP_CMD, SCHEDULER_NODE_MANAGER_IDLE_TIME, SCHEDULER_NODE_MANAGER_SLEEP_TIME in oar.conf)
- a command can be specified to switch on nodes in the Absent state according to the resource property *cm_availability* in the table resources (SCHEDULER_NODE_MANAGER_WAKE_UP_CMD in oar.conf).
- if a job goes in Error state and this is not its fault then OAR will resubmit this one.