OAR Documentation - User Guide

Authors: Capit Nicolas, Emeras Joseph

Address: Laboratoire d'Informatique de Grenoble Bat. ENSIMAG - antenne de Montbonnot ZIRST 51, avenue Jean Kuntzmann 38330 MONTBONNOT SAINT MARTIN

Contact: nicolas.capit@imag.fr, joseph.emeras@imag.fr

Authors: LIG laboratory

Organization: LIG laboratory

Status: Stable

Dedication: For users.

Abstract: OAR is a resource manager (or batch scheduler) for large clusters. By it's functionnalities, it's near of PBS, LSF, CCS and Condor. It's suitable for productive plateforms and research experiments.

BE CAREFULL : THIS DOCUMENTATION IS FOR OAR >= 2.3.0

PDF version : OAR-DOCUMENTATION-USER.pdf

Table of Contents

1 OAR capabilities

Oar is an opensource batch scheduler which provides a simple and flexible exploitation of a cluster.

It manages resources of clusters as a traditional batch scheduler (as PBS / Torque / LSF / SGE). In other words, it doesn't execute your job on the resources but manages them (reservation, acces granting) in order to allow you to connect these resources and use them.

Its design is based on high level tools:

- relational database engine MySQL or PostgreSQL,
- scripting language Perl,
- confinement system mechanism cpuset,
- scalable exploiting tool Taktuk.

It is flexible enough to be suitable for production clusters and research experiments. It currently manages over than 5000 nodes and has executed more than 5 million jobs.

OAR advantages:

- No specific daemon on nodes.
- No dependence on specific computing libraries like MPI. We support all sort of parallel user applications.
- Upgrades are made on the servers, nothing to do on computing nodes.
- CPUSET (2.6 linux kernel) integration which restricts the jobs on assigned resources (also useful to clean completely a job, even parallel jobs).
- All administration tasks are performed with the taktuk command (a large scale remote execution deployment): http://taktuk.gforge.inria.fr/.
- Hierarchical resource requests (handle heterogeneous clusters).
- Gantt scheduling (so you can visualize the internal scheduler decisions).
- Full or partial time-sharing.
- Checkpoint/resubmit.
- Licences servers management support.
- Best effort jobs : if another job wants the same resources then it is deleted automatically (useful to execute programs like *SETI@home*).
- Environment deployment support (Kadeploy): http://kadeploy.imag.fr/.

Other more *common* features:

- Batch and Interactive jobs.
- Admission rules.
- Walltime.
- Multi-schedulers support.
- Multi-queues with priority.
- Backfilling.
- First-Fit Scheduler.
- Reservation.
- Support of moldable tasks.
- Check compute nodes.
- Epilogue/Prologue scripts.
- Support of dynamic nodes.
- Logging/Accounting.
- Suspend/resume jobs.

2 Description of the different commands

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

2.1 *oarstat*

This command prints jobs in execution mode on the terminal.

Options

```
-j, --job                 show informations only for the specified job (eve
-f, --full                show full informations
-s, --state               show only the state of a job (optimized query)
-u, --user                show informations for this user only
-g, --gantt               show job informations between two date-times
-e, --events              show job events
-p, --properties          show job properties
    --accounting          show accounting informations between two dates
    --sql                 restricts display by applying the SQL where claus
                          on the table jobs (ex: "project = 'p1'")
-D, --dumper              print result in DUMPER format
-X, --xml                 print result in XML format
-Y, --yaml                print result in YAML format
    --backward-compatible OAR 1.* version like display
-V, --version             print OAR version number
-h, --help                show this help screen
```

Examples

```
# oarstat
# oarstat -j 42 -f
# oarstat --sql "project = 'p1'"
# oarstat -s -j 42
```

2.2 *oarnodes*

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

Options

```
-a                : shows all resources with their properties
-r                : show only properties of a resource
-s                : shows only resource states
-l                : shows only resource list
--sql "sql where" : Display resources which matches this sql where clause
-D                : formats outputs in Perl Dumper
-X                : formats outputs in XML
-Y                : formats outputs in YAML
```

Examples

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

2.3 *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
```

Options:

```
-I, --interactive             Request an interactive job. Open a login shel
                              on the first node of the reservation instead
                              running a script.
-C, --connect=<job id>        Connect to a running job
-l, --resource=<list>         Set the requested resources for the job.
                              The different parameters are resource propert
                              registered in OAR database, and 'walltime' wh
                              specifies the duration before the job must be
                              automatically terminated if still running.
                              Walltime format is [hour:mn:sec|hour:mn|hour]
                              Ex: nodes=4/cpu=1,walltime=2:00:00
-S, --scanscript              Batch mode only: asks oarsub to scan the give
                              script for OAR directives (#OAR -l ...)
-q, --queue=<queue>           Set the queue to submit the job to
-p, --property="<list>"       Add constraints to properties for the job.
                              (format is a WHERE clause from the SQL syntax
-r, --reservation=<date>      Request a job start time reservation,
                              instead of a submission. The date format is
                              "YYYY-MM-DD HH:MM:SS".
    --checkpoint=<delay>      Enable the checkpointing for the job. A signa
                              is sent DELAY seconds before the walltime on
                              the first processus of the job
    --signal=<#sig>           Specify the signal to use when checkpointing
                              Use signal numbers, default is 12 (SIGUSR2)
-t, --type=<type>             Specify a specific type (deploy, besteffort,
                              cosystem, checkpoint, timesharing)
-d, --directory=<dir>         Specify the directory where OAR will launch t
                              command (default is current directory)
    --project=<txt>           Specify a name of a project the job belongs t
```

```
-n, --name=<txt>                Specify an arbitrary name for the job
-a, --anterior=<job id>         Anterior job that must be terminated to start
                                this new one
    --notify=<txt>              Specify a notification method
                                (mail or command to execute). Ex:
                                    --notify "mail:name\@domain.com"
                                    --notify "exec:/path/to/script args"
    --resubmit=<job id>         Resubmit the given job as a new one
-k, --use-job-key               Activate the job-key mechanism.
-i, --import-job-key-from-file=<file>
                                Import the job-key to use from a files instea
                                of generating a new one.
    --import-job-key-inline=<txt>
                                Import the job-key to use inline instead of
                                generating a new one.
-e  --export-job-key-to-file=<file>
                                Export the job key to a file. Warning: the
                                file will be overwritten if it already exists
                                (the %jobid% pattern is automatically replace
-O  --stdout=<file>             Specify the file that will store the standart
                                output stream of the job.
                                (the %jobid% pattern is automatically replace
-E  --stderr=<file>             Specify the file that will store the standart
                                error stream of the job.
                                (the %jobid% pattern is automatically replace
    --hold                      Set the job state into Hold instead of Waitin
                                so that it is not scheduled (you must run
                                "oarresume" to turn it into the Waiting state
-D, --dumper                    Print result in DUMPER format
-X, --xml                       Print result in XML format
-Y, --yaml                      Print result in YAML format
-h, --help                      Print this help message
-V, --version                   Print OAR version number
```

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:
```

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*.

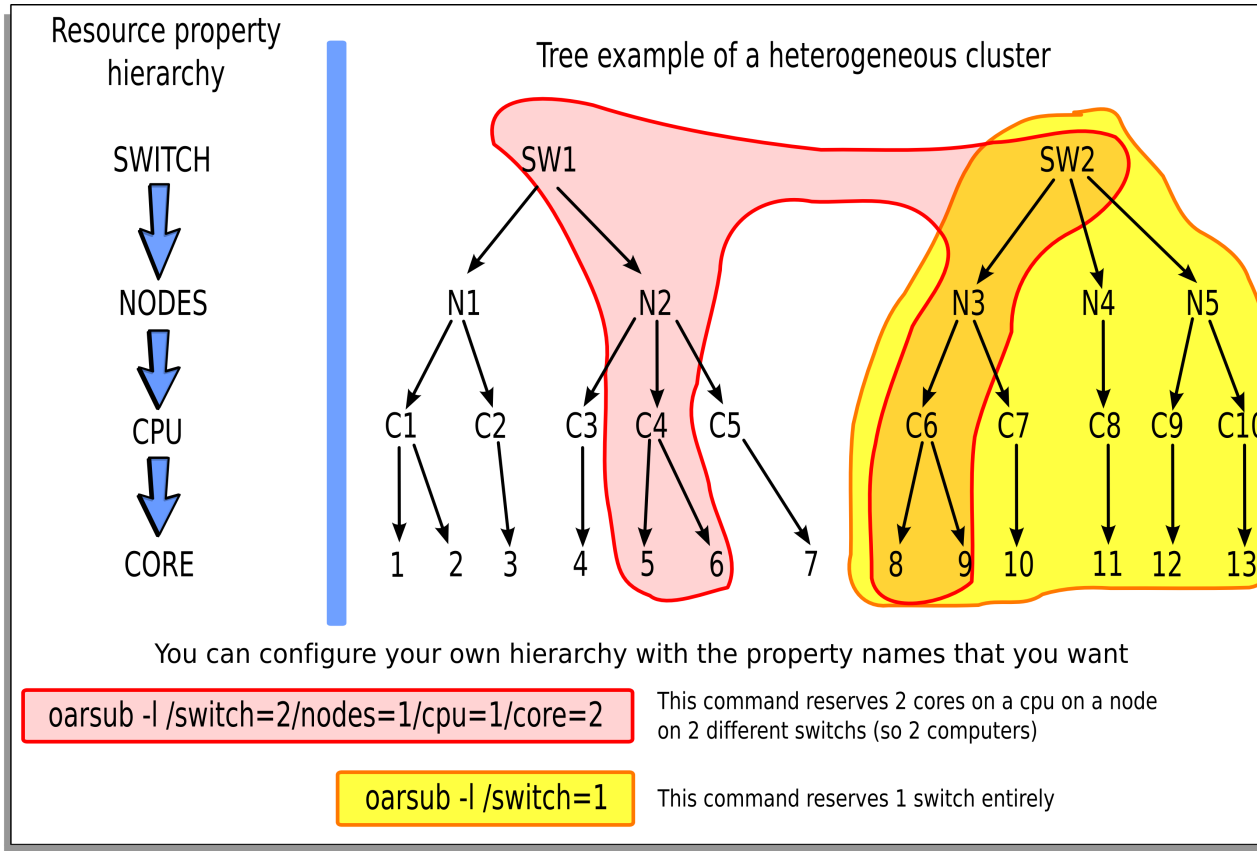

Figure 1: Example of a resource hierarchy and 2 different oarsub commands

hierarchical_resources.svg
Examples

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

(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 /node=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)

2.4 *oardel*

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

Option

```
--sql     : delete/checkpoint jobs which respond to the SQL where clause
            on the table jobs (ex: "project = 'p1'")
-c job_id : send checkpoint signal to the job (signal was
            definedwith "--signal" option in oarsub)
```

Examples

```
# oardel 14 42
```

(delete jobs 14 and 42)

```
# oardel -c 42
```

(send checkpoint signal to the job 42)

2.5 *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. In that way, a **SIGINT** signal is sent to every processes.

Options

```
--sql : hold jobs which respond to the SQL where clause on the table
        jobs (ex: "project = 'p1'")
-r    : Manage not only Waiting jobs but also Running one
        (can suspend the job)
```

2.6 *oarresume*

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

```
--sql : resume jobs which respond to the SQL where clause on the table
        jobs (ex: "project = 'p1'")
```

3 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.

4 Visualisation tools

4.1 Monika

This is a web cgi normally installed on the cluster frontal. This tool connects to the DB, gets relevant information then format data in a html page.

Thus you can have a global view of cluster state and where your jobs are running.

4.2 DrawOARGantt

This is also a web cgi. It creates a Gantt chart which shows job repartition on nodes in the time. It is very useful to see cluster occupation in the past and to know when a job will be launched in the future.

5 Mechanisms

5.1 How does an interactive *oarsub* work?

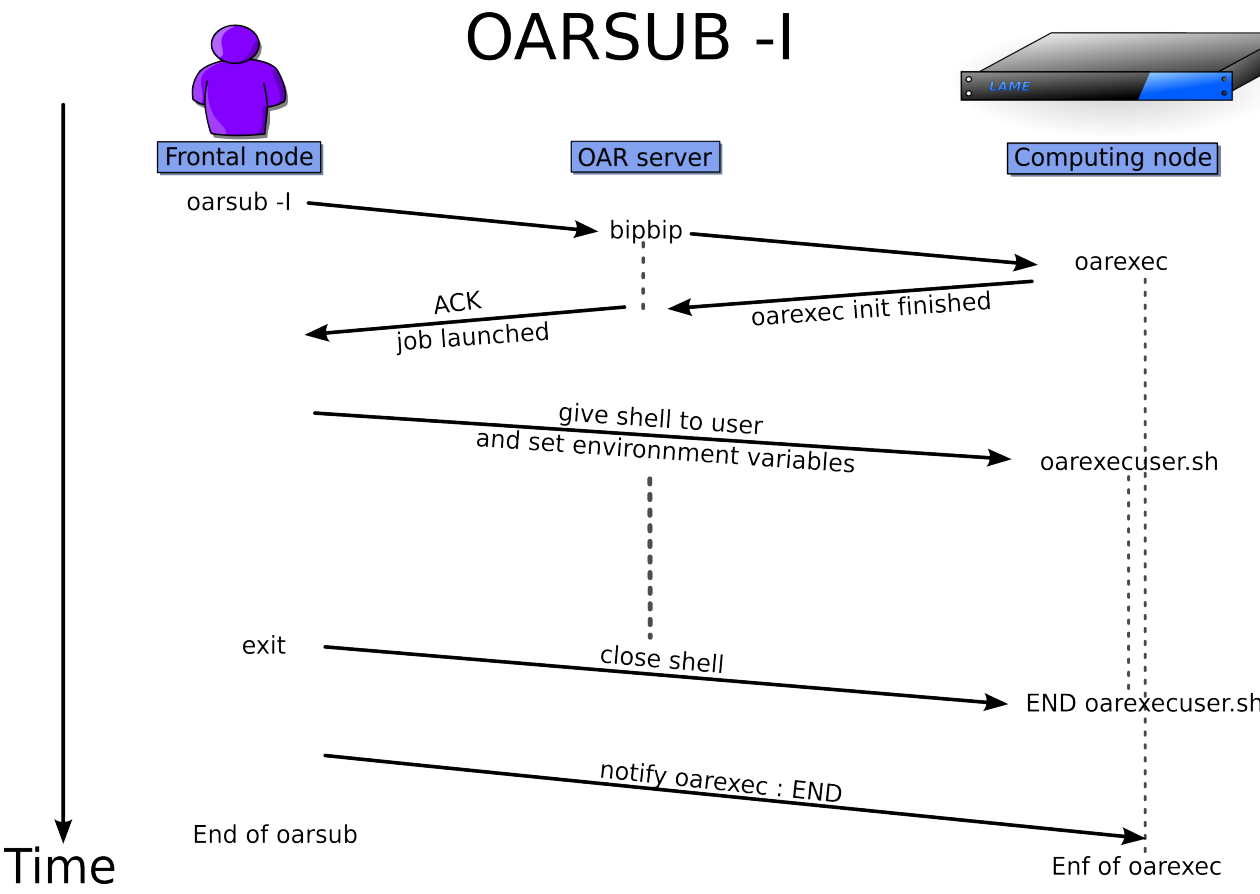

Figure 2: Interactive oarsub decomposition

interactive_oarsub_scheme.svg

5.2 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.

5.3 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.

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.

5.4 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.p
```

5.5 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.

5.6 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 0 then another job is automatically created and scheduled with same behaviours.

5.7 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.

5.8 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.

5.9 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.

5.10 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

5.11 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.

5.12 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.

5.13 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.

You have 4 different ways to share your slot:

1. *timesharing=*,** : This is the default behavior if nothing but timesharing is specified. It indicates that the job can be shared with all users and every job names.
2. *timesharing=user,** : This indicates that the job can be shared only with the same user and every job names.
3. *timesharing=*,job_name* : This indicates that the job can be shared with all users but only one with the same name.
4. *timesharing=user,job_name* : This indicates that the job can be shared only with the same user and one with the same job name.

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

5.14 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.

5.15 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.

5.16 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.

5.17 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.

5.18 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.

6 FAQ - USER

6.1 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.

6.2 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.

6.3 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.

6.4 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 = "nom_du_switch_sc
   -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.

6.5 How can a checkpointable job be resubmitted automatically?

You have to specify that your job is *idempotent*. 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 it will be resubmitted.

6.6 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
```

7 OAR CHANGELOG

7.1 next version

- Bugfix: Fix a regression (only for PostgreSQL) in Drawgantt introduced in the version 2.4.6 (thanks to Yann Genevois).

7.2 version 2.4.7:

- Backport: Debug checkpoint feature with cosystem or deploy jobs.

7.3 version 2.4.6:

- Fix the user variable used in oarsh. When using oarsh from the frontal, the variable OAR_USER was not defined in the environment, and make oarsh unable to read the user private key file.
- Backport: Bugfix #13434: reservation were not handled correctly with the energy saving feature
- Draw-Gantt: Do not display Absent node in the future that are in the stanby "sub-state".
- Bugfix: spelling error (network_addess > network_address)

7.4 version 2.4.5:

- backport: 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)
- backport: 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.

- backport: oar_Tools: add in notify_almighty a check on the print and on the close of the socket connected to Almighty.
- backport: switch to /bin/bash as default (some scripts currently need bash).

7.5 version 2.4.4:

- 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
- 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.
- Bugfix: oaradmin rules edition/add was broken
- Bug #11599: missing pingchecker line into Leon
- Bug #10567: enabling to bypass window mechanism of hulot (Backport from 2.5)
- Bug #10568: Wake up timeout changing with the number of nodes (Backport from 2.5)
- 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

7.6 version 2.4.3:

- Hulot module now has customizable keepalive feature (backport from 2.5)
- Added a hook to launch a healing command when nodes are suspected (backport from 2.5)
- 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)

7.7 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

7.8 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)

7.9 version 2.4.0:

- 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 + job-types
- 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}/noc

- 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).

7.10 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)

7.11 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 + job-types
- 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

7.12 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
```

7.13 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

7.14 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.

7.15 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.

7.16 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.

7.17 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)

7.18 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).

7.19 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.

7.20 version 2.2.7:

7.21 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.

7.22 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)

7.23 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).

7.24 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.

7.25 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".

7.26 version 2.2.6:

- **Bug fix: job types was not sent to cpuset manager script anymore.**
  (border effect from bug 4069 resolution)

7.27 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.

7.28 version 2.2.4:

- Redirect third party commands into oar.log (easier to debug).
- Add user info into drawgantt interface.
- Some bug fixes.

7.29 version 2.2.3:

- Debug prologue and epilogue when oarexec receives a signal.

7.30 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

7.31 version 2.2.1:

- install the "allow_clasic_ssh" feature by default
- debug DB installer

7.32 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

7.33 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

7.34 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

7.35 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.