GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

grid-control

The Swiss Army knife of job submission tools

GridKA School 2009 - September 2nd 2009

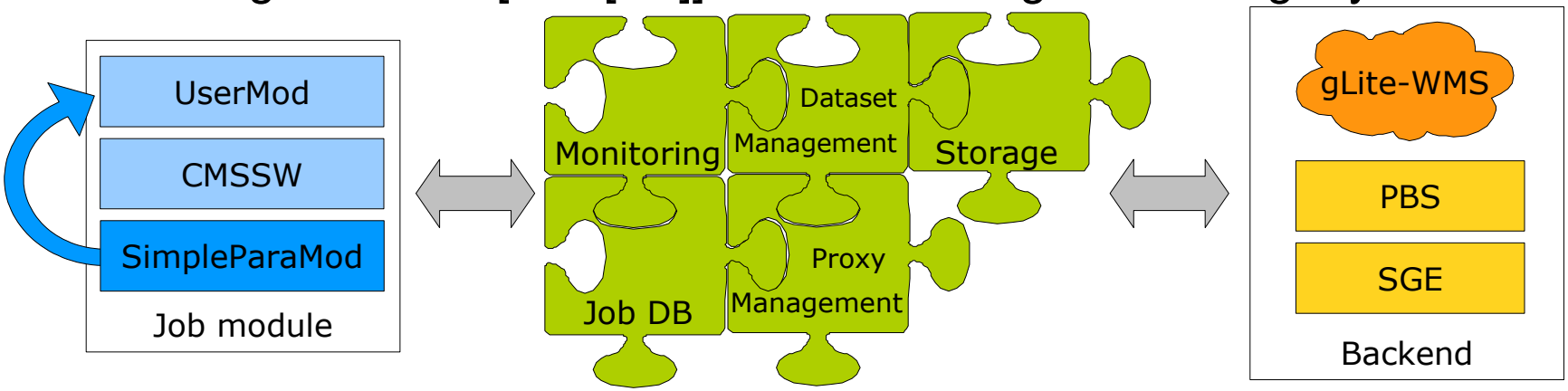
Fred-Markus Stober, Christophe Saout, Andreas Oehler,
Armin Scheurer, Volker Buege, Matthias Wolf, Manuel Zeise,
Oliver Oberst, Klaus Rabbertz, Jasmin Gruschke, Jochen Ott

What is grid-control?

- grid-control is a versatile job submission tool for both local and grid jobs
- Modular, simple, small, extendable, coded in python
- Supports parameterized and dataset based jobs
- Can manage a large number of jobs O(10,000)
- Hosts arbitrary scripts, has support for CMS specific software/databases (code 450 loc, 9%)
- Plays nice with external scripts / interventions
  Everything is human readable / editable
  There is no heavy database / server architecture
- Used since February 2007 in Karlsruhe, CERN, Gent

How to use grid-control – Configuration

- grid-control is setup with an ini-style configuration file. For most options there are sensible default values provided which can be shown with **`--help-conf`**. An existing config file can be reduced to the non-default values with the option **`--help-confmin`**. Own default values can be provided by a “parent” config files specified by **`([global] include)`**.
- Configuration options are given as key = value pairs, where the time values are given as hh[:mm[:ss]] and sizes are given in megabytes.

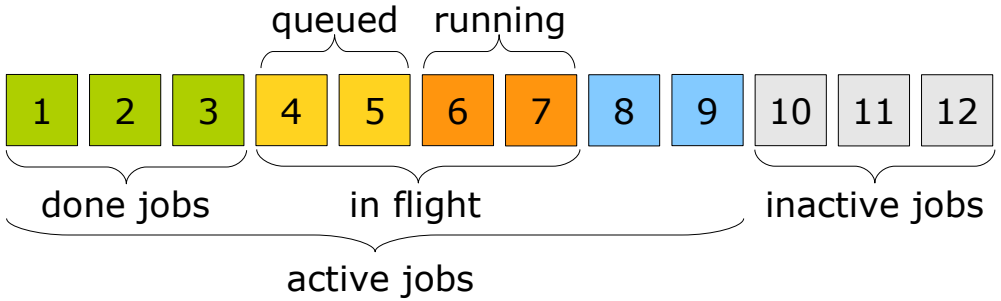


Type of job is described by selecting a certain job module
General cases are covered by UserMod, CMSSW does some additional “comfort” functions. (ROOT module on the horizon.)

- Job submission is controlled by local or grid backend modules

Job configuration

- Job submission is controlled in the **`[jobs]`** section
- With **`jobs`**, you can specify the maximum number of jobs to run, which is truncated to an maximum given by the job module.
- The number of concurrently submitted jobs is given by **`in flight`**



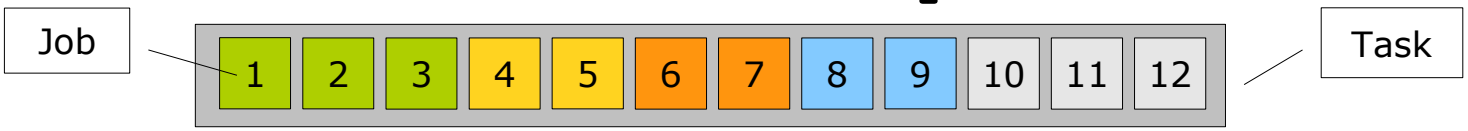
- Each job may request a certain amount of memory with **`memory`** (default: 512, NAF jobs should specify 2000MB!)
- The **`wall time`**, and **`cpu time`** requirements are given by the config options of the same name. In case **`cpu time`** is missing, the wall time is used. The wall time is also used to check proxy lifetime requirements.

Job configuration

- Since there are sometimes jobs hanging in the queue as either waiting or queued, there is the possibility to specify a timeout after which the jobs are canceled and resubmitted **`(queue timeout)`**. This functionality is disabled by default.
- In case there is a problem with a job hanging on some worker node, it is possible to specify a time after which the job is forcibly quit with **`node timeout`**. This allows to get error messages unavailable when running into a wall time timeout.
- By default the jobs are submitted sequentially. To submit jobs in random order, use the option **`shuffle = True.`**
- It is possible to set fixed random seeds to derive random numbers. These random seeds are set with **`(seeds = 32 51 346 234)`**, but can be overwritten by the command line argument **`-S`**. By default 10 seeds are randomly created while initializing the job.

Variables

- The job module provides many different variables to each job. They are accessible in the form of environment variables.
- These variables can be job number specific or task specific. While some variables are predetermined, other variables can only be determined on the worker node itself. An overview of all variables and their values is available with **`–help-vars`**.

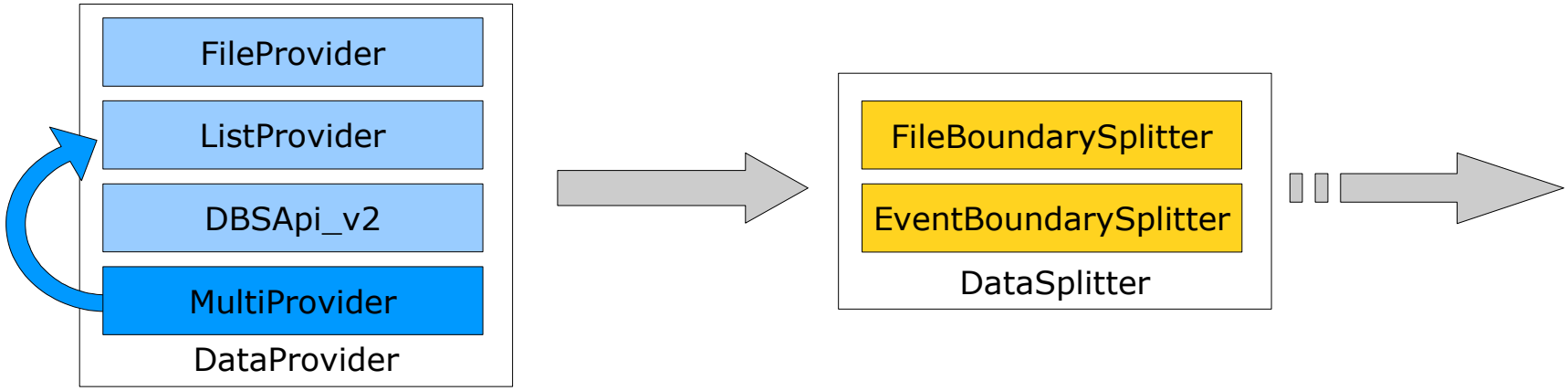


- In “patterns” there is a special variable X, which marks the spot of some replacement operation (shown later).
- With the **`subst files`** statement of the module, it is possible to give a list of files in which variables will be automatically substituted on the worker node.
- All occurrences of either **`__VAR__`** or **`@VAR@`** will be replaced. Lines prefixed with **`@IF@VARNAME@`** or **`@IFEQ@VARNAME@VALUE@`** will only be kept if the variable **`VARNAME`** exists or has a certain value.

UserMod job module

- The UserMod module **`([global] module = CMSSW)`** is used to manage the running of arbitrary scripts / executables.
- They are specified via **`([UserMod] executable)`** and the arguments are given by **`([UserMod] arguments)`**. It is also possible to use variables in these statements.
- Files included into the input and output sandbox can be specified by the **`input files`** and **`output files`** statements of any job module.
- A job may require environment variables which have to be setup (eg. to access CMSSW). Since this setup of variables is often shared between many tasks (eg. run CMSSW or access a standalone MC generator packaged with it), this can be detached from the job script and made generally available. By specifying eg. **`depends = CMSSW`**, the file **`share/env.cmssw.sh`** will be executed at the beginning of the job.

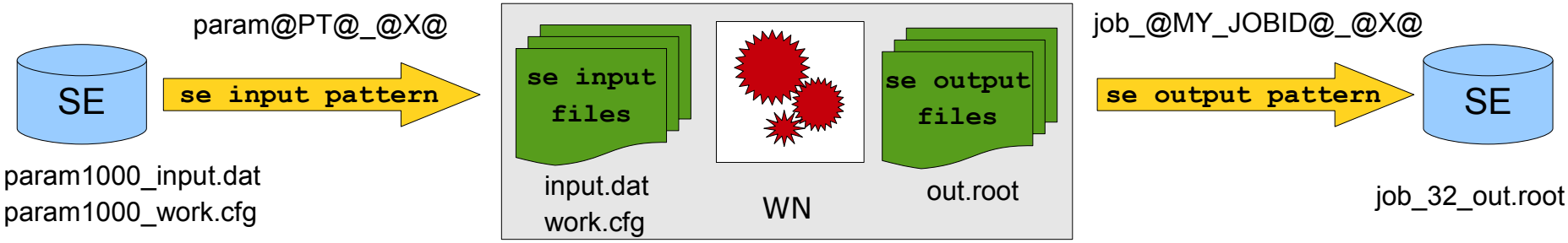
Dataset based jobs



- The location / size of the dataset is given by some DataProvider. The dataset itself is specified by the **`dataset`** statement and the default DataProvider is set with **`dataset provider`** in the job module.
- The dataset is afterwards split up into several jobs according to some DataSplitter which reads certain parameters.
- In the **`dataset`** statement it is possible to give several dataset entries at once and to name them with a nickname for easy identification of the dataset.

Storage Element access

- Supported SE types
  - gsiftp (gsiftp://ekp-lcg-se.physik.uni-karlsruhe.de/wlcg/data/users/cms)
  - SRM (srm://dcache-se-cms.desy.de:8443/pnfs/desy.de/cms/tier2/store/user/)
  - CASTOR (rfio:///castor/cern.ch/user/x/username)
  - local directory access (dir:///absolute/path/to/directory)
  - Setup via **`[storage] se path`**
- Transfer of files both before and after the job
- Output directory has to exist beforehand (except for dir:// access)
- Allows to transform input / output file names according to several variables
- Flag too small output files as failed – **`[storage] se min size`**



Monitoring

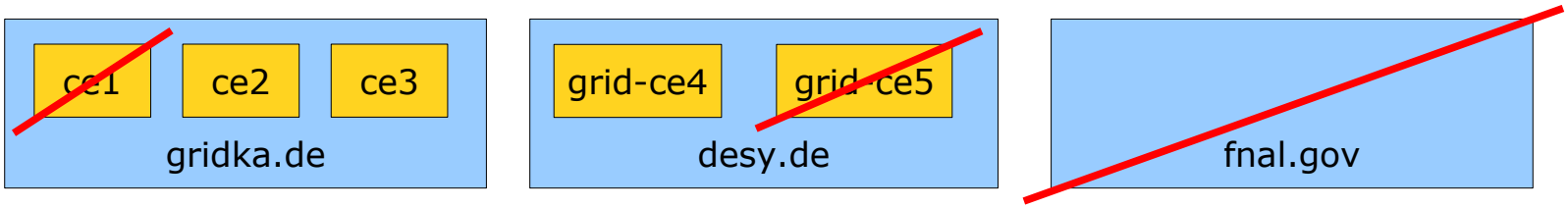
- Event based actions
  - `[events] on submit` – call script upon submission
  - `[events] on status` – call script in case of status change
  - `[events] on output` – call script after job output is retrieved
  - `[events] on finish` – called at the end of the task
  - These scripts are called with several parameters and environment variables set – can be used to move files from SE area to other directories, add output files to dataset file or simply inform the user (Jabber, send email, play fanfare)
- Dashboard Monitoring
  - Update the CMS job monitoring website `[jobs] monitor job = true`
- Intrinsic job reports (site/CE/queue specific) about site failure rates are available from the command line (`-R/-T`, stackable)
- Worker node space usage monitoring
  - Storage usage is monitored and the job is aborted in case certain thresholds are exceeded
  - `[storage] landing zone space left, landing zone space used`
  - `[storage] storage space left, storage space used`

Backends – Local submission

- Local backends **`([global] backend = local)`**
  - The following batch systems are currently supported: PBS, SGE, LSF, JMS/SLURM, Host
  - The correct batch system is guessed automatically by default but can be chosen manually with **`([local] WMS = PBS,SGE,LSF,SLURM,Host)`**
  - Select the correct queue(s) **`([local] queue = short,medium)`**
  - Fairshare group (in case they are needed) **`([local] group = cmsqcd)`**
  - Supply sandbox path **`([local] sandbox path = /shared/tmp/sbox)`** The default is to use a sub-directory of the work directory.
  - In conjunction with the ParaMod each job could have different requirements (setup via variables WALLTIME, CPUTIME, MEMORY). These requirements are automatically considered for grid / SGE jobs – batch systems without requirement support need to have a broker defined. A broker is responsible for the queue selection. **`([local] broker = DummyBroker)`**
  - The default broker randomly selects a queue, while the **`SimpleBroker`** first filters out queues not matching the job requirements.

Backends – Grid submission

- Grid backends **`([global] backend = grid) [default]`**
  - Currently supported middleware tools: GliteWMS **`([grid] WMS = GliteWMS)`**
  - Obsolete: LCG, Glite
  - Can be supplied with own config files (eg. to use certain RB) **`([glite-wms] config = docs/glite_wms_DESY.conf)`**
  - White / Blacklist certain sites with problems **`([grid] sites = gridka.de -ce1.gridka.de desy.de -grid-ce5.desy.de)`**

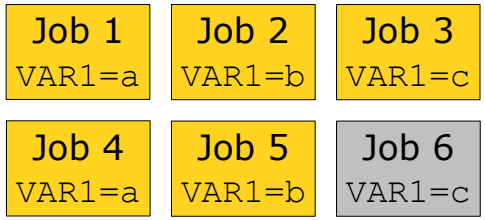


  - In case of complex VO memberships, you can setup the VO which should be written into the jdl file – by default the VO is taken from your proxy **`([grid] vo = cms)`**

Parameterized jobs

- Needed for parameter scans, Monte Carlo methods
- Can be used in conjunction with dataset based jobs
- Instead of some job module, a module derived from ParaMod is selected
- The ParaMod sets variables which another selected module **`([ParaMod] module)`** can use during execution
- In case the base module doesn't specify the maximum number of jobs itself (due to datasets, ...), it is possible to set the basic number of jobs. **`([ParaMod] jobs = 2)`**

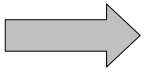
```
[global]
module = SimpleParaMod
[jobs]
module = UserMod
jobs   = 5
[ParaMod]
module = UserMod
jobs   = 2
parameter name = VAR1
Parameter values = a b c
```



Parameterized jobs

- There are several parameterizations available:
  - **SimpleParaMod**: Simple variable scan
  - **LinkedParaMod**: Simple tuple scan
  - **FileParaMod**: Get parameter variables and values from a csv file such as written by MS Excel / OO Calc
  - **UberParaMod**: Parameter scan based on the cross product of several variables / tuples. Example:

```
[ParaMod]
Parameters   = spam (ham, eggs)
spam         = 0 1
(ham, eggs) = (A, a) (B, b)
```



```
(spam, ham, eggs) = (0, A, a)
                    (1, A, a)
                    (0, B, b)
                    (1, B, b)
```

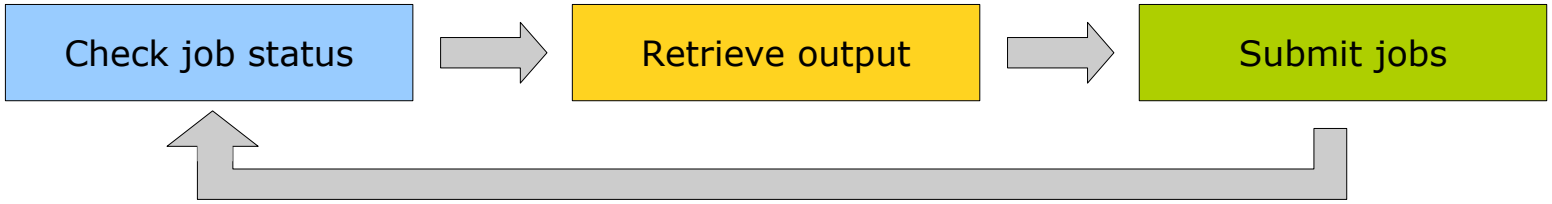
  - There are several config options for each module, which is documented in docs/documentation.conf
  - An overview of the parameters can be viewed with **-M**

- Writing your own parameterization module is VERY simple:

```
class MyParaMod(ParaMod):
    def getParams(self):
        return [{'VAR1':1}, {'VAR1':5, 'VAR2':10}]
```

Job workflow

- **Initialization phase**: The work directory (specified via `[global] workdir`) is initialized, dataset information is queried, ...
- During the **run phase**, grid-control will check, retrieve and submit jobs



- Several modes / workflow modifiers are available
  - `-c` continous mode, grid-control doesn't exit, checks job status,
  - `-G` GUI: Keep job report updated while giving status information
  - `-s` Disable the submission of jobs – just keep the job status updated and retrieve outputs
  - `-m` Specify the maximum number of job resubmission retries (default is unlimited)

CMSSW jobs

- A dedicated CMSSW module **([global] module = CMSSW)** is available, which makes working with CMSSW quite easy.
- For CMSSW jobs, you can either send your local CMSSW project files given by **([CMSSW] project area = path/CMSSW_3_1_2)** or specify the scram project to use for a clean project area:
  ([CMSSW] scram project = CMSSW CMSSW_3_1_2,
  [CMSSW] scram arch = slc4_ia32_gcc345)
- When sending project files, only those files selected by
  ([CMSSW] area files = <list of regexp>)
  will be send (default is not to include any source files)
- Give paths to CMSSW config files which will be executed:
  ([CMSSW] config file = /path/to/config.py)
- Since the CMSSW runtime is almost always too large, the runtime should be transferred via the SE for grid jobs (default: False)
  ([CMSSW] se runtime = true)

CMSSW dataset access

- The default dataset api for CMSSW modules is the CMS DBS interface **`([CMSSW] dbsapi = DBSApiv2)`**
- This means that you can simply specify CMS dataset paths in the dataset setting (you don't need an `DBS://` prefix):
  `([CMSSW] dataset = /Zmumu/Summer08/GEN-SIM)`
- By default, the global CMS production database is queried. For user datasets, either the global DBS instance URL can be changed **`([CMSSW] dbs instance)`** or the instance is given together with the dataset path in postfix notation:
  `(/Zmumu/Summer08/USER@cms_dbs_prod_local_09)`
- There is a config file fragment **`(docs/fragmentForCMSSW.py)`**, which will ready the CMSSW config file for dataset access when appended to an config file.

Conclusion / Summary

- grid-control is easy to use, powerful, modular
- grid-control is extendable through modules or external scripts
- grid-control supports submission to
  PBS, SGE, LSF, JMS/SLURM, the localhost and to the Grid
- multi-dimensional parameter scans
- CMSSW: automatic environment and dataset management
- How can you get it?
  svn co https://ekptrac.physik.uni-karlsruhe.de/public/grid-control/tags/stable
- Project website:
  https://ekptrac.physik.uni-karlsruhe.de/trac/grid-control
- Any questions ?