ILCDIRAC

A grid solution for the LC community

Stéphane Poss et al.

CERN

October 31, 2012

ILC VO

The ILC VO is dedicated to the study of future Linear Colliders: ILC and CLIC, specifically the study of the detectors concepts: ILD and SID

Members of the VO: DESY (DE), KEK (JP), CERN (CH), SLAC (USA), PNNL (USA), LAPP (FR), VINCA (R.S.), etc.

ILCDIRAC motivations and context

- Need for distributed analysis framework for mass production of MC data
  - CLIC CDR had to be written
  - Now SID DBD is being written
- Use existing solutions: DIRAC was thought to answer the needs
- Needed framework for supporting the ILC VO applications: generation, simulation and reconstruction for both detector concepts
- File catalog: metadata and replica information.

ILCDIRAC was born

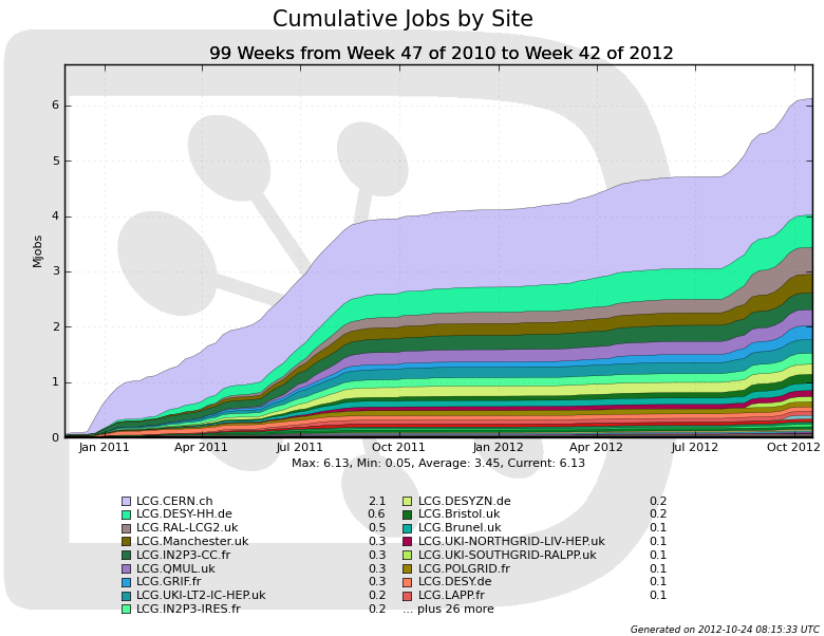
Usage

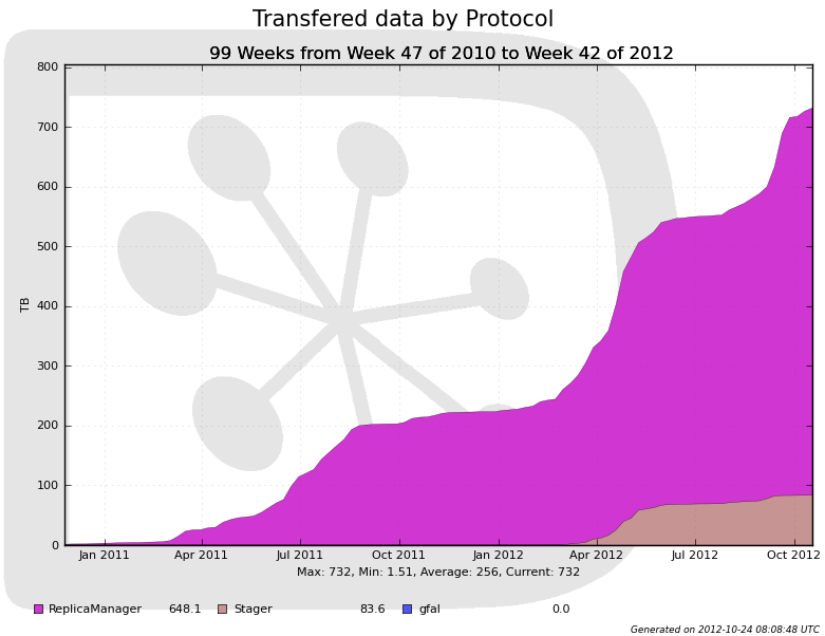
ILCDIRAC uses most services:

- Framework
- Accounting
- Workload management
- Request management
- Data management: File catalog, stager, storage element, etc.
- Transformation system: production job management

Not using Resource Status System yet!

Performance achieved in 2 years





Ran more than 6 million jobs in $\approx$ 50 sites

Transfered more than 730 TB between sites

6 million LFNs in the File Catalog and associated meta data.

Interface developments

Start of ILCDIRAC:

- Interface inspired from LHCbJob interface
- Applications are handled in one python module declaring one class
- No relations between them: all applications redefine everything
- Using output of one application as input to another was cumbersome.
- Most of all: maintenance and additions very time consuming

After 1.5 years, we needed a new interface system!

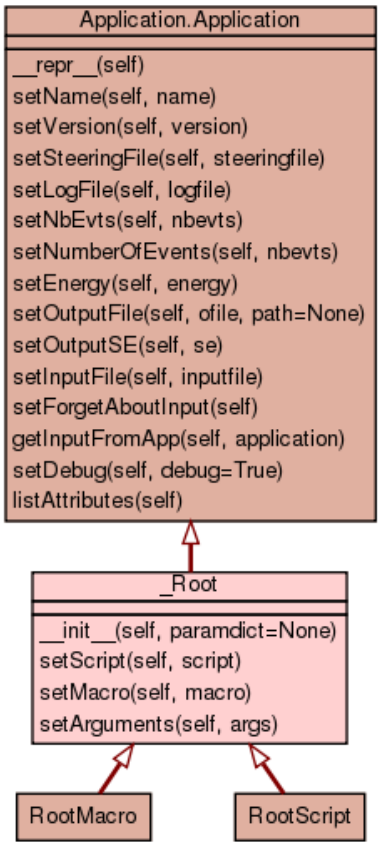
Redesigning the job interface

Motivation: separate the job definition from the application definition, add flexibility, ease maintenance

- What is an application?
  - Convert something into something else, or produce something out of nothing
  - Has a name, and certainly a version
  - Needs instructions on what to do (parameters, or set of parameters)
  - Produces log files
  - Can produce something that could be used by another application in the same job
- What kind of jobs:
  - User jobs: all inputs can be different for every job
  - Production jobs: all jobs share the same properties, but the input changes, preferably automatically (Transformation system)

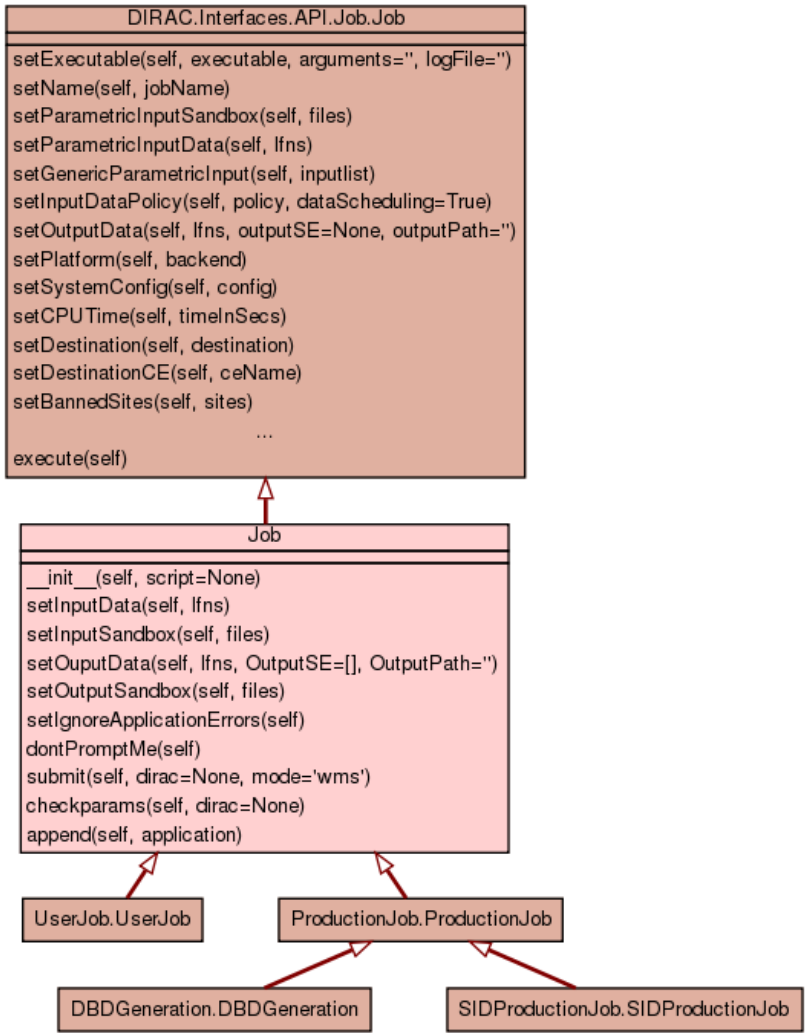
Application handling

Motivation: Provide a general framework for handling any application, using the Workflow functionality of DIRAC



This example: how ROOT is handled

Job types


DIRAC.Interfaces.API.Job.Job
setExecutable(self, executable, arguments='', logFile='')
setName(self, jobName)
setParametricInputSandbox(self, files)
setParametricInputData(self, lfns)
setGenericParametricInput(self, inputlist)
setInputDataPolicy(self, policy, dataScheduling=True)
setOutputData(self, lfns, outputSE=None, outputPath='')
setPlatform(self, backend)
setSystemConfig(self, config)
setCPUTime(self, timeInSecs)
setDestination(self, destination)
setDestinationCE(self, ceName)
setBannedSites(self, sites)
...
execute(self)
Job
__init__(self, script=None)
setInputData(self, lfns)
setInputSandbox(self, files)
setOuputData(self, lfns, OutputSE=[], OutputPath='')
setOutputSandbox(self, files)
setIgnoreApplicationErrors(self)
dontPromptMe(self)
submit(self, dirac=None, mode='wms')
checkparams(self, dirac=None)
append(self, application)
UserJob.UserJob
ProductionJob.ProductionJob
DBDGeneration.DBDGeneration
SIDProductionJob.SIDProductionJob

Job types

User jobs (UserJob):

- All jobs can have different set of inputs/parameters
- Jobs are directly submitted to Workload Management System
- Goes through the Dirac API

Production jobs (ProductionJob):

- All jobs share the same parameters, only the input is different
  - No local input sandbox allowed, only LFNs!
- Workflow is submitted to the Transformation System which creates the jobs on request, setting the right input (if any)
- Cannot go through Dirac API, but easy to create a UserJob out of it

Application definition does NOT depend on the job type.

Example of a UserJob

```
#import statements not added
ga = GenericApplication()
ga.setScript("myscript.py")
ga.setArguments("some args")

root = RootScript({"Script":"/path/to/script.exe",
                   "Arguments":"1,2,3"})
root.getInputFromApp(ga)

d = DiracILC()
j = UserJob()
res =j.append(ga)
if not res['OK']:
  print res['Message']
  exit(1)
res =j.append(root)
if not res['OK']:
  print res['Message']
  exit(1)
print j.submit(d)
```

Check
http://lcd-data.web.cern.ch/lcd-data/doc/ilcdiracdoc/
for API documentation.

Advantages

- Adding an application is independent of the job: inherit from the Application class and add whatever parameter you need, also need to create the workflow module (I provide a base module class)
- Adding a certain job type with different handling of (for example) the output is easier: inherit from the Job class and implement one method. UserJob and ProductionJob are already suitable for most applications
- Maintenance is easier

Code elements are available for anyone willing to try it. Some bits in the DIRAC Dirac.py and Job.py class would need changes to simplify the code.

Conclusion

- Very successful usage of DIRAC
- Very successful collaboration with DIRAC developers

Redesign of the ILCDIRAC job interface in a model that can be used by others: code elements are available, should be imported in DIRAC for optimal usage.

My contract ends at the end of this year: André Sailer and Christian Grefe will take over