The EGI Workload Manager

A. Tsaregorodtsev,
Aix Marseille Univ, CNRS/IN2P3, CPPM
EGI Conference, 4 November 2020
DIRAC Interware project
EGI Services
  - Managing jobs
  - Managing data
  - Managing computing resources
  - Managing workflows
Development Framework
Conclusions
Interware

- A software framework for distributed computing
- A **complete** solution to one (or more) **user community**
- Builds a layer between users and **resources**
- A **framework** shared by multiple experiments, both inside HEP, astronomy, and life sciences
An open source project

- Started as an LHCb project, became experiment-agnostic in 2009
  - First users (after LHCb) end of 2009
- Developed by communities, for communities
  - Open source (GPL3+), GitHub hosted, python 2.7
  - No dedicated funding for the development of the “Vanilla” project
  - Publicly documented, active assistance forum, yearly users workshops, open developers meetings
  - 4 FTE as core developers, a dozen contributing developers
- The DIRAC consortium as representing body
  - CNRS, CERN, IHEP, KEK
  - PNNL, University of Montpellier, Imperial College
Managing user jobs
Pilot jobs are submitted to computing resources by specialized Pilot Directors

Pilots retrieve user jobs from the central Task Queue and steer their execution on the worker nodes including final data uploading

Pilot based WMS advantages:
- increases efficiency of the user job execution
- allows to apply efficiently community policies at the Task Queue level
- allows to integrate heterogeneous computing resources
DIRAC was initially developed with the focus on accessing conventional Grid computing resources

- WLCG grid resources for the LHCb Collaboration

Grid infrastructures

- E.g. EGI, WLCG, OSG
- CREAM, HTCondorCE, ARC

Cloud infrastructures

- EGI Federated Cloud, France-Grilles cloud

Others

- Vacuum, Volunteer grids
Standalone computing clusters

- Users can connect their own computing resources
  - Not making part of any grid infrastructure

- The user site can be:
  - a single computer or several computers without any batch system
  - a computing cluster with a batch system
    - LSF, BQS, SGE, PBS/Torque, Condor
    - Commodity computer farms
  - SLURM
  - HPC centers
Users are managing jobs using various tools

Command line (batch system like interface):

```
bash-4.2# dsub /bin/echo "Hello world"
53917277
bash-4.2# dstat
JobID  Owner  JobName  OwnerGroup  JobGroup  Site  Status  MinorStatus  SubmissionTime
53917277  atsareg  Unknown  wenmr_user  NoGroup  EGI.NIKHEF.nl  Running  Application  2020-10-22 19:06:24
bash-4.2# doutput 53917277
bash-4.2# ls -l 53917277
total 4
-rw-r--r-- 1 71139 2062 12 Oct 22 19:06 std.out
```

Python API

```python
from DIRAC.Interfaces.API.Job import Job
from DIRAC.Interfaces.API.Dirac import Dirac

dirac = Dirac()
j = Job()

j.setCPUTime(500)
j.setExecutable('/bin/echo hello')
j.setExecutable('/bin/hostname')
j.setExecutable('/bin/echo hello again')
j.setName('API')

dirac.submitJob(j)
```
Several methods to install DIRAC client on user workstations/laptops (Linux flavors)

- **dirac-install** installer tool
  - Rather tedious (see tutorials)
  - Suitable for various flavors of Linux

- **Docker** container (Linux, MacOS)
  - `docker run -it -v $HOME:$HOME -e HOME=$HOME diracgrid/client:egi`

- **CVMFS** client installation (Linux)
  - `source /cvmfs/dirac.egi.eu/dirac/bashrc_egi`

- **Conda** environment (Linux, MacOS)
  - `conda create -c conda-forge --name dirac ipython dirac-grid`
  - `conda activate dirac`
Other job interfaces

- **REST API**
  - A language neutral interface for job manipulation

- The next generation DIRAC service interface will be based on HTTPS
  - Will allow for a language neutral RPC interface

- **Jupyter Notebook interface**
  - Soon available
  - DIRAC API enabled iPython shell
  - Terminal with DIRAC command line interface
  - Managing user credentials is being sorted out
    - Functional for users having grid certificates and registered in the Check-In SSO service
Bulk jobs with multiple parameters

Example JDL

```plaintext
Executable = "testParametricJob.sh";
JobName = "Parametric_ %{Name}";
Arguments = "%{Energy}";
Parameters = 3;
Parameter.Energy = {0.1, 0.2, 0.3};
Parameter.Name = {"Good", "Better", "Best"};
StdOutput = "StdOut_%{j}";
StdError = "StdErr_%{j}";
InputSandbox = {"testJob.sh"};
OutputSandbox = {"StdOut_%{j}","StdErr_%{j}"};
```
Managing user data
LFN: unique identifier within DIRAC of a file

Logical File Name
(described as paths)

LFNs are registered in catalog(s).

and there are implementations like the DFC
and you can connect as many catalogs as you want
(including the LFC or Rucio catalog)

LFNs may have PFNs, stored in SEs.

Physical File Name on Storage Elements

PFNs can be accessed with several protocols.

e.g. root, gsiftp, srm, http, file, dips
(and can also be brought online - i.e. staged)
Storage plugins

- Storage element abstraction with a client implementation for each access protocol
  - DIPS – DIRAC data transfer protocol
  - FTP, HTTP, WebDAV
  - SRM, XROOTD, RFIO, DCAP, etc
    - HEP centers specific protocols
    - Using gfal2 library developed at CERN
  - S3, Swift, CDMI: cloud specific data access protocols

- Each SE is seen by users as a logical entity
  - With some specific operational properties
    - Archive, limited access, etc
  - SE’s can be configured with multiple protocols

- New data access technologies require creating new specific plug-ins
File Catalog is a service to keep track of all the physical file replicas in all the SE’s.

- Stores also file properties:
  - Size, creation/modification time stamps, ownership, checksums
  - User ACLs

DIRAC relies on a *central* File Catalog:
- Defines a single logical name space for all the managed data
- Organizes files hierarchically like in common file systems
Combined data API

- Together with the data access components DFC allows to present data to users as a single global file system

- DataManager API is a single client interface for logical data operations
DFC is Replica and Metadata Catalog
- User defined metadata
- The same hierarchy for metadata as for the logical name space
  - Metadata associated with files and directories
  - Allow for efficient searches
- Efficient Storage Usage reports
  - Suitable for user quotas

Example query:
- `find /lhcb/mcdata LastAccess < 01-01-2012 GaussVersion=v1,v2 SE=IN2P3,CERN Name=*.raw`
Deploying a DIRAC Storage Element service in front of a user File Server

- Needs minimal DIRAC installation on the server
  - Plus adding a record to the Configuration Service
- Files should be registered in the DIRAC File Catalog
  - `dirac-dms-register-directory` tool
    - keeping file hierarchical namespace
    - registering file checksums
- The SE will be accessible with the user credentials and ACL defined in the File Catalog
- Example: Eiscat-disk Storage Element
  - With 117M files registered in a dedicated File Catalog
DM user interfaces

- **Command line tools**
  - Multiple dirac-dms-… commands
  - File Catalog console (dirac-dms-filecatalog-CLI)

- **COMDIRAC**
  - Representing the logical DIRAC file namespace as a parallel shell
    - dls, dcd, dpwd, dfind, ddu, etc commands
  - Commands for file upload/download/replication
    - dput, dget, drepl

```
bash-4.2# dput test.jdl /enmr.eu/user/a/atsareg/test/test.jdl
bash-4.2# dls -L /enmr.eu/user/a/atsareg/test/test.jdl
-rwrxrwxr-x 1 atsareg wenmr_user 256 2020-10-22 22:33:12 test.jdl
   CYFRONET-USER    dips://dirac-dms.egi.eu:9148/DataManagement/StorageElement/enmr.eu/user/a/atsareg/test/test.jdl
bash-4.2# rm test.jdl
bash-4.2# dget /enmr.eu/user/a/atsareg/test/test.jdl
bash-4.2# ls test.jdl
test.jdl
bash-4.2# drm /enmr.eu/user/a/atsareg/test/test.jdl
1 object(s) removed in total
```
Managing workflows
Massive operations

- DIRAC can deal with large numbers of jobs
  - > 100K simultaneously running jobs
  - > 10M jobs in the WMS
- DIRAC can deal with large volumes of scientific data
  - 10’s of Petabytes
  - $10^7$-$10^8$ of files and directories
- There is a need for massive (bulk) operations
  - Examples:
    - Submit and monitor 50K jobs
    - Replicate $10^5$ files from SE A to SE B
    - Remove $10^5$ files and all their replicas in all the storages
- Massive operations supported
  - Asynchronous execution
  - Automatic failure recovery
  - Data integrity checking
  - Automated data driven workflows
Transformation System for data driven workflows

- Data driven workflows as chains of data transformations
  - Transformation: input data filter + recipe to create tasks
  - Tasks are created as soon as data with required properties is registered into the system
  - Tasks:
    - Jobs submission
    - Data replication, removal
    - etc

- Transformations can be used for automatic data driven bulk data operations
  - Scheduling RMS tasks
  - Often as part of a more general workflow
DIRAC Project site: [http://diracgrid.org](http://diracgrid.org)


Tutorials:
- [https://github.com/DIRACGrid/COMDIRAC/wiki](https://github.com/DIRACGrid/COMDIRAC/wiki)
Backup slides
DIRAC Framework
DIRAC systems consist of well defined components with clear recipes for developing

**Services**
- Passive components reacting to client request
- Keep their state in a database

**Agents**
- Light permanently running distributed components, animating the whole system

**Clients**
- Used in user interfaces as well as in agent-service, service-service communications
The Framework allows to easily build these components concentrating on the business logic of the applications

- Development environment: Python, MySQL
- Using framework services (configuration, service discovery, access control, etc)
- Specific functionality can be provided in many cases as plugin modules, e.g.
  - Data access policies
  - Job scheduling policies
Extending DIRAC

- Adding new general or community specific functionalities
  - Or overriding existing algorithms
- Tools for extensions packaging and deployment
  - Example extensions in the EGI DIRAC installation
    - EiscatDIRAC: File Catalog with custom file ACLs
    - EscapeDIRAC: Corsica application portal

“Horizontal” extensibility
- For specific requirements

“Vertical” extensibility
- Community driven
Several ongoing developments

- **dips:// → https://**
  - **dips**: proprietary DIRAC protocol for RPC calls
  - **http(s)**: frameworks already exists in python 2&3 for server-side (tornado framework) and client side (requests Python module)

**Python 3**

- Migration started, first production release next year
  - DIRAC client in Python 3 available before

**DIRAC ←→ Rucio bridge**

- Development in the context of Belle II and SKA collaborations
There are multiple examples of SSO solutions

The EGI Check-in service enables access to EGI services and resources using federated authentication mechanisms

- A hub between federated Identity Providers (IdPs) and Service Providers (SPs) that are part of EGI
Oauth/OIDC Authentication

Web Portal
functional prototype

Command Line
functional prototype
Conclusions

- Large scientific communities have to employ various geographically distributed computing and storage resources.

- DIRAC provides a framework for building distributed computing systems aggregating multiple types of resources.

- DIRAC provides an integrated solution with a reach set of ready to use services for managing computing resources, application workloads and data.

- DIRAC modular architecture allows for extending the existing functionality to build high level services specific for particular user communities and architectures.

http://diracgrid.org
This work is co-funded by EGI and the EOSC-hub project (Horizon 2020) under Grant number 777536
**Useful links**

- DIRAC Project site: [http://diracgrid.org](http://diracgrid.org)