Exploiting cloud resources with DIRAC

A.Tsaregorodtsev, A.Lytovchenko
CPPM-IN2P3-CNRS, France,
D.Bauer, S.Fayer, Imperial College, UK,
23d March 2022 ISGC'22, Taipei

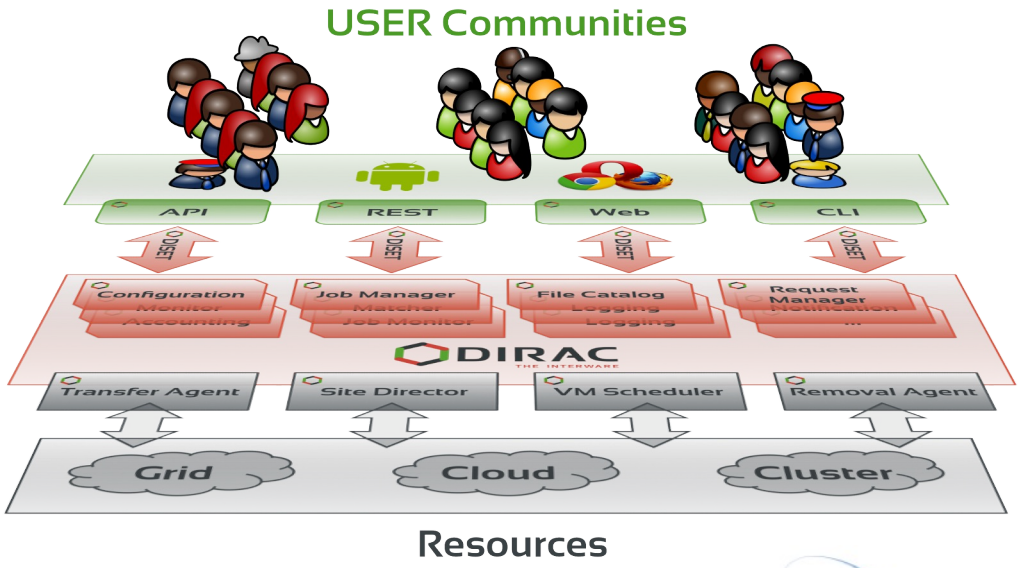
DIRAC
THE INTERWARE

Plan

- DIRAC Project
- Services: EGI, GridPP
- DIRAC WMS Architecture
- Managing Cloud resources
  - Architecture
  - Managing VM life cycle
    - Security aspects
    - Bootstrapping
    - Monitoring
  - Managing multi-core VMs
- Cloud resources usage examples
- Conclusion

Interware

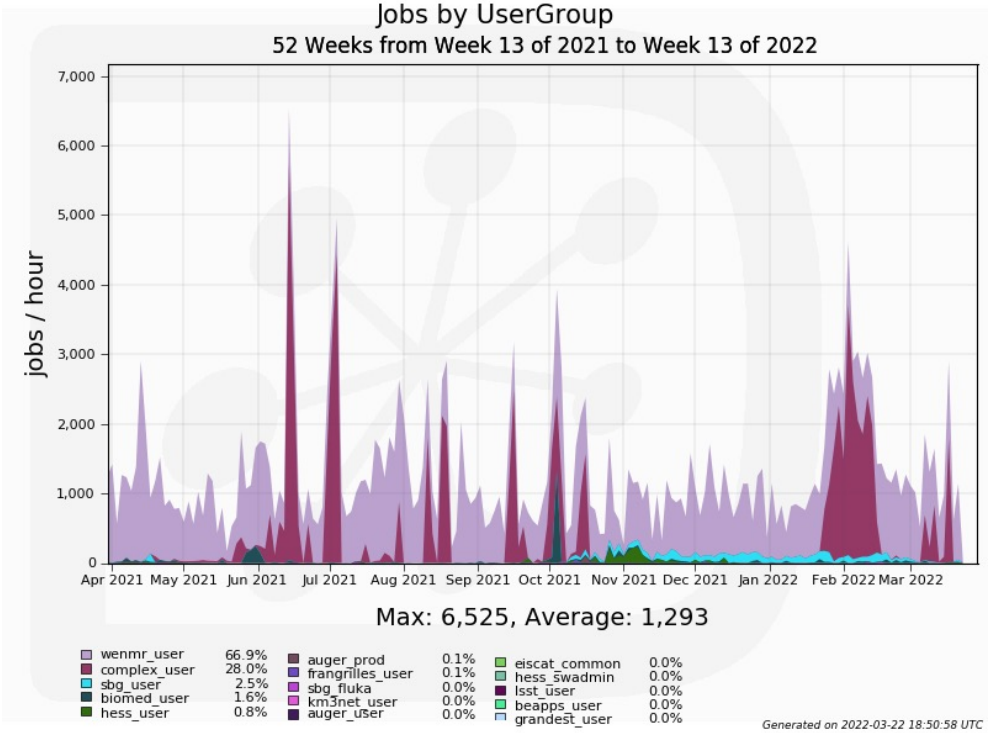
- A software framework for distributed computing
- A **complete** solution to one (or more) user community
- Builds a layer between users and resources
- A *framework* shared by multiple experiments, both inside HEP, astronomy, and life sciences



EGI Workload Manager Service

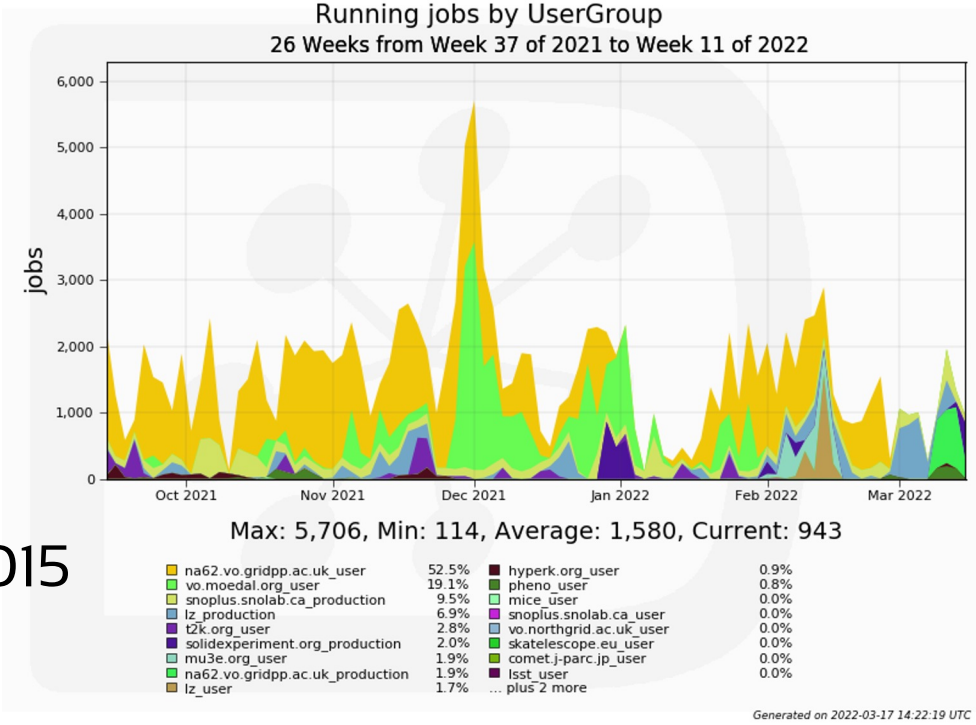
- Partners
  - Operated by **EGI**
  - Hosted by **CC/IN2P3**, Lyon
  - **DIRAC@IN2P3** Project providing software, consultancy
  - Supported via the **EGI-ACE** grant
  - *dirac.egi.eu*
- ~40 Virtual Organizations
  - enmr.eu
  - eiscat.se
  - biomed
  - Opencoast
  - …
- Usage
  - > 3 million jobs processed since the beginning of the year

DIRAC4EGI activity snapshot



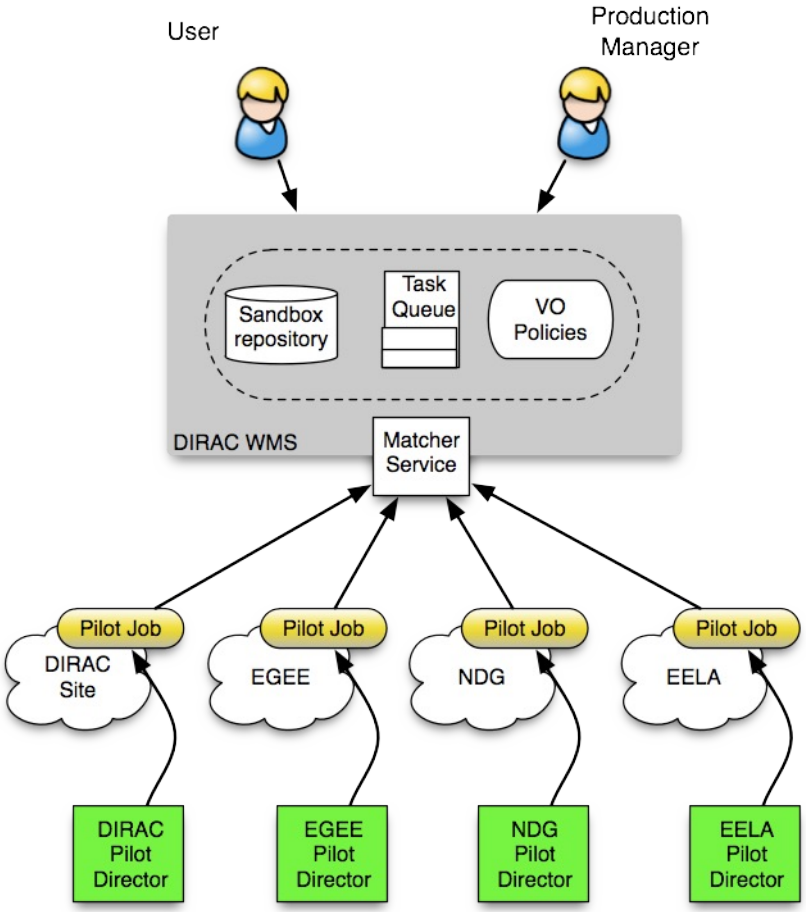
DIRAC Service

- **GridPP** is a collaboration of 19 UK universities tasked with providing grid services for the WLCG and related communities.
- **GridPP** has been providing a DIRAC service for VOs with UK participation since 2015
- 16 VOs registered
  - typically ~8 active at any one time.
- **DIRAC** is used for workload management (**grid & cloud**) and data management.



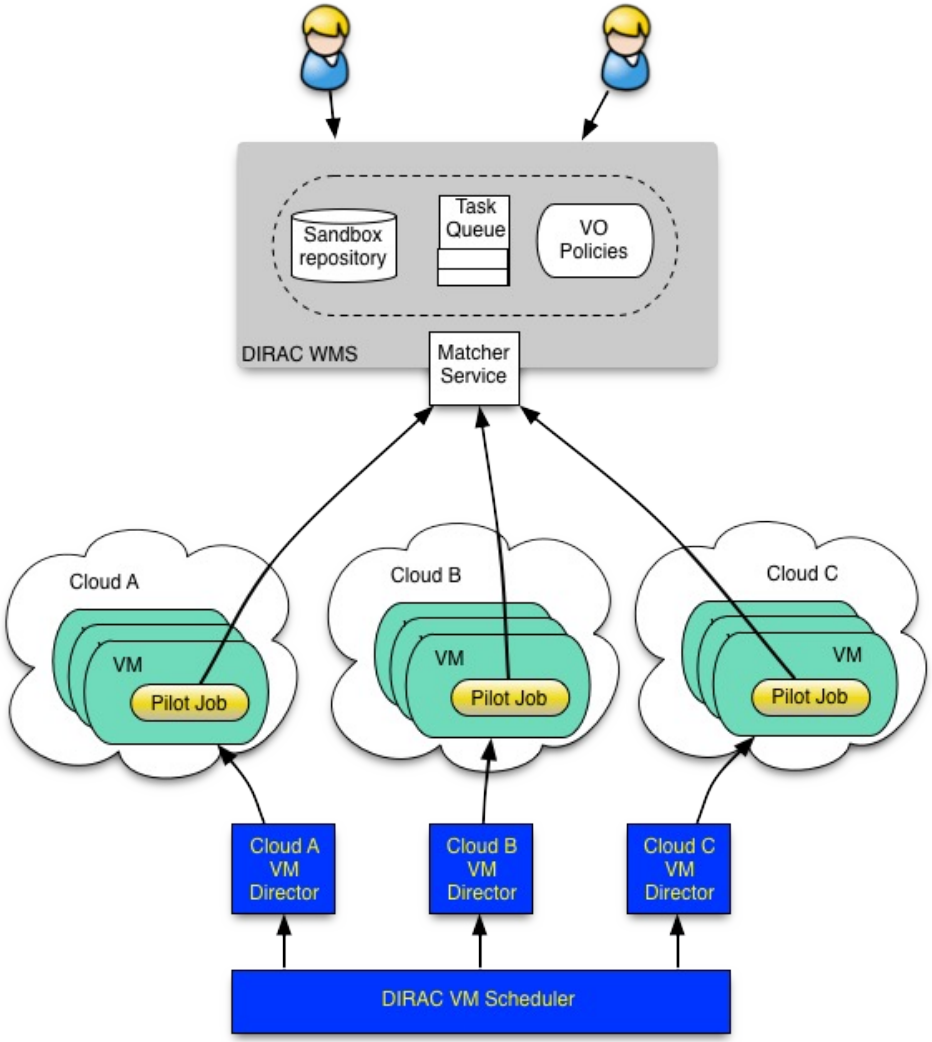
Job scheduling

- 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



Clouds

- VM scheduler
  - Dynamic VM spawning taking Task Queue state into account
  - Discarding VMs automatically when no more needed
- The DIRAC VM scheduler by means of dedicated Cloud Directors is interfaced to
  - Private:
    - OpenStack, OpenNebula
  - Amazon EC2
  - …



VM life cycle: Creation

- Cloud Director logic
  - VM creation when there are waiting user jobs in the Task Queue
  - VM creation with a predefined profile for a given VO
    - OS image, flavor/size
    - General and VO specific packages to be installed: CVMFS, gcc, perl, etc
  - A VM created for one VO can not be reused for jobs of users from another VO
    - More safety although less efficiency
  - VMs are registered in the central database to allow monitoring and controls

VM life cycle: API

- VM Creation API
  - Abstract Cloud Endpoint class
  - Implementations:
    - Openstack **REST** interface
    - OpenNebula **REST** interface
    - Amazon **boto3** Python binding
    - Apache **libcloud** API
      - **libcloud** drivers exist for multiple popular clouds
- VM Bootstrapping
  - cloud-init yaml configuration file (user data)

VM life cycle: Security(1)

- Cloud Endpoint secure access methods
  - Login/password
  - VOMS X509 certificates
  - OAuth2 tokens
- DIRAC **ProxyManager** service to maintain long living proxies
  - Similar to MyProxy service
  - Provides certificate proxies for communication with cloud endpoints
- DIRAC **TokenManager** service to provide access tokens
  - Keeping long living user refresh tokens used to generate access tokens for communications with Cloud Endpoints
- One user credentials per VO

VM life cycle: Security(2)

- Using a service certificate deployed in the VMs for communication with central DIRAC services
  - Deployed via a secure encrypted connection as part of the VM bootstrapping process
- Only outbound connections VM -> Central Service are allowed
  - Inbound SSH connection only for debugging problems in specially created VMs

VM life cycle: Monitoring

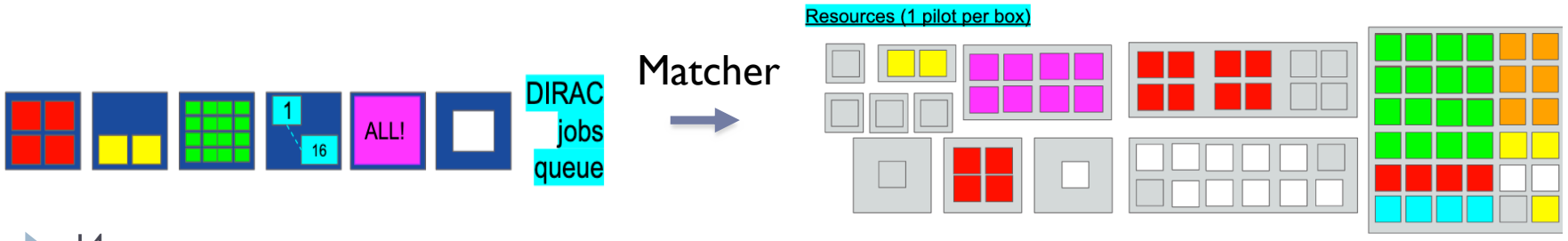
- **VirtualMachineMonitorAgent** is launched during the bootstrapping process
  - Monitors the VM status and reports to the central **VirtualMachineManager** service
    - VM heart beats, load, executed work, etc
  - Detects conditions for stopping the VM
    - No running jobs for a predefined period
    - No CPU activity for a predefined period
    - Other strategies are possible
  - Gets instructions from the central service
    - VM stopping
      - Gracefull after finishing current tasks
      - Forcefull

VM Pilots

- A standard DIRAC pilot is launched in the end of the bootstraping process
  - Installing and configuring DIRAC software
  - Checking the environment
  - Requesting and steering execution of the user payloads
    - User jobs are executed with user credentials received together with the payloads

Fat multi-core nodes

- Pilots exploit multi-core VMs using **PoolCE** "inner" Computing Element
  - On-WN batch system
  - Flexible strategy with prioritized job requests to the Matcher, e.g.:
    - First, ask for jobs requiring WholeNode tag
    - If none, ask for jobs requesting as many cores as available
    - If none, ask for jobs with MultiProcessor requirement
    - If none, ask for single-core jobs
  - The goal is to fill the nodes with payloads fully exploiting there multi-core capacity



VM life cycle: Pilots

- User payloads execution options (configurable)
  - Execution by a pilot user process
    - **InProcessCE** « inner » Computing Element
    - No barriers between parallel user jobs
    - No barriers between the user job and the pilot
  - Execution in separate processes with different user unix identities
    - **SudoCE** « inner » Computing Element
    - Parallel user jobs are isolated from each other (a la **glexec**)
  - Execution in separate **Singularity** containers

Singularity for Cloud Nodes

- **SingularityCE** allows running predefined containers as wrappers around user payloads
  - Can also be used in grid environments for a consistent environment across all platforms

- For VMDIRAC this:
  - Isolates payload from the pilot for security
  - Allows a more generic cloud image to be used
    - "Standard" CernVM WN container can be used for payload
- **SingularityCE** can be nested inside **PoolCE** so stacked jobs are also isolated from each other
- Nesting can also be used to allow a user to run their own container image.

Using clouds in the **EGI** service

- Using cloud resources for grid-like and special workflows
  - Mixing conventional grid and cloud resources in the same workflow
    - WeNMR structural biology application portal
    - ~1% of the total resources
  - Using cloud as resources with special features (biomed, climatology applications)
    - High memory nodes
    - Multi-core nodes
    - GPU nodes
- Accessing cloud endpoints through the **Openstack REST** interface and **cloud-init** for instance contextualisation

Using clouds in the **GridPP** service

Motivation for Cloud usage:

- Used by the **LZ** experiment (dark matter) to access high memory nodes for calibration workflows.
  - LZ uses DIRAC for all their processing, hence cloud integration was essential.
  - 57% of LZ job ran on cloud resources.
- Cloud is shared between cloud and grid style workflows.
- Backfill on non-GridPP resources (**LSST**).
- GridPP DIRAC uses the **libcloud** backend and **cloud-init** for instance contextualisation.

DIRAC
THE INTERWARE

Conclusions

- Cloud resources are provided by several distributed computing infrastructures in increasingly high volumes
- DIRAC is providing a framework for integration of those resources together with grid systems and computing clusters
- Cloud resources can be used as additional capacity for workflows already exploiting grids and clusters
- Clouds allow to provide resources with specific features for particular applications with requirements like high memory, GPU and others still maintaining the same DIRAC job submission and management interfaces

Acknowledgements

- This work is co-funded by the EOSC-hub project (Horizon 2020) under Grant number 777536
- EGI-ACE receives funding from the European Union's Horizon 2020 research and Innovation programme under grant agreement no. 101017567