Le Service DIRAC

A.Tsaregorodtsev, CPPM
Lille, 10 mai 2012

Plan

- DIRAC Project
  - Motivation, history
  - Open source project
- DIRAC grid middleware
  - Framework,
  - WMS
  - DMS
  - Advanced services
  - User interfaces
- DIRAC as a Service
- Conclusion

Motivation and origins

User problems on the grid

- Complicated interfaces
  - Especially for non-computing experts
- Confusing security infrastructure
  - Not easy to get and properly set up grid certificates
- Frustration with failing resources and middleware
  - Why my jobs worked yesterday and not today ?
- For small communities difficult to organize collective work
  - Lack of expertise in high level computing tasks
    - Massive jobs, massive data movement, etc
- Small communities tend to become larger with time

Large community issues

- Large user communities (Virtual Organizations) have specific problems
  - Dealing with heterogeneous resources
    - Various computing clusters, grids, etc
  - Dealing with the intracommunity workload management
    - User group quotas and priorities
    - Priorities of different activities
  - Dealing with a variety of applications
    - Massive data productions
    - Individual user applications, etc

Problems of resources providers

- Difficult to add local cluster to the pool of community resource
  - Installing grid middleware
  - Joining grid infrastructure
- Difficult to manage local resources to suite various VO requirements
  - Avoid complex VO specific configuration on sites
  - Avoid VO specific services on sites

DIRAC Grid Solution

- LHC experiments developed their own middleware to address the above problems
  - DIRAC is developed originally for the LHCb experiment
- DIRAC is providing a complete grid middleware stack with the goal:
  - Integrate all the heterogeneous computing resources available to LHCb
  - Minimize human intervention at LHCb sites
  - Make the grid convenient for the LHCb users:
    - Fault tolerance, quicker turnaround of user jobs
    - Enabling Community policies

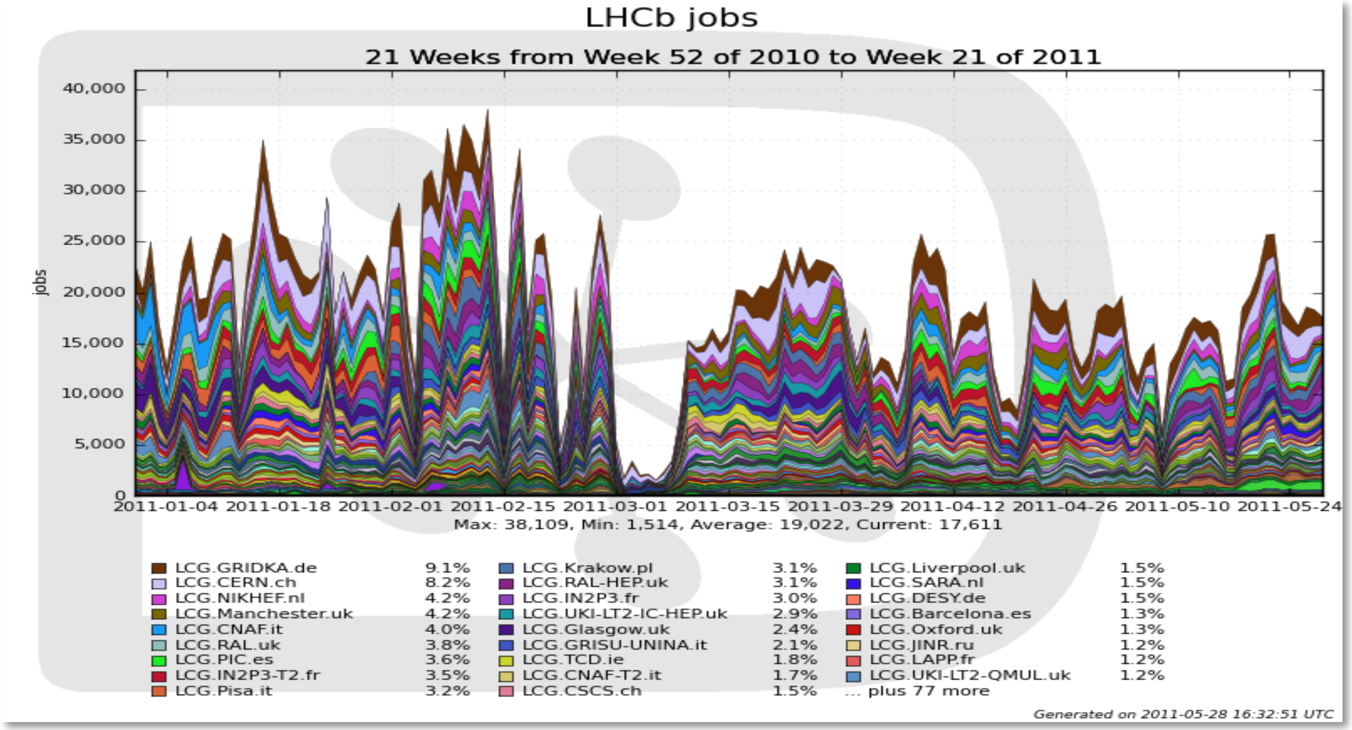
Origins

- DIRAC project was started as the LHCb distributed computing project
  - First, as a MC production engine
  - Then extended for all the other LHCb distributed computing tasks
- DIRAC was reorganized to separate generic and LHCb specific functionality in 2008-2010
  - Since 2010 DIRAC became an independent project
    - With LHCb staying the main client of the project
  - Main DIRAC developers are also LHCb experiment members
    - Guarantees of the project sustainability

LHCb Grid System

- The LHCb Distributed Computing system is built entirely with DIRAC
  - Workload Management
  - Data Management
  - High level Data Production tasks
  - User analysis
- A unique example of a complete middleware build in the same framework
  - The successful LHCb experience can now be shared with other user communities

LHCb DIRAC performance



- DIRAC performance in production
  - Up to 35K concurrent jobs in ~120 distinct sites
  - 5 mid-range central servers hosting DIRAC services
  - Further optimizations to increase capacity are possible
    - Hardware, database optimizations, service load balancing, etc

DIRAC Consortium

- Other projects are starting to use or evaluating DIRAC
  - CTA, SuperB, BES, VIP(medical imaging), …
    - Contributing to DIRAC development
    - Increasing the number of experts
  - Need for user support infrastructure
- Turning DIRAC into an Open Source project
  - DIRAC Consortium agreement in preparation
    - IN2P3, Barcelona University, CERN, …
  - http://diracgrid.org
    - News, docs, forum

DIRAC middleware

DIRAC middleware

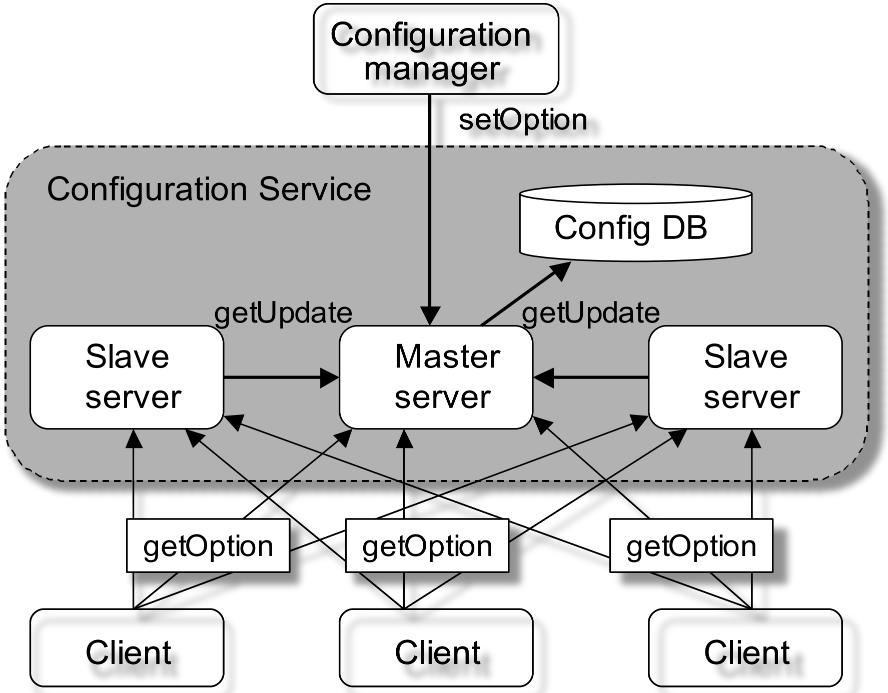
- Services oriented architecture (SOA)
- DIRAC systems consist of
  - Services
    - passive components reacting to client request
    - Keep their state in a database
  - Light distributed agents
    - permanently running components, animating the whole system
  - Clients
    - User interfaces
    - Agent-service, service-service communications
- Framework allows to easily build these components concentrating on the business logic of the applications

DIRAC Framework

- All the communications between the distributed components are secure
  - DISET custom client/service protocol
    - Focus on efficiency
    - Control and data communications
  - X509, GSI security standards
  - Fine grained authorization rules
    - Per individual user FQAN
    - Per service interface method
    - Per job

DIRAC base services

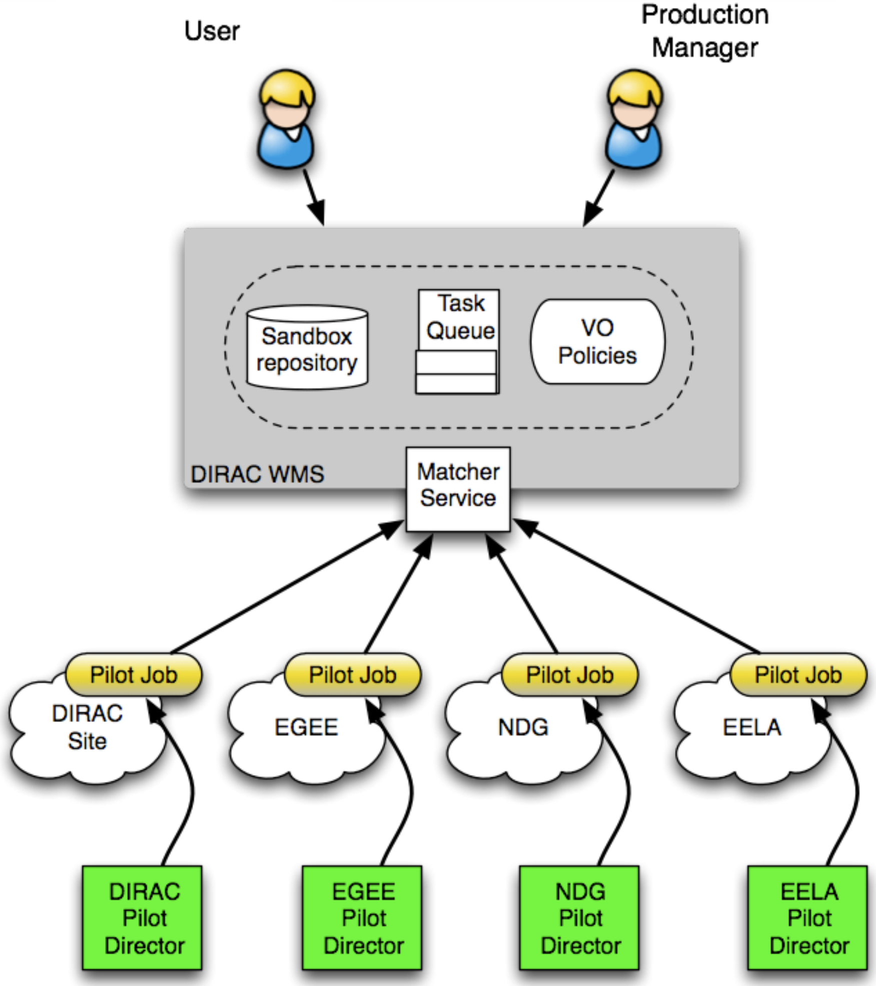
- Redundant Configuration Service
  - Provides service discovery and setup parameters for all the DIRAC components
- Full featured proxy management system
  - Proxy storage and renewal mechanism
  - Support for multiuser pilot jobs
- System Logging service
  - Collect essential error messages from all the components
- Monitoring service
  - Monitor the service and agents behavior
- Accounting service



Workload Management

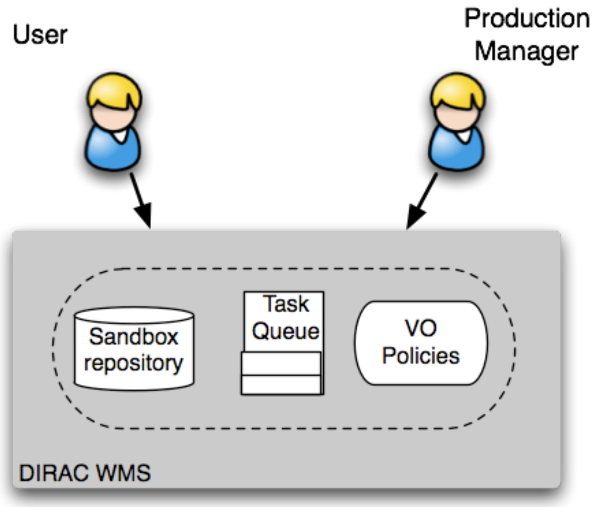
DIRAC WMS

- Jobs are submitted to the DIRAC Central Task Queue with credentials of their owner (VOMS proxy)
- Pilot Jobs are submitted by specific Directors to a Grid WMS with credentials of a user with a special Pilot role
- The Pilot Job fetches the user job and the job owner's proxy
- The User Job is executed with its owner's proxy used to access SE, catalogs, etc



WMS: applying VO policies

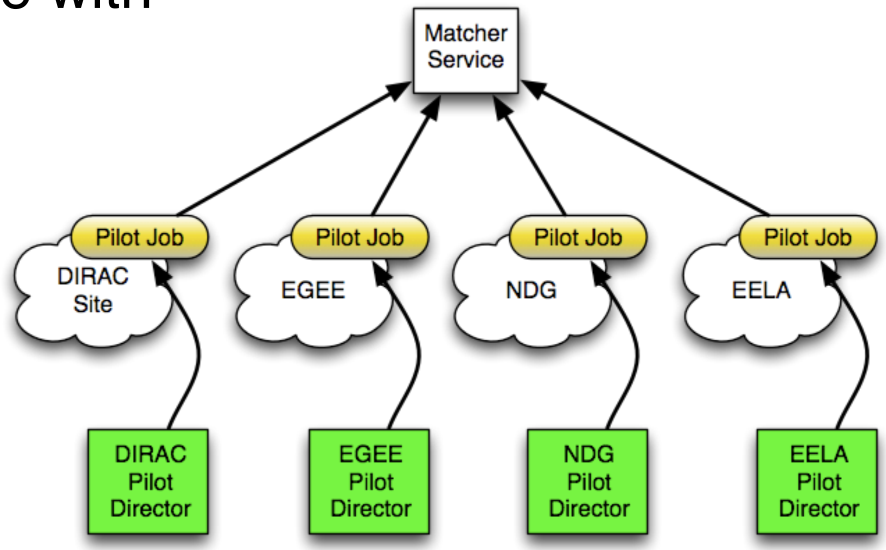
- In DIRAC jobs of all the users are treated by the same WMS
  - Same Task Queue
- This allows to apply efficiently policies for the whole VO
  - Assigning Job Priorities for different groups and activities
  - Static group priorities are used currently
  - More powerful scheduler can be plugged in
    - demonstrated with MAUI scheduler



- The VO policies application in the central Task Queue dictates the use of Multiuser Pilot Agents
  - Do not know apriori whose job has the highest priority at the moment of the user job matching
  - Similar to robot certificates
- DIRAC fully supports this mode of operation
  - Multiuser Pilots Jobs submitted with a special "pilot" VOMS role
  - Using glexec on the WNs to track the identity of the payload owner

WMS: using heterogeneous resources

- Including resources in different grids and standalone clusters is simple with Pilot Jobs
  - Needs a specialized Pilot Director per resource type
  - Users just see new sites appearing in the job monitoring
  - Grids, Clouds, Clusters, Desktop Grids, PCs
- No need for a variety of local batch queues per VO
- No need for specific VO configuration and accounting on sites



Data Management

Data Management components

- Storage Elements
  - gLite/EGI Storage Elements
    - Standard SRM interface
    - Gridftp protocol
      - Need Globus libraries, limited number of platforms
    - Allow third party transfers between them
    - Managed by the site managers within EGI SLAs
  - DIRAC Storage Elements
    - DISET based components
    - DIPS (Dirac Secure Protocol)
    - Does not allow third party transfers
      - Replication through local cache
      - Third party transfers will be available in the future
  - More Storage Elements can be included
    - (F,SF,HT,BBF)TP servers
    - iRods ?

Data Management components

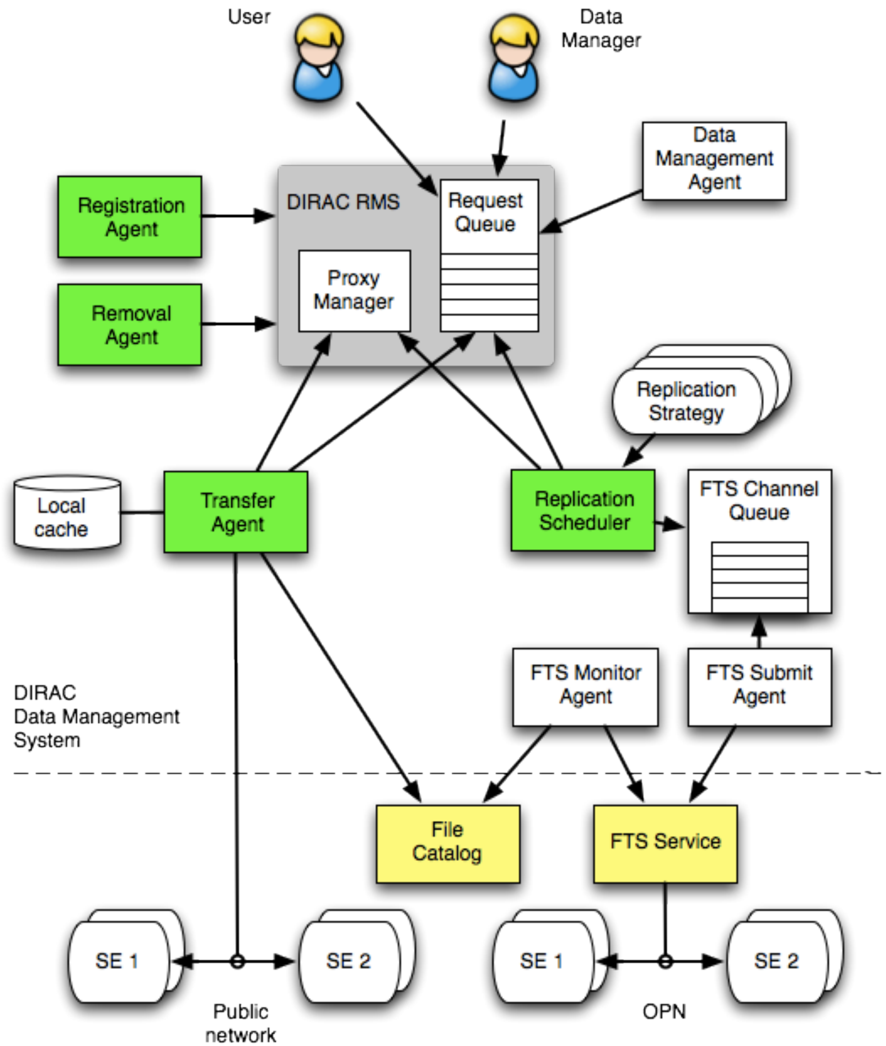
- File Catalogs
  - LCG File Catalog (LFC)
    - Part of the EGI middleware
    - Service provided by the NGI
      - ORACLE backend
    - Client tools: command line, Python API
      - Need Globus libraries
    - No User Metadata support
  - DIRAC File Catalog
    - DISET based components
    - Part of the DIRAC set of services
      - Community service
      - MySQL backend
    - Client tools: command line, CLI, Python API
    - Support of the User Metadata
  - More Catalogs can be included
    - LHCb has developed several specific catalogs in the same framework
    - iRods ?

Data Management components

- For DIRAC users the use of any Storage Element or File Catalog is transparent
  - Community choice which components to use
  - Different SE types can be mixed together
  - Several File Catalogs can be used in parallel
    - Complementary functionality
    - Redundancy
- Users see depending on the DIRAC Configuration
  - Logical Storage Elements
    - e.g. DIRAC-USER, M3PEC-disk
  - Logical File Catalog

Data Management

- Based on the Request Management System
- Asynchronous data operations
  - transfers, registration, removal
- Two complementary replication mechanisms
  - Transfer Agent
    - user data
    - public network
  - FTS service
    - Production data
    - Private FTS OPN network
    - Smart pluggable replication strategies



File Catalog Metadata

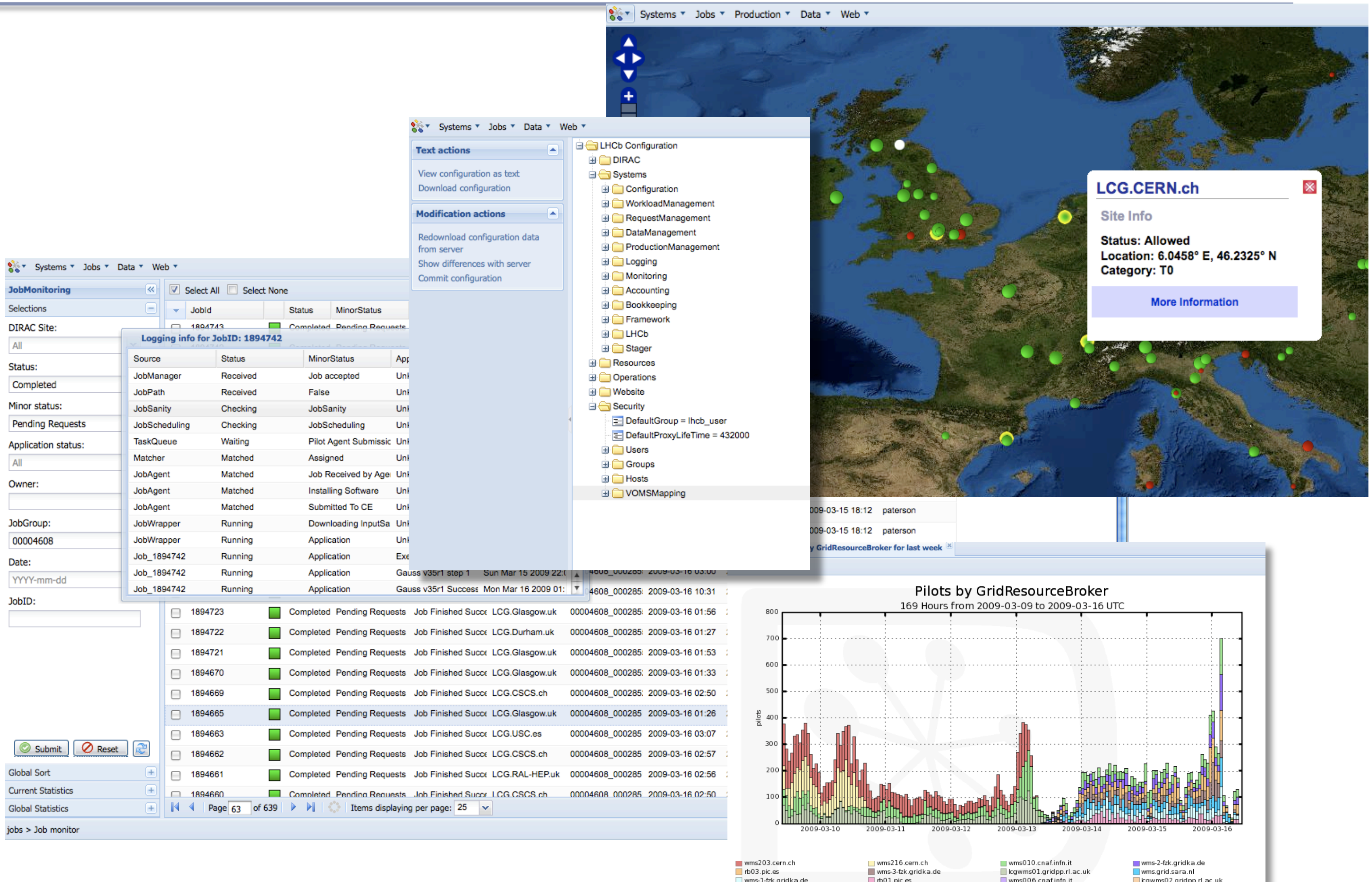
- Similar functionality with the AMGA metadata service
  - But coupled with the replica catalog to boost efficiency
- Metadata can be associated with each directory as key:value pairs to describe its contents
  - Int, Float, String, DateTime value types
- Some metadata variables can be declared indices
  - Those can be used for data selections
- Subdirectories are inheriting the metadata of their parents
- Data selection with metadata queries. Example:
  - `find . Meta1=Value1 Meta2>3 Meta2<5 Meta3=2,3,4`
- File metadata can also be defined

User Interfaces

DIRAC: Secure Web Portal

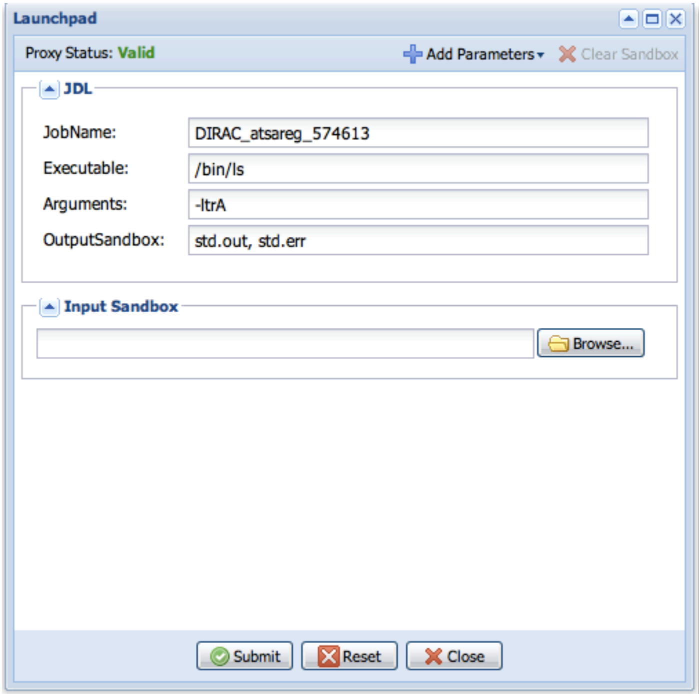
- Focus on the Web Portal as the main user tool for interactions with the grid
- Intuitive desktop application like interface
  - Ajax, Pylons, ExtJS Javascript library
- Monitoring and control of all activities
  - User job monitoring and manipulation
  - Data manipulation and downloads
  - DIRAC Systems configuration and management
- Secure access
  - Standard grid certificates
  - Fine grained authorization rules

Web Portal: example interfaces



Web Portal: user tasks

- Job submission through the Web Portal
  - Full GSI security
  - Sandboxes uploading and downloading
    - Difficult for bulky data files though
  - Generic Job Launchpad panel exists in the basic DIRAC Web Portal
    - Can be useful for newcomers and occasional users



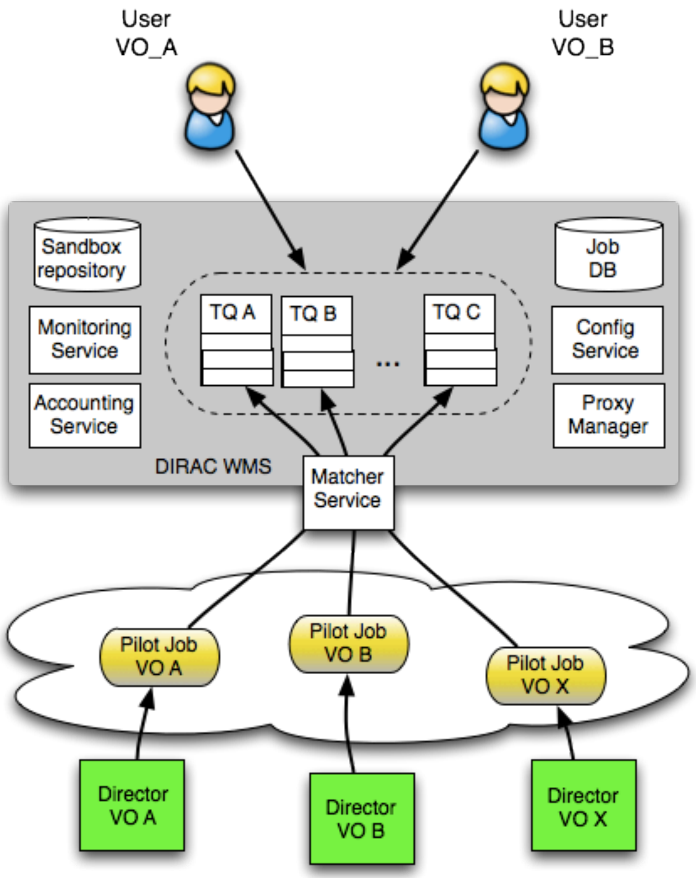
- Specific application Web Portals can be derived
  - Community Application Servers
    - All the grid computational tasks steered on the web
  - VO "formation" DIRAC instance to be deployed at CC/IN2P3

DIRAC user interfaces

- Easy client installation for various platforms (Linux, MacOS)
  - Includes security components
- JDL notation for job description
  - Simplified with respect to the « standard » JDL
- Command line tools
  - à la gLite UI commands
  - e.g. `dirac-wms-job-submit`
- Extensive Python API for all the tasks
  - Job creation and manipulation, results retrieval
    - Possibility to use complex workflow templates
  - Data operations, catalog inspection
  - Used by GANGA user front-end

DIRAC as a service

- Need to manage multiple VOs with a single DIRAC installation
  - Per VO pilot credentials
  - Per VO accounting
  - Per VO resources description
- Pilot directors are VO aware
  - Job matching takes pilot VO assignment into account



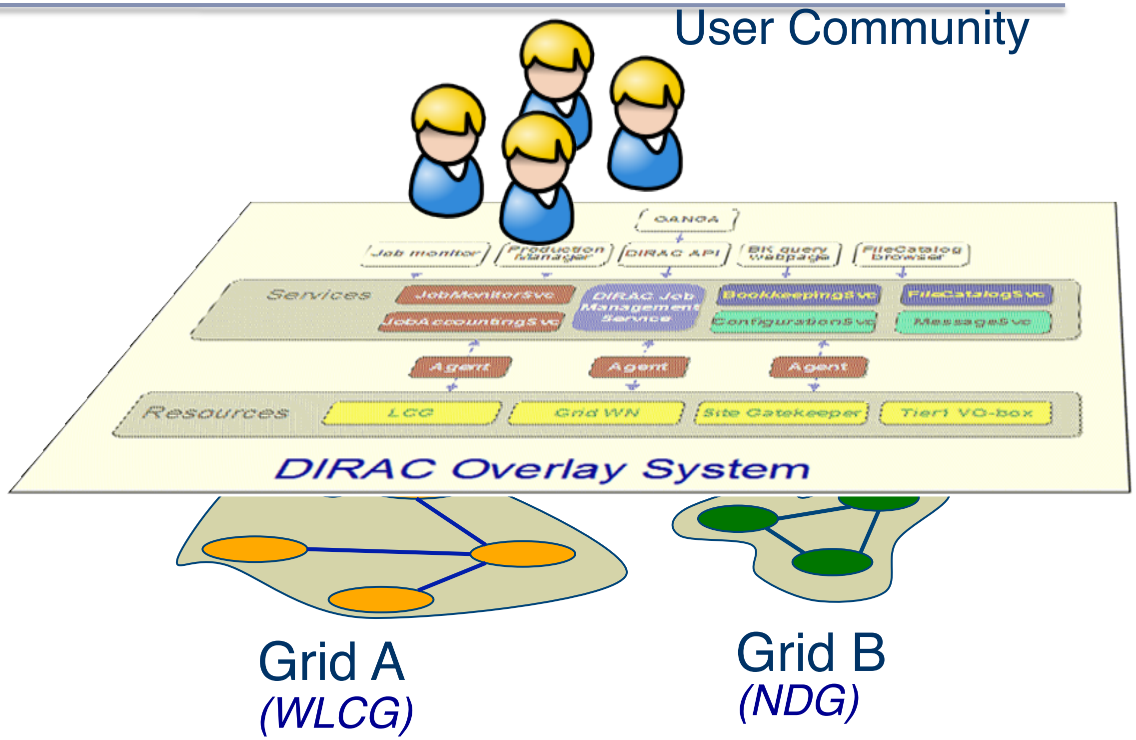
Conclusions

- DIRAC Project provides a general purpose middleware which is proven by a successful use in LHCb and other experiments
- The project has a very active user and developer communities
- The main goal is to facilitate the use of the grid and other distributed resource
- The DIRAC middleware is taken on board now by the France-Grilles Initiative to provide a user friendly grid access service

Backup slides

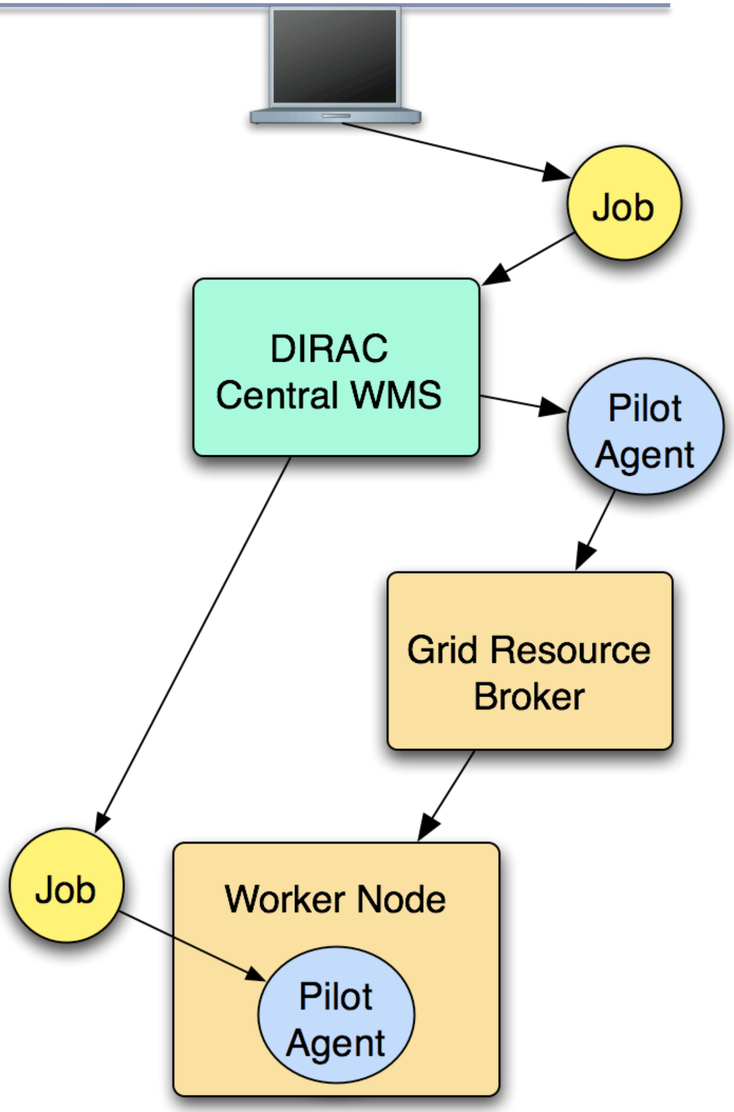
DIRAC overlay network

- DIRAC pilots form an overlay network hiding the variety of underlying resources
  - A way for grid interoperability for a given Community
  - Needs specific Agent Director per resource type
- From the user perspective all the resources are seen as a single large “batch system”



Pilot Jobs in a nutshell

- Pilot agents are deployed on the Worker Nodes as regular jobs using the standard grid scheduling mechanism
  - Form a distributed Workload Management system
  - Reserve the resource for immediate use
- Once started on the WN, the pilot agent performs some checks of the environment
  - Measures the CPU benchmark, disk and memory space
  - Installs the application software
- If the WN is OK the user job is ***pulled*** from the central DIRAC Task Queue and executed
  - Terminate gracefully if no work is available

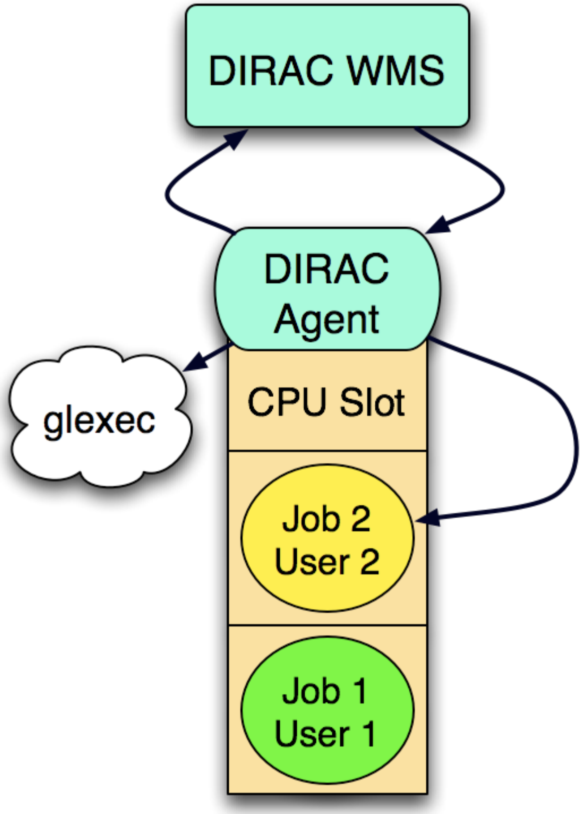


User Job efficiency

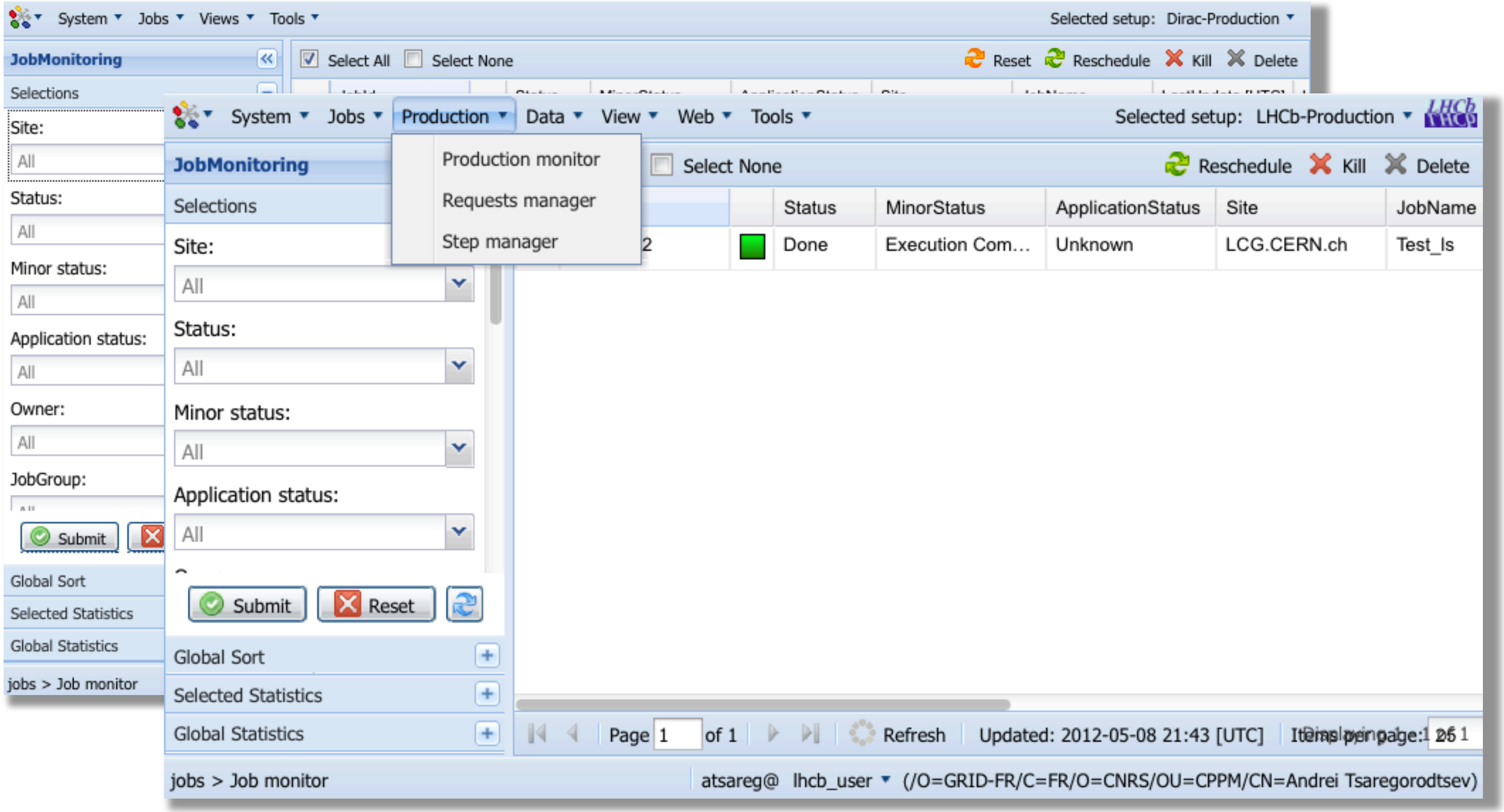
- Improved visible reliability due to pilot agents
  - ~96% efficiency for DIRAC jobs vs 70-90% efficiency for the WLCG jobs
- If some resources are failing, it is just seen as a reduced pool of resources for the users
- An excess of Pilot Jobs over User Jobs just to cover inefficiencies of Computing Resources or Grid middleware
  - it is normal that computing resources are failing but
  - it is not normal that users are suffering from that

Workload optimization

- Pilot Agents work in an optimized 'Filling Mode'
  - Multiple jobs can run in the same CPU slot
  - Significant performance gains for short, high priority tasks
  - Also reduces load on LCG since fewer pilots are submitted
  - Needs reliable tools to estimate remaining time in the queue
- Considering also agents in a "preemption" mode
  - Low priority task can be preempted by a high priority tasks
    - Low priority, e.g. MC, jobs behave as resource reservation for analysis jobs



LHCb Portal


System Jobs Production Data View Web Tools
Selected setup: LHCb-Production
JobMonitoring
Production monitor
Requests manager
Step manager
Select None
Reschedule
Kill
Delete
Selections
Site:
All
Status:
All
Minor status:
All
Application status:
All
Submit
Reset
Global Sort
Selected Statistics
Global Statistics
Status
MinorStatus
ApplicationStatus
Site
JobName
Done
Execution Com…
Unknown
LCG.CERN.ch
Test_ls
Page 1 of 1
Refresh
Updated: 2012-05-08 21:43 [UTC]
jobs > Job monitor
atsareg@ lhcb_user
(/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev)

Asynchronous operations

- File Catalog operations are generally synchronous
  - Quick, can wait for the prompt
- Physical data operations can take very long time
  - And even fail in the end
- For example, consider removing data:
  - Delete replicas on all the SEs
  - Delete files (lfns)
  - Delete directories (recursively)
- Long operations are performed asynchronously
  - Do not wait for completion
  - Make sure the operation is accomplished despite possible problems

Example job submission

```
from DIRAC.Interfaces.API.Dirac import Dirac
from Extensions.LHCb.API.LHCbJob import LHCbJob
…
myJob = LHCbJob()
myJob.setCPUTime(50000)
myJob.setSystemConfig('slc4_ia32_gcc34')
myJob.setApplication('Brunel','v32r3p1','RealDataDst200Evts.opts','LogFileName.log')
myJob.setName('DIRAC3-Job')
myJob.setInputData(['/lhcb/data/CCRC08/RAW/LHCb/CCRC/420157/420157_0000098813.raw'])
#myJob.setDestination('LCG.CERN.ch')
dirac = Dirac()
jobID = dirac.submit(myJob)
...
dirac.status(<JOBID>)
dirac.parameters(<JOBID>)
dirac.loggingInfo(<JOBID>)
...
dirac.getOutputSandbox(<JOBID>)
```

Advantages for site resources providers

- No need for a variety of local batch queues per VO
  - One long queue per VO would be sufficient
  - 24-48 hours queue is a reasonable compromise
    - Site maintenance requirements
  - Reduced number of grid jobs
- No need for specific VO configuration and accounting on sites
  - Priorities for various VO groups, activities
  - User level accounting is optional
- In the whole it can lower the site entry threshold
  - Especially useful for newcomer sites

Resources provisioning

- DIRAC middleware facilitates access to various types of resources
  - gLite based grids
  - Standalone clusters
    - Simple SSH accessible account is sufficient to include the site
  - Clouds (Amazon, OpenNebula, OCCI compliant)
    - Virtual machine scheduling
  - Desktop Grid
    - Based on BOINC technology
    - Support for multiple platforms with virtualization
  - Standalone PCs