EPJ Web of Conferences **245**, 03035 (2020) https://doi.org/10.1051/epjconf/202024503035
CHEP 2019

The DIRAC interware: current, upcoming and planned capabilities and technologies

Federico Stagni[1,*], *Andrei* Tsaregorodtsev[2,**], *André* Sailer[1,***], and *Christophe* Haen[1,****]

[1]CERN, EP Department, European Organization for Nuclear Research, Switzerland
[2]Aix Marseille University, CNRS/IN2P3, CPPM, Marseille, France

Abstract. Efficient access to distributed computing and storage resources is mandatory for the success of current and future High Energy and Nuclear Physics Experiments. DIRAC is an interware to build and operate distributed computing systems. It provides a development framework and a rich set of services for the Workload, Data and Production Management tasks of large scientific communities. A single DIRAC installation provides a complete solution for the distributed computing of one, or more than one collaboration. The DIRAC Workload Management System (WMS) provides a transparent, uniform interface for managing computing resources. The DIRAC Data Management System (DMS) offers all the necessary tools to ensure data handling operations: it supports transparent access to storage resources based on multiple technologies, and is easily expandable. Distributed Data management can be performed, also using third party services, and operations are resilient with respect to failures. DIRAC is highly customizable and can be easily extended. For these reasons, a vast and heterogeneous set of scientific collaborations have adopted DIRAC as the base for their computing models. Users from different experiments can interact with the system in different ways, depending on their specific tasks, expertise level and previous experience using command line tools, python APIs or Web Portals. The requirements of the diverse DIRAC user communities and hosting infrastructures triggered multiple developments to improve the system usability: examples include the adoption of industry standard authorization and authentication infrastructure solutions, the management of diverse computing resources (cloud, HPC, GPGPU, etc.), the handling of high-intensity work and data flows, but also advanced monitoring and accounting using no-SQL based solutions and message queues. This contribution will highlight DIRAC's current, upcoming and planned capabilities and technologies.

1 Introduction

DIRAC [1] is a software framework that enables communities to interact with distributed computing resources. It builds a layer between users and resources, hiding diversities across computing, storage, catalog, and queuing resources. DIRAC has been adopted by several

*e-mail: federico.stagni@cern.ch
**e-mail: atsareg@in2p3.fr
***e-mail: andre.philippe.sailer@cern.ch
****e-mail: christophe.denis.haen@cern.ch

HEP and non-HEP experiments' communities [2], with different goals, intents, resources and workflows: it is experiment agnostic, extensible, and flexible [3].

1.1 The DIRAC project

DIRAC is an open source project, which was started around 2002 as an LHCb project. Following interest of adoption from other communities its code was made available under open licence in 2009. Now, it is hosted on GitHub[1] and is released under the GPLv3 license. DIRAC has no dedicated funding scheme; communities using it are welcome to participate in its development. DIRAC is publicly documented, with an active assistance forum animated by the users and developers themselves. A yearly user workshop and weekly open developers meetings gather together users and experts. The project counts about five core programmers, and a dozen contributing developers.

The DIRAC consortium has been established in 2014 as a representing body for the development and maintenance of the DIRAC software. The consortium counts a small set of active members, each of which elects a representative, while consortium members elect every second year a Director, and a Technical Director. Institutes that are part of the consortium engage in the maintenance and in the promotion of the DIRAC software.

DIRAC is a collection of "systems", i.e. the Workload Management System (WMS), the Data Management System (DMS), the Transformation System (TS), the Request Management System (RMS), and others. Each of these DIRAC systems has components (processes) like services and agents, while databases are used to persist the needed information; some of these systems are introduced in the following sections. DIRAC architecture is a microservice architecture.

1.2 Goal and organization of this paper

Within this paper we explore current and upcoming DIRAC features. The intent is to show how DIRAC is a fully capable distributed computing system. This paper is organized as following: section 2 explains the capabilities of the DIRAC Workload Management System. Section 3 explores DIRAC Data Management System. Section 4 explains how DIRAC provides a fully capable Production and Dataset management system. Section 5 explains DIRAC development plans. Finally, conclusions are given in the section 6.

2 Exploiting computing resources

The DIRAC Workload Management System (WMS) is in charge of exploiting distributed computing resources. In other words, it manages jobs, and pilot jobs [4] (from here on simply called "pilots").

The Grid model was initially conceived as a "push" model, where jobs were submitted from a *queue* of jobs, managed by each and every experiment in an independent way through their Workload Management software. The "push" model proved to be inefficient and error prone. To face these issues DIRAC introduced, back in 2006, the so-called pilots, which are startup scripts which land at the worker node, perform sanity checks and then pull payload jobs from the central queue, by matching the capabilities of the worker nodes with the requirements of the waiting jobs. A pilot job that fails prior to having matched a job causes no particular troubles. The advantages of the pilot job concept are now well established: pilots are not only increasing the aggregate users' job throughput efficiency, but also helping to

[1] https://github.com/DIRACGrid

manage the heterogeneous computing resources, presenting them to the central services in a uniform coherent way. Each LHC Virtual Organization (VO) has, since then, moved to the pilots model, which is now a standard solution.

More recently, the emergence of new distributed computing resources (private and commercial clouds, High Performance Computing clusters, volunteer computing, etc) changed the traditional landscape of computing for offline processing. It is therefore crucial to provide a very versatile and flexible system for handling distributed computing (production and user data analysis). If we restrict for a moment our vision to LHC experiments, and we analyze the amount of CPU cycles they used in the last year, we can notice that all of them have consumed more CPU-hours than those official reserved (pledged) to them by WLCG (the Worldwide LHC Computing Grid) in accordance with Memorandum of Understanding (MOU) [5]. Each community found ways to exploit non-reserved CPUs (or even GPUs), often not supported resources and computing elements. Such resources may be private to the experiment (e.g. the "online" computing farm - often simply called "High Level Trigger" farm) or public; resources may sometimes be donated free of charge, like in the case of volunteer computing, or not, like public commercial cloud providers. Integrating non-grid resources is common to all communities that have been using WLCG in the past, and still do. Communities that use DIRAC want to exploit all possible CPU or GPU cycles. Software like DIRAC aims to make this easy, and the DIRAC pilot is the *federator* of each and every computing resource.

The transparent access to the underlying resources is realized by implementing the pilot model. The DIRAC pilot has the following characteristics:

- a DIRAC pilot is what creates the possibility to run jobs on a worker node;
- it is a simple, standalone python project, that can be easily extended by communities which want to provide their own commands;
- it can be sent, as a "pilot job", to all types of GRID Computing Elements (CEs);
- can be run as part of the contextualization of a (Virtual, or not) Machine;
- can run on almost every computing resource, provided that:
  - Python 2.6+ is installed on the WN;
  - it hosts an Operating System onto which DIRAC can be installed (i.e. a Red Hat Enterprise Linux derivative, on a x86 architecture).

The main role of the DIRAC WMS is to exploit different types of computing resources. These include:

- Grids, via Computing Element. The Computing Element types supported are CREAM (EGI), HTCondor-CE (OSG), and ARC (NorduGRID);
- Clusters behind a batch system: it often happens that (e.g. university) computing clusters are only accessible through a locally configured batch system. For this case, DIRAC provides the possibility to access through an SSH/GSISSH tunnel, a really thin layer that we call "SSH CE". In this case, DIRAC interacts directly with the batch system (PBS, LSF, Condor, Torque and SLURM are among those supported);
- Vacuum type of resources, like VAC/vcycle resources, but also BOINC Volunteer resources, or experiments' farms, like in the case of LHCb's HLT farm (the High Level Trigger farm);
- Virtual Machines (VMs) schedulers, with support to Openstack, Keystone v2 and v3, OpenNebula XML-RPC, Amazon EC2 (boto2), Apache libcloud, rocci cli, OCCI REST. For this

case, the contextualization of VMs come from standard images with, at least, the DIRAC pilot;

- High Performance Computing (HPC) sites, that may require specific configurations.

DIRAC WMS manages single jobs, and single pilots. Collections of jobs are normally dubbed "Productions": to know how DIRAC manages Productions, please refer to section 4.

3 Organizing Data

The DIRAC Data Management System (DMS), together with the DIRAC Storage Management System (SMS) provides the necessary functionalties to execute and control all activities related with your data. The DMS provides the basic functionalities to upload (or remove) a local file in (or from) a Storage Element (SE), and register the corresponding replica(s) in the configured File Catalog(s) (FC). Several SEs endpoint types are supported, as well as several FC types.

DIRAC also provides the functionalities for running massive data replications (also using File Transfer Service - FTS[6]) or retrievals of data archived on Tape for its later processing. This functionality can be achieved using the DIRAC Request Management System (RMS) and DIRAC Transformation System (TS) that will be discusses in later sections.

To achieve this functionality the DMS and SMS require a proper description of the involved external servers (SE, FTS, etc.) as well as a number of Agents and associated Services that animate them. In the following sections the different aspects of each functional component are explained in some detail.

The whole DIRAC DMS relies on a few key concepts:

- **Storage Element (SE)**: abstraction of a physical storage endpoint, described in the DIRAC *Configuration System*, together with all the configuration necessary to physically access the files.
- **Logical File Name (LFN)**: the LFN is the name of a file, a path. It uniquely identifies a File throughout the DIRAC namespace. A file can have one or several *Replicas*.
- **Replica**: The physical copy of an LFN, stored at a Storage Element (SE). The couple `(LFN,Storage Element)` uniquely identifies a physical copy of a file (also known as **Physical File Name**, or *PFN*). PFNs can be accessed via several protocols, e.g. `root, gsiftp, srm, http, file, dip`.
- **Catalog**: This is the namespace of the DMS. Files and their metadata are listed there. The concept of *Catalogs* is just the one of a *namespace*. It is a place where you list your files and their metadata (size, checksum, list of SEs where they are stored, etc). Arbitrary metadata keys are also possible, and can be used to find files. DIRAC supports having several catalogs: in general, any operation done to one catalog will be performed to the others. If a new catalog needs to be added, such catalog just need to have a corresponding plugin.

Systems in DIRAC (other than DMS) or users, when dealing with files, only have to care about LFNs. If, for some (unlikely) reasons, they need to address a specific replica, then they should use the couple `(LFN, Storage Element name)`. At no point, anywhere, there is a protocol or a URL leaking out of the low level of the DMS.

The DIRAC DMS is a system for managing single files, and implementing the concepts listed above. Collections of files are normally dubbed *Datasets*: to know how DIRAC manages *Datasets*, please refer to section 4.

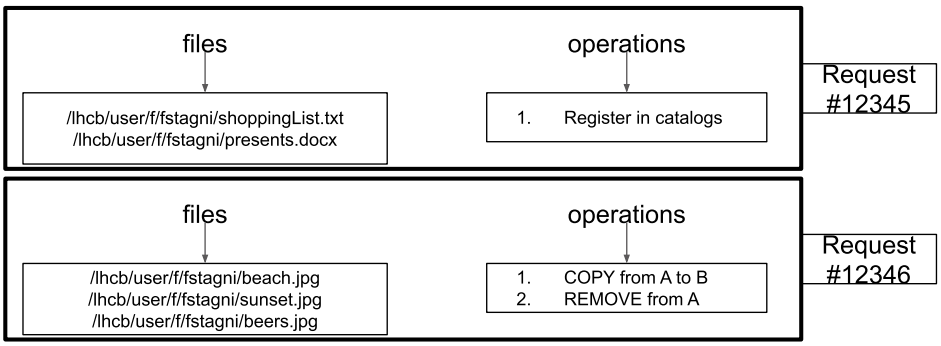

Figure 1. An example of two `Requests` in the RMS, each containing one or more `operation`, and one or more `file`

4 DIRAC for jobs Productions, and Datasets management

In order to know how DIRAC organizes jobs into *Productions*, and files into *Datasets*, we first need to know about two DIRAC systems: the **Request Management System (RMS)**, and the **Transformation System (TS)**.

4.1 The DIRAC Request Management System (RMS)

The DIRAC Request Management System (RMS) is a generic system that allows for asynchronous actions execution. Its application ranges from failover system (to cover the case when, e.g. a DIRAC service or a Storage Element is unavailable at a certain point in time) to asynchronous task list (typically, for large scale data management operations like replications or removals). The RMS service is itself resilient to failure thanks to Request Proxies that can be scattered around your installation.

At the core of the RMS are `Requests`, `Operations` and `Files`.

- A **Request** is like a TODO list associated to a User and group. For example, this TODO list could be what is left to do at the end of a job (setting the job status, moving the output file to its final destination, etc).
- Each item on this TODO list is described by an **Operation**. There are several types of Operations readily available in DIRAC, for example `ReplicateAndRegister` (to copy a file), `RemoveFile` (to remove a file), `ForwardDISET` (to execute DISET calls), `SetFileStatus` (for setting a status of a file in the Transformation System), etc. DIRAC extensions may code their specialized Operations if needed.
- **Files** are LFNs. When an Operation acts on LFNs, Files corresponding to the LFNs are associated to the Operation. But not all the Operations have Files.

An example of two fictitious requests, together with their files and operations can be found in figure 1.

4.2 The DIRAC Transformation System (TS)

The DIRAC Transformation System (TS) is used to automatise common tasks related to production activities. Just to make some basic examples, the TS can handle the generation of Simulation jobs, or Data Re-processing jobs as soon as a ‘pre-defined‘ data-set is available,

or Data Replication to 'pre-defined' SE destinations as soon as the first replica is registered in the Catalog.

The TS is a generic system for queueing similar **operation types** on certain **datasets** and forward them to the appropriate **systems**. A system is either (today) the DIRAC WMS (for productions) or the DIRAC RMS (for dataset management operation types)

In practice, the TS is a key component in DIRAC for managing datasets, and jobs productions. It is a very flexible systems, with a neat and simple design. The main parameters of a Transformation are the following:

- Type (e.g. Simulation, DataProcessing, Removal, Replication)
- The possibility of having (or not) Input Files.
- A *Plugin*, which defines the way input datasets are split into groups (e.g. by size, by destination, by metadata, or by whatever can be coded)

Within the TS a user can (for example), generate several identical tasks, differing by few parameters (e.g. Input Files list), extend the number of tasks, have one single high-level object (the Transformation) associated to a given production for global monitoring. Two admittedly simplistic examples follow:

- Example for *dataset management*: take all my holidays pictures from 2018 with `tag=sunset`, make sure that there is one copy on tape and one on disk, distributed on all the sites according to free space, and group the operations by group of at most 100 files;
- Example for *jobs productions*: take all my holidays pictures from 2018 with `tag=sunset`, make sure to run (only once) the `red-enhancer` workflow on each one of them, using only Tier2 sites.

From the examples above, it should become apparent that full flexibility is given with respect to the files' distribution and jobs' management.

4.3 Combining DIRAC systems for Dataset and Productions Management

By combining the functionalities of the DIRAC TS with those of the DIRAC WMS, DIRAC provides a system for running Jobs productions, with a workflow presented in figure 2. For achieving a full Dataset management, the TS, RMS and DMS system needs to work together with a worklow like the one presented in figure 3. In these figures other DIRAC systems appear: the *Configuration System* (CS), the *Resource Status System* (RSS) [7] and the Accounting and Monitoring System. The details of these systems are not presented in this paper. More information can be found in DIRAC documentation at dirac.readthedocs.io.

DIRAC version 7, released in 2019, also introduced a new system named Production Management System (PMS), which provides a high-level interface for productions management. Details are not discussed within this paper, but documentation, like for the previously mentioned systems, can be found in the DIRAC official documentation.

5 Ongoing developments

There are multiple developments going on within the DIRAC Project in order to follow the technology evolution of distributed computing systems, add new functionalities and improve the overall quality of the software. One specific development involves moving the code base to python 3: the DIRAC pilot is already python 2 and 3 compatible, and the server code is gradually moving in the same direction. There are several other developments going on at the moment, but within this paper we concentrate on a specific one, which can be found in the next section.

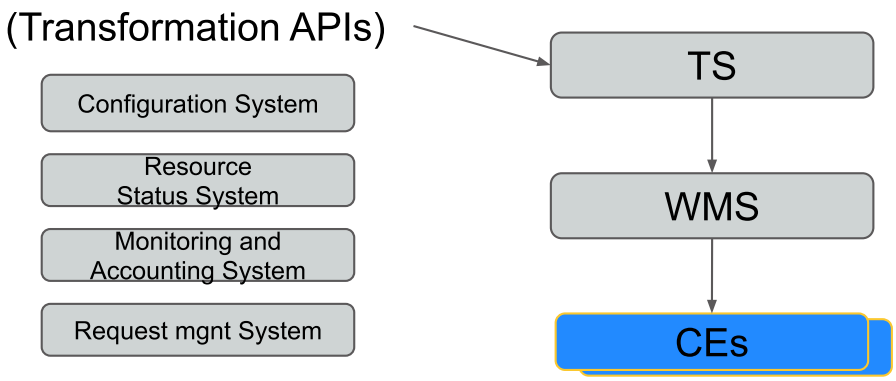

Figure 2. Productions management through DIRAC Transformation and Workload Management Systems

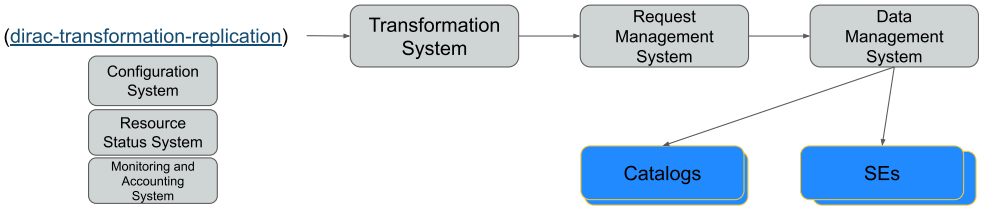

Figure 3. Datasets management through DIRAC Transformation, Request Management and Data Management Systems

5.1 Support for OAuth2/OIDC based authentication and authorization infrastructures

In most of the currently existing multi-community grid infrastructures the security of all operations is based on the X.509 standard. Each user obtains a certificate from one of the recognized Certification Authorities (CA) and registers it with a VO Management Service (e.g. VOMS [8]), which keeps the user identity information together with an associated profile for each user. In order to access grid resources users are generating proxy certificates which can be delegated to remote services in order to perform operations on the user's behalf. While DIRAC does not have a strong dependency from VOMS, the users' information stored in the DIRAC Registry can be synchronized with VOMS and mapped onto the definition of the DIRAC groups to which the user can belong. The properties of the groups define the rights that users can have when accessing DIRAC services. Users can therefore generate proxy certificates with VOMS extensions to access Grid resources.

The X.509 standard based security is well supported in academia institutions but is not well suited for all researchers, nor for all DIRAC users. On the other hand, there are well-established industry standards, developed mostly for web applications, that allow identification of users as well as delegation of user rights to remote application servers. Therefore, grid projects started migrating to (or adding) new security infrastructures based on the

OAuth2/OIDC [9] technology. With this technology, the users' registration is done by local identity providers, for example, a university LDAP index. On the grid level a Single-Sign-On (SSO) solution is provided by a federation of multiple identity providers to ensure mutual recognition of user security tokens. In particular, the EGI infrastructure has come up with the Check-In SSO service as a federated user identity provider (aai.egi.eu).

The DIRAC user management subsystem was recently updated in order to support this technology. Users can be identified and registered in the DIRAC Registry based on their SSO tokens containing also additional user metadata. The metadata defines user rights within the DIRAC framework similarly to the VOMS user profile data. When a user authenticates for the first time in the DIRAC Web Portal, she can be registered automatically to the Registry based on configurable rules for interpreting the user metadata. Alternatively, service administrators can receive the user registration request to perform extra validation of the user registration request if needed.

For each authenticated user, a session is created which keeps up-to-date metadata and access tokens to be used when accessing third party services. Another long-living session can be also created to maintain valid user tokens to be used for asynchronous operations performed on the users' behalf by the DIRAC system. This is a mechanism analogous to the MyProxy [10] service or to the DIRAC ProxyManager service. It is important to note that the DIRAC implementation of the new security framework was initially done for the EGI Check-In SSO service. However, the resulting solution is generic and can be configured to work with multiple identity providers and SSO systems.

The DIRAC service/client communication protocol is still based on the X.509 certificates. Also access to most of the grid services is based on this standard. Therefore, a mechanism to provide proxy certificates to access the services is needed. New mechanisms of provisioning proxy certificates were developed in addition to the already existing DIRAC ProxyManager service. The proxy certificates can be generated on the fly using a special DIRAC Certification Authority (CA) which can be configured for a given DIRAC installation. These proxies can only be used for internal service/client communications. Another mechanism is to use an external Proxy Provider service, which generates on the fly an X.509 proxy certificate upon a successful user authentication with a given SSO system. An example of such service is the RCAuth portal (rcauth.eu). With these mechanisms, users can get proxy certificates without the necessity to obtain an original certificate from one of the CAs. The complex authorization flow is made transparent for the user with standard Web interfaces. This includes also obtaining the proxy certificate with command line tools (dirac-proxy-init command). In the latter case, the user receives a URL of the DIRAC Authentication interface, copies it to a web browser and follows a standard login dialogue with a given SSO portal. Alternatively, the URL can be in the form of a QR code suitable for authentication with a smart phone. After the successful authentication, the proxy certificate is generated in the user environment. The user gets the command line prompt back and continues to use DIRAC services in a usual way.

6 Summary and conclusions

DIRAC aims at being a complete distributed computing management tool. For many years now it has been adopted as distributed computing system of choice by several communities.

This paper introduced the most commonly used DIRAC functionalities, but many more are provided, including a full-capable web portal. To know more about it, DIRAC developers and consortium members maintain a quite large documentation, which can be found in the official DIRAC documentation [11]. Users workshops are held once a year, developers and hackathons are run weekly. DIRAC developers put in high regards testing and automation, as well as using de-facto technology standards.

DIRAC is developed by members of a few collaborations. Notably, LHCb still maintain, to date, the largest fraction of expertise and developments.

References

[1] F. Stagni, A. Tsaregorodtsev, M.U. Garcia, P. Charpentier, K.D. Ciba, Z. Mathe, A. Sailer, R. Graciani, C. Haen, W. Krzemien et al., *Diracgrid/dirac: v6r20p15* (2018), https://doi.org/10.5281/zenodo.1451647

[2] F. Stagni, A. Tsaregorodtsev, L. Arrabito, A. Sailer, T. Hara, X. Zhang, Journal of Physics: Conference Series **898**, 092020 (2017)

[3] S. Camarasu-Pop et al., *Exploiting GPUs on distributed infrastructures for medical imaging applications with VIP and DIRAC*, in *42nd international convention on information and communication technology, electronics and microelectronics (MIPRO 2019) Opatija, Croatia, May 20-24, 2019* (2019), pp. 190–195

[4] F. Stagni, A. McNab, C. Luzzi, W. Krzemien, Journal of Physics: Conference Series **898**, 092024 (2017)

[5] various, Tech. rep. (2019), https://cds.cern.ch/record/2696169

[6] A. Kiryanov, A.A. Ayllon, O. Keeble, Procedia Comp. Sci. **66**, 670 (2015)

[7] F. Stagni, M. Ubeda, A. Tsaregorodtsev, V. Romanovskiy, S. Roiser, P. Charpentier, R. Graciani, Journal of Physics: Conference Series **513**, 032093 (2014)

[8] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lörentey, F. Spataro, *VOMS, an Authorization System for Virtual Organizations.*, in *European Across Grids Conference*, edited by F.F. Rivera, M. Bubak, A. Gómez-Tato, R. Doallo (Springer, 2003), Vol. 2970 of *Lecture Notes in Computer Science*, pp. 33–40, ISBN 3-540-21048-2

[9] D. Hardt, *The OAuth 2.0 Authorization Framework*, RFC 6749 (2012), https://rfc-editor.org/rfc/rfc6749.txt

[10] J. Caballero, J. Hover, M. Litmaath, T. Maeno, P. Nilsson, M. Potekhin, T. Wenaus, X. Zhao, J. Phys.: Conf. Ser. **219**, 072028. 6 p (2010)

[11] DIRAC, *Dirac documentation*, http://dirac.readthedocs.io/