EPJ Web of Conferences **214**, 03012 (2019) https://doi.org/10.1051/epjconf/201921403012
CHEP 2018

LHCb and DIRAC strategy towards the LHCb upgrade

Federico Stagni[1,*,**], *Andrei* Tsaregorodtsev[2], *Christophe* Haen[1], *Philippe* Charpentier[1], *Zoltan* Mathe[1], *Wojciech Jan* Krzemien[3], and *Vladimir* Romanovskiy[4]

[1]CERN, EP Department, European Organization for Nuclear Research, Switzerland
[2]Centre de Physique de Particules de Marseille CPPM, France
[3]National Centre for Nuclear Research, Swierk, Poland
[4]Institute for High Energy Physics of NRC Kurchatov Institute, Russia

Abstract. The DIRAC project is developing interware to build and operate distributed computing systems. It provides a development framework and a rich set of services for both Workload and Data Management tasks of large scientific communities. DIRAC is adopted by a growing number of collaborations, including LHCb, Belle2, CLIC, and CTA. The LHCb experiment will be upgraded during the second long LHC shutdown (2019-2020). At restart of data taking in Run 3, the instantaneous luminosity will increase by a factor of five. The LHCb computing model also need be upgraded. Oversimplifying, this translates into the need for significantly more computing power and resources, and more storage with respect to what LHCb uses right now. The DIRAC interware will keep being the tool to handle all of LHCb distributed computing resources. Within this contribution, we highlight the ongoing and planned efforts to ensure that DIRAC will be able to provide an optimal usage of its distributed computing resources. This contribution focuses on DIRAC plans for increasing the scalability of the overall system, taking in consideration that the main requirement is keeping a running system working. This requirement translates into the need of studies and developments within the current DIRAC architecture. We believe that scalability is about traffic growth, dataset growth, and maintainability: within this contribution we address all of them, showing the technical solutions we are adopting.

1 Introduction

The aim of this paper is to show how DIRAC [1] is being developed with scalability and flexibility in mind, with the idea of satisfying the needs of LHCb [2], the user that mostly stresses today's DIRAC functionalities.

1.1 The DIRAC project

The DIRAC project enables communities to interact with distributed computing resources. It forms a layer that hides diversities across computing, storage, catalog, and queuing resources. DIRAC has been adopted by several HEP and non-HEP experiments' communities, with

*e-mail: federico.stagni@cern.ch
**On behalf of the LHCb collaboration.

different goals, intents, resources and workflows. DIRAC is experiment agnostic, extensible, and flexible.

DIRAC is a truly open source project: it has been started as an LHCb project, and following interest of adoption from another 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 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.

1.2 The LHCb upgrade and DIRAC

The LHCb Upgrade [2] will operate through LHC Runs 3 and 4. The instantaneous luminosity will increase by a factor of five, from 4×10^{32} to $2 \times 10^{33}\text{cm}^{-2}\text{s}^{-1}$. The event selection and processing at the software level with this increase have a major impact on software and computing systems.

LHCb has implemented an extension of DIRAC, called LHCbDIRAC [3, 4] and uses fully the functionalities that DIRAC provides. LHCb has customized some of the DIRAC systems, and created two new ones: the Bookkeeping [5] (a provenance and metadata catalog) and the Production Management [6] (a high-level system for managing all LHCb productions). They both provide Web User Interface (WUI) extensions within the LHCbWebAppDIRAC (the LHCb extension of WebAppDirac, which is the DIRAC portal, based on Tornado [7]). The Pilot project is extended within the LHCbPilot project.

The LHCb computing team wrote a computing TDR for the LHCb upgrade [8]. In that document, we recognized the need for two complementary adaptation strategies: a revolution in some part of the LHCb software, and an evolution for other parts. DIRAC and LHCbDIRAC fall in the second category, and within this paper we will explain why. With DIRAC, LHCb operates a service, and it is mandatory for us to keep a running system working, with continuity. It will become clear, by reading further, that, while several changes will be needed to DIRAC and LHCbDIRAC, we don't see the need for a revolution, so the system will keep evolving gradually, in a backward compatible way.

1.3 Paper organization

This paper is organized as following: section 2 shortly explains the versatility of the DIRAC pilots, used to exploit all computing resources. Section 3 will give details on how we define scalability challenge and how we want to address it. Section 4 will hint some details on user analysis for LHCb within the upgrade era. Finally, conclusions are given in the section 5.

2 Exploiting computing resources

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

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

WMS software. The "push" model proved to be inefficient and error prone. To face these issues DIRAC introduced, back in 2006, the so-called pilot jobs, which are not real payload jobs but rather the startup scripts which land at the worker node, perform sanity checks and then pull payload jobs from the central queue: this action is often called "matching" a job. 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 manage the heterogeneous computing resources presenting them to the central services in a uniform coherent way. Each LHC 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) is changing 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) [9]. 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 an ongoing activity for 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 DIRAC pilot has the following characteristics:

- a DIRAC pilot is what creates the possibility to run jobs on a worker node;
- it is a standalone python script, 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 Machine, or whatever 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 (a Red Hat Enterprise Linux derivative).

The transparent access to the underlying resources is realized by implementing the pilot model. A scalable monitoring system based on the Message Queue (MQ) architecture is foreseen to be included as an option to the Pilot code. The monitoring system running as early as the pilot scripts itself, before any job matching, will be able to provide information about the WN environment and possible errors occurring during the installation and configuration phases, by sending logging messages to the dedicated servers.

3 System scalability

We think that scalability issue covers more than one area. Within this paper we approach scalability by dealing with the following points:

1. *traffic growth*, meaning increase of query rate;
2. *dataset growth*, meaning increase of data volume;
3. *maintainability*.

We also think that scalability solutions should not affect end users' functionalities. It is mostly about introducing those new/better/faster solutions, that can scale up your system, without users noticing (most of) them. At the same time, scalability should end up improving the users' perception of the system, that it should feel smoother and faster. All of this without lowering the usability for the end users.

3.1 Traffic growth

The DIRAC architecture is a web service architecture. In this sense, approaching traffic growth means, mostly, making sure that messages can be treated by the system in a reasonable time, and that the system can react to a growing number of messages without being affected, performance-wise.

Web services based software architectures have been an exhaustively discussed topic in the vast web community. Various architectures have been proposed and implemented within the HEP community, but mostly outside of it. The DIRAC architecture and framework have been designed and implemented more than ten years ago. So, first of all we asked ourselves whether the DIRAC architecture needed, or not, an upgrade.

3.1.1 DIRAC *architecture analysis*

The basic DIRAC components are *Services*, *Agents*, and *Executors*.

- *Services* are passive components listening to incoming client requests and reacting accordingly by serving requested information to or from the Database backend. Services themselves can be clients of other Services. DIRAC provides the end users and administrators with dozens of specialized services.
- *Agents* are active components, similar to cron jobs, which execution is invoked periodically. Agents are animating the whole system by executing actions, sending requests to the DIRAC or third party services. DIRAC provides the end users and administrators with dozens of specialized agents.
- *Executors* are also active components, similar to consumers of a message queue system, which execution is invoked at request. Executors are used within the DIRAC Workload Management System (WMS).

DIRAC components are combined together to form *Systems*. A DIRAC system is delivering a complex functionality to the rest of DIRAC, providing a solution for a given class of tasks. Examples of systems are the Workload Management System (WMS) or the Configuration System (CS) or the Data Management System (DMS). And then there are databases, which keep the persistent state of a System. They are accessed by Services, Agents and Executors as a kind of shared memory.

A trendy approach for modern Service Oriented Architectures (SOA) is the so-called "microservices" based architecture. Within a microservices architecture services are small, self-contained, single-responsibility units that can be independently developed, tested, and deployed. DIRAC services maintain also these characteristics.

The vast majority of DIRAC services can be easily duplicated, meaning that more than one instance of the same type can run. This also is true for DIRAC executors. In contrast,

the DIRAC agents cannot be duplicated in a similar, simple way. This limitation imposes the requirement of the only architectural change of DIRAC, which otherwise will largely stay the same as of today.

3.1.2 *The* DIRAC *framework*

The DIRAC Core and Framework has been developed more than 10 years ago, and it covers several features on top of which DIRAC components rely:

- logging: the DIRAC logging is used for each DIRAC component, including the commands
- threadpools and processpools implementations
- DISET, the heart of the DIRAC communication protocol *dip*, provides implementation for:
  - Securing sockets with SSL
  - marshalling and un-marshalling, done through a tiny DIRAC specific library called DEncode

DIRAC implements within DISET its own RPC protocol, which can be summarized very briefly as Python sockets, secured with SSL, and using DEncode for the marshalling. It is an extremely light protocol that maximizes speed and was preferred, at the time, to the XMLRPC implementations that were native of Python.

Most of the DISET functionalities have been developed long time ago, but nowadays all the aforementioned libraries are available and maintained elsewhere. Currently the adaptation of DIRAC code to fully use the standard Python libraries is ongoing. This under-the-hood work takes long time and will significantly reduce DIRAC's codebase.

DIRAC also started exploratory studies towards using HTTPS (instead of *dips*), to allow usage of standard libraries. HTTPS is widely used, it has a big community with a low risk of security breach, and for which it is easy to get help. Many conventions already exists to send data (like JSON or multipart/form-data), and frameworks already exist in python 2 and 3 for server-side (e.g. Tornado) and client side (e.g. requests [10]).

For what regards the aspects of authentication (and partially of authorization), DIRAC currently relies on openSSL [11] and x509 certificates. DIRAC makes use of a quite thin layer on top of openSSL, named pyGSI[2], to create proxy certificates and secure the RPC calls. This package has been created by DIRAC developers long time ago, and it is part of DIRAC's external packages bundle. We would like to replace it with the M2Crypto[3] package, keeping OpenSSL and X509 as security means.

3.1.3 DIRAC *and Message Queues*

Message Queues systems gained popularity for their simplicity and scalability. DIRAC can interact with some Message Queues systems using the stomp protocol[4], but the same time the DIRAC infrastructure does not require, at the moment, Message Queueing systems.

[2] https://github.com/DIRACGrid/pyGSI
[3] https://gitlab.com/m2crypto/m2crypto
[4] https://stomp.github.io/

3.1.4 DIRAC *and orchestrators*

Up to now we talked about scalability from the pure DIRAC software point of view, while we did not yet discuss how orchestrator software could be used for administering DIRAC's components. We made an extensive trial to use them, and achieved partially satisfactory results: on one side, high availability, scalability and ease of administration are achieved. However, a lot of training to maintain the underlying infrastructure is needed, some methodologies might need to be adapted, and client traceability is partially lost at the level of the application [12]. We will go back to this topic once some of the developments discussed above will be completed.

3.2 Dataset growth

The vast majority of DIRAC relational databases have been implemented using MySQL as a backend. At the same time DIRAC can also interact with Oracle, and within LHCbDIRAC we have implemented the Bookkeeping database within Oracle. We will keep using these RDBMS solutions, and we do not see any need to integrate more; recently, new players came on the market (e.g. cockroachDB[5], crate.io[6], or clustrix[7] are just some of them) but in our humble opinion it is too early to start thinking about investing time in exploring these alternatives.

We have at the same time introduced ElasticSearch[8] as a NoSQL Database and search engine for some specialized purposes e.g. for monitoring.

3.3 Maintainability

We believe that the scalability issue cannot be properly addressed if the system, and the software, are not easily maintainable. Throughout these years we invested into applying industry-standard software engineering practices that help developing and maintaining DIRAC.

Recognizing the importance of the fact that DIRAC is an open source software used by several communities with different goals and computing models, we invested into having code that is easily extendable, both horizontally and vertically [13].

3.3.1 Testing, DevOps and Continuous Delivery

Writing a good test suite and a testing process is as much work as the development of the system itself, and we believe that an extensive testing phase, for a project like DIRAC, is an absolute must. For this reason, in the past few years we invested in the developement of what we called the "certification process": the certification is a lengthy, slowly automatized process of checking the quality of DIRAC software. It consists of static code analyses, unit tests, integration tests, regressions tests, system tests, and performance tests. The certification process is applied to each DIRAC minor and major releases, and it is paying off in terms of software quality and correctness. While we use and apply continuous integration techniques, we are still far from adapting the Continuous Delivery approach.

3.3.2 Documentation

We regard documentation as a very important part of the code development process. DIRAC's documentation is automatically built, and versioned together with the code.

[5]https://github.com/cockroachdb
[6]https://crate.io/products/cratedb/
[7]https://www.clustrix.com/
[8]https://www.elastic.co/products/elasticsearch

3.3.3 Code maintenance, and Python 3

DIRAC is developed in python 2.7. We started looking into migrating the code to python 3, even though we do not yet have a target release. We have been polishing the code for long time now, in order to make the migration process simpler. We are aware that wide, deep, testing is fundamental, and the testing and certification process which we have in place described in section 3.3.1 is fundamental.

DIRAC anyway relies on several packages, that also would need to be moved to python 3.

4 LHCb User analysis for the LHCb upgrade

Everything that is done for LHCbDIRAC will also benefit LHCb user analysis. The Ganga tool [14] is the software of choice for LHCb users-driven analysis jobs. Ganga is a frontend to users' jobs submission, and relies on LHCbDIRAC for every interaction with LHCb distributed computing. Software maintenance, and continuous users feedback is needed for adapting to LHCb users needs in the upgrade era, and possible changes in DIRAC APIs and in the LHCb core software. Working Group productions [15] are encouraged as a good approach for doing analysis on the Grid where requests, coming from the working groups, are managed by the central production team. Working Group productions do not eliminate fully the need for private analysis jobs on the Grid, but are thought as a better organized way of running large sets of analysis jobs.

Individual users will still need to submit jobs to test their code before going for centralized productions, or even to launch their private production (even though the latter will be unpractical, due to the expected size of the datasets). The Ganga tool was used in Run 2 for that, and will be used in Run 3 as well. The possibility of slimming down Ganga by keeping only the features relevant to LHCb should be explored, in order to overcome possible scalability bottlenecks that might appear in case of large user productions.

5 Conclusions

LHCb uses DIRAC for all its distributed computing activities, and will keep using it for Run3 and beyond. Many other communities use DIRAC for managing their own distributed computing processes, and some of them contribute to DIRAC's codebase as active software developers. Still, LHCb maintains a large fraction of the DIRAC code, and it is the community that stresses its capabilities the most. The LHCb upgrade should be seen not only as a challenge, but also as an opportunity to furher develop and improve DIRAC.

Our focus is on usability, flexibility and scalability. We believe that DIRAC has already proven, within the past years, to be a flexible tool, capable of satisfying the needs of a growing number of users. This contribution focuses on scalability: we address it considering traffic and dataset growth, and maintainability. We believe that DIRAC, and its extension LHCbDIRAC, will not need to go through a revolution to sustain the increased load of LHCb: rather, our strategy, which we defined already a few years ago, relies on the constant evolution, that we outlined within this paper.

Several of the developments mentioned throughout this contribution have already started, some of them are already completed, and others are yet to start. What LHCb is doing, and will do for DIRAC, will also benefit all other communities that nowadays use this platform.

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] Tech. Rep. CERN-LHCC-2012-007, CERN, Geneva (2012)

[3] F. Stagni, P. Charpentier, R. Graciani, A. Tsaregorodtsev, J. Closier, Z. Mathe, M. Ubeda, A. Zhelezov, E. Lanciotti, V. Romanovskiy et al., Journal of Physics: Conference Series **396**, 032104 (2012)

[4] LHCb, *Lhcbdirac* (2018), https://doi.org/10.5281/zenodo.1451768

[5] Z. Mathe, P. Charpentier, Journal of Physics: Conference Series **513**, 042032 (2014)

[6] F. Stagni, P. Charpentier, the LHCb Collaboration, Journal of Physics: Conference Series **368**, 012010 (2012)

[7] M. Dory, A. Parrish, B. Berg, *Introduction to Tornado* (O'Reilly Media, Inc., 2012), ISBN 1449309070, 9781449309077

[8] Tech. Rep. CERN-LHCC-2018-007, CERN, Geneva (2018)

[9] Tech. rep. (2005), https://cds.cern.ch/record/2220712

[10] R.V. Chandra, B.S. Varanasi, *Python Requests Essentials* (Packt Publishing, 2015), ISBN 1784395412, 9781784395414

[11] J. Viega, M. Messier, P. Chandra, *Network security with openSSL: cryptography for secure communications* (" O'Reilly Media, Inc.", 2002)

[12] C. Haen, B. Couturier, Journal of Physics: Conference Series **898**, 092016 (2017)

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

[14] R. Currie, U. Egede, A. Richards, M. Slater, M. Williams, Journal of Physics: Conference Series **898**, 052032 (2017)

[15] L. Anderlini, M. Clemencic, A. Falabella, M. Fontana, B. Sciascia, F. Stagni, D. Hill, Tech. Rep. LHCb-INT-2016-029. CERN-LHCb-INT-2016-029, CERN, Geneva (2016), https://cds.cern.ch/record/2161754