The DIRAC Data Management System and the Gaudi dataset federation

Christophe Haen¹, Philippe Charpentier¹, Markus Frank¹, Andrei Tsaregorodtsev²

¹ European Organization for Nuclear Research, CERN CH-1211, Genve 23, CH
² CPPM, Aix-Marseille Universit, CNRS/IN2P3, Marseille, France
E-mail: christophe.haen@cern.ch

Abstract. The DIRAC Interware provides a development framework and a complete set of components for building distributed computing systems. The DIRAC Data Management System (DMS) offers all the necessary tools to ensure data handling operations for small and large user communities. It supports transparent access to storage resources based on multiple technologies, and is easily expandable. The information on data files and replicas is kept in a File Catalog of which DIRAC offers a powerful and versatile implementation (DFC). Data movement can be performed using third party services including FTS3. Bulk data operations are resilient with respect to failures due to the use of the Request Management System (RMS) that keeps track of ongoing tasks.

In this contribution we will present an overview of the DIRAC DMS capabilities and its connection with other DIRAC subsystems such as the Transformation System. This paper also focuses on the DIRAC File Catalog, for which a lot of new developments have been carried out, so that LHCb could migrate its replica catalog from the LCG File Catalog to the DFC. Finally, we will present how LHCb achieves a dataset federation without the need of an extra infrastructure.

1. Introduction

Like any other modern particle physics experiment, LHCb[1] faces multiple challenges when it comes to the data management aspect. The amount of data produced by the detector is such that a naive approach to storage is not enough, and one needs well defined policies for data handling, complex systems replicating the data all over the world, keeping a clear track of each individual file, guaranteeing the highest availability possible, but also allowing physicists to forget about the files themselves to focus only on the dataset that they want to analyze.

In order to face these challenges, LHCb uses and develops the DIRAC interware[2]. DIRAC is an integrated Data and Workload management system. This paper presents the various components of the Data Management system of DIRAC and how it tackles the problems mentioned earlier.

2. LHCb Data Management

During the first running period of the LHC, LHCb has produced 1.1 million unique raw physics data files, which amounts to 2.8 PB. These files undergo several processing steps on the Grid,
and every file is replicated multiple times. In total, LHCb handles about 10 million unique production files and 7.4 million user files.

2.1. Logical namespace

DIRAC, internally, as well as the users, uses a logical namespace to address files[3]. In this logical namespace, each file is uniquely identified by a so called Logical File Name (LFN). The LFN is constructed using various pieces of information related to the data contained in the file, such as the year or the run number, for example /lhcb/MC/2012/ALLSTREAMS.DST/00001075/0000/00001075. This name however must only be unique and is irrelevant for any other purpose but providing a unique file identifier.

2.2. Physical namespace

Each LFN can have multiple replicas, i.e. physical files. These files are stored on Storage Elements (SE). Each SE is defined in the DIRAC Configuration System (CS) as a storage endpoint as well as multiple parameters that allow to physically access the file, such as a port number, a base path, a protocol, etc. Hence, a given replica of a file is uniquely identified by the pair (LFN, SE). LHCb decided to physically store the files in a way that the local path on the storage will always finish with the LFN. In this way, the URL to access a replica can be dynamically constructed at run time using the pair (LFN, SE) and the information about this SE stored in the CS. This avoids storing information about replicas that would need to be updated every time the storage endpoint changes configuration or offers a new access protocol.

Information about files are stored in Catalogs. LHCb uses two catalogs in parallel.

2.3. Bookkeeping catalog

The Bookkeeping is a homemade catalog relying on an Oracle database[3][4]. It is used as a provenance catalog, for production files only. For each individual file, the Bookkeeping provides information regarding:

- From which file(s) this file is derived.
- What software versions were used to produce it.
- The physics quality of the data.
- The integrated luminosity contained.
- Whether it has any physical replica.
- etc

The Bookkeeping is also used to group files in datasets that correspond to specific conditions and periods of data taking (or simulation conditions), and that have been processed in similar conditions. It is this dataset description which is used by physicists for selecting the set of files they want to analyze. A dataset is identified by a Bookkeeping path containing the origin of the data, the conditions (beam energy, magnetic field polarity, detector configuration, etc), the processing chains (reconstruction applications, physics selections, etc), the event type and the file type. Each dataset consists of a set of LFNs whose path is not related to the dataset path. Only the dataset path has a meaning for physicists.
2.4. Replica Catalog

The replica catalog is simply a list of all the LFNs known to LHCb, regardless of whether they were provided by the production system or by users. It associates to each LFN some metadata, such as a checksum, a size, a creation date, etc, as well as a list of SEs where a replica can be found. No access URL needs to be stored in the catalog because of the convention followed by LHCb described in 2.2. The Replica catalog is used mostly for job brokering. The LCG File Catalog (LFC) was used until recently[3], and has now been replaced by the DIRAC File Catalog (DFC), as described in 5.

2.5. DIRAC Data Management System

All the complexity related to this dual aspect of files (logical vs physical) is hidden from the users and internals of DIRAC by the Data Management System. This system offers to the outside a simple interface which corresponds to all the traditional data management related activities, such as adding, replicating or removing files or replicas. The input parameters to these methods are always from the logical namespace, and the physical namespace is only used internally by this system.

The Data Management implementation described in Fig. 1 offers several layers of abstraction that allow to:

- Seamlessly replace the catalog we are using, or use multiple catalogs at the same time.
- Transparently and with very limited changes support more access or transport protocols.

![Diagram](image)

**Figure 1.** The DIRAC Data Management System implementation.

The FileCatalog object abstracts all the interactions with any catalog, the StorageElement object abstracts all the communications with the storage endpoints, and the DataManager object contains the logic of operations that require to interact with catalogs and SEs (e.g. replicating a file means to physically copy it and then register the new replica in the catalog).

While simple and small data operations can be achieved using a set of scripts provided with DIRAC and its LHCb-specific extension LHCbDIRAC, production level activities need some more powerful tools, described in the following sections.

3. Transformation System

The Transformation System[4] (TS) pictured in Fig. 2 is used to automate common tasks related to the production activity, be it related to data processing or data management. For
data processing, the production manager uses a web application and predefined templates to describe the processing to be performed and the datasets, expressed as Bookkeeping queries (see subsection 2.3), on which the processing should be applied. These descriptions are stored in a database table, the 'Input Data Query Table'. The datasets are then resolved and transformed into a list of LFNs that will fill the 'Production Input Files Table'. Similarly the data manager can create a Data Management transformation, specifying the input dataset and the operation to be performed. Based on this information, each transformation is creating a set of Tasks to be performed in order to achieve the result. The task creation is performed by a dedicated plugin. Each Task is finally transformed into a Job that will actually perform the work in the case of data processing and a data management request (see section 4) for data management.

![Diagram of the DIRAC Transformation System](image)

**Figure 2.** The DIRAC Transformation System.

All this procedure is totally automated by means of several agents that take care of the various transitions described previously.

### 4. Request Management System

The Request Management System (RMS) of DIRAC, described in Fig. 3, is a very generic system which aims at executing asynchronously virtually any type of action. It is currently used by the Transformation System (see section 3) to execute large data management activities, and by jobs as a failover solution. For example, if a job cannot contact the final destination storage to upload its output data, it will store the file on a local storage, and issue a Request to the RMS so that it transfers it later. Even though they represent the vast majority of actions, the
RMS is not limited to data management operations, and can be used for very heterogeneous activities, such as monitoring information forwarding or even as an authentication gateway.

![Diagram of the DIRAC Request Management System](image)

**Figure 3.** The DIRAC Request Management System.

An action to be executed by the RMS is described as a Request object which contains an ordered list of Operation objects to be possibly applied on File objects. The Request object contains meta information about the action to perform, such as the Distinguish Name (DN) to be used or the component requesting the action. The actual tasks to be performed are described by the list of Operation objects. Each Operation has a Type and all the necessary data it needs to be executed. Examples of Operation Types are 'ReplicateAndRegister', 'ForwardDISET', 'UploadLog', 'RegisterFile', etc. Finally, if the Operation Type is a data management activity, a list of File objects is associated to the Operation.

The RequestExecutingAgent will pick up Requests in the database and execute them in parallel. The code is very generic and easily modular: adding a new Type of Operation simply means adding a new plugin and declaring it in the DIRAC Configuration System.

When the Operation consists in replicating a file, another DIRAC system is used: the DIRAC File Transfer system (FTS system), also visible in Fig. 3. The execution of the replication is delegated to this system, which abstracts the communication with the FTS servers. This system is compatible with both FTS2 and FTS3 servers [5]. Once the replication has been performed, a callback is triggered in the RMS, and the execution of the Request continues. For the time being, the FTS System is only used for replication of production files. However, we plan to use it for user files as well in the coming release.

5. DIRAC File Catalog

The LFC service as currently existing is coming to an end of life. Thus, LHCb had to find a replacement for it, and an obvious option was the catalog integrated to DIRAC: the DIRAC
File Catalog (DFC).

The DFC was first presented at CHEP 2012[6]. It was designed using the practical experience of the LHCb Data Management System and the use of the LFC, and was optimized for bulk operations. The DFC is fully integrated into DIRAC and uses its framework protocol called DISET. It aims at being a replica catalog together with a provenance catalog (see Fig. 4). Among its interesting features, one can mention:

- User defined metadata: possibility for users to define their own metadata to be associated with their files.
- Storage usage reports: the DFC keeps an accounting of the number of files and their size per storage and per directory, which is useful for strict quota enforcement.
- Web interface: the DFC comes with a web interface that allows directory browsing.

![Figure 4. The DIRAC File Catalog.](image)

The DFC is already used in production at several places, such as ILC, CLIC or BES II (see [7],[8],[9]), and hence seemed to be suitable to replace the LFC in LHCb. As such, only the replica catalog functionality is relevant for LHCb, since the provenance catalog remains the Bookkeeping.

5.1. DFC for LHCb

In order to check whether the DFC was adapted to our needs, we started by evaluating the amount of information that will be stored in it and the load against it we expect by the end of the second running period of the LHC, that is around 2018. Unfortunately, it turned out that the DFC would not cope with the load, and needed some adaptations.

Our requirements were as follows:
• Load at the end of Run 2: 20M directories, 100M files, 200M replicas.
• Strong consistency between the entities guaranteed (e.g. no file can belong to a non-existing user).
• Hosted on MySQL, preferably the CERN MySQL Database on Demand service.

The code of the DFC follows the Aspect Oriented Programming (AOP). Roughly speaking, AOP consists of splitting the program’s code and logic according to so-called ‘concerns’. The concerns are loosely coupled to each other, and the clients talk to a unified and abstract interface. In the case of the DFC, one aspect would be the directories management, another one would be the users management, another one the files management, and so on, as described in Fig. 5.

![Figure 5. The DFC Aspects.](image)

Thanks to the AOP approach followed by the DFC, the developments needed by LHCb were relatively limited:

• New aspects only for the directories and the files management, without having to touch anything related to the other aspects.
• Introduction of Foreign Keys in the database. This strongly couples the different aspects at the database level, however this is a price that we were willing to pay in order to ensure the consistency of the data.
• Usage of stored procedures for performing the actions. One advantage of this is the small gain in performance one can obtain, since the procedures are compiled ‘once’ only, contrary to SQL statements started from the code. But the main advantage is that manual interventions on the database, if necessary, become much easier since adding/removing directories or files require interactions with several tables.

We ran extensive testing campaigns against the DFC. The tests consisted in running for a long period against the DFC at least the peak load observed against our LFC, with an activity shape similar to the real one. Also the spreading of files in directories and depths of paths was inspired by the production system. The conditions were as follows:
- Access pattern
  - Read: 90%
  - Write: 5%
  - Delete: 5%
- Bulk operations: between 1 and 1000 files

The tests were run on a standard 12 core machine with 48Gb of memory and local disk without RAID setup. Since the production instance is hosted on the CERN Database On Demand instance for which we have no control on the hardware, the raw performance numbers coming out of these tests have little importance, but we were interested in seeing their evolution as the amount of data increases. The DFC was loaded with two different loads (see table 5.1): one that is similar to our current LFC (plots on the left) and one that we expect at the end of Run 2 (plots on the right). The

| Load         | Directories | Files | Replicas |
|--------------|-------------|-------|----------|
| Now (left)   | 6M          | 13M   | 25M      |
| Run2 (right) | 22M         | 73M   | 365M     |

Table 1. Loads to test the DFC.

Figure 6. Insertion performance

The main and very positive lesson of Figs. 6,7 and 8 is that the performance of the DFC are not dramatically impacted by an increased amount of data, and thus we are confident in using the DFC at least until the end of Run2.

5.2. LFC to DFC migration

Thanks to the abstraction layers in DIRAC presented in subsection 2.5, switching from the LFC to the DFC is not an issue from the code perspective. However, migrating the existing data from one to the other represents a major operational challenge. This operation cannot be performed in one go and required several steps:

- Cleanup of the LFC. A careful analysis of its content showed that many data were obsolete, but more importantly that there were inconsistencies. Since the DFC ensures the consistency, the data could not be migrated as such and required a cleanup first.
Perform the actual migration, which will be described further in this subsection.

Keep the synchronization between the LFC and the DFC for a few months. One interest of this is that in case we notice a major problem with the DFC, we could always just fallback on the LFC and reattempt the migration later. The second interest is that we have a comparison model, and we can check whether the DFC properly executes the operations. Keeping the synchronization is technically simple, again thanks to the design of the DIRAC DMS.

Totally retire the LFC.

Various scenarios were considered to perform the actual data migration from the LFC to the DFC. The criteria taken into account in order to make a choice include the time it would take, the consequence of an error, the impact on the rest of the system and the complexity and reproducibility of the procedure. The approach we decided to take is described in Fig. 9.

Stop all the writes. This allows to “freeze” the LFC database for the time of the migration.

Keep the LFC in read mode. Read operations do not harm the migration procedure, and allow users and jobs to still perform some work.
Copy the data from the Oracle LFC schema to the MySQL DFC schema. This is done using a tailored Python and SQL script that performs all the necessary schema conversions.

- Reopen the LFC and the DFC in write. Any new item inserted is present in both catalogs.
- Open the DFC and close the LFC in read. In this way, we are sure to notice problems if the write operations were not executed properly against the DFC.

From the operational point of view, we could have left the system running. Indeed, at no point in the procedure are the read operations disturbed; For the write operations, the job have failover capabilities, and they would simply have created a Registration request (see section 4) when failing to register their output data, that would have been executed later, since the RMS was stopped during the migration. However, as a safety measure, we decided to slow down the production activities during the migration. New jobs could be submitted, but they were not started on the grid; old jobs were left untouched and went through the failover mechanism described previously.

All in all, the migration procedure took about 12 hours, from slowing down the services to going full steam again, including all the necessary sanity checks that such an important operation requires. There was no noticeable accident during the plan execution, and the restart went smoothly as well. We can consider this a major achievement from the operational perspective. We have been running the DFC in production since February 2015, and are generally happy with it. We encountered very minor issues due to slight differences between the APIs of the LFC and DFC. A more problematic aspect is the authorization management. Indeed, while the LFC relied on VOMS roles, the DFC only has knowledge about the DIRAC users and groups. This results in some mapping problems, that are currently being worked out.

6. LHCb Dataset federation

The aim of a dataset federation is to present a unified view of all the files and allow access to any replica of a file in a seamless manner, including failover strategies. This strategy is praiseworthy, especially in our grid context where experiments tend to broker the jobs where the data required by a job is parked. Typically, the jobs are brokered using the information contained in the replica catalog. The advantage of a dataset federation is that in case the local replica is not
available (either due to temporary problems on disk server or service overload), the job could fallback on another replica.

One method to achieve this result that seems to be popular lately is to have a central service such as the xrootd or the HTTP federation. Basically, when the job fails to open the local replica, it will ask this central service for an alternative one. The central service will query all the storage element endpoints to see if another replica is available at this very moment. Once a proper replica is found, redirection occurs such that the job can keep running on the remote file. The drawback of this method is the need of a central service, that represents a single point of failure, and could become resource hungry depending on the scale of the experiment.

LHCb decided to go for an alternative approach. Since all the required information concerning all the replicas of a file can be obtained from the various tools presented in this paper, we decided to provide the job with all the data location information. This takes the form of an XML file which contains the URLs for all the replicas of the files required by the job, starting with the local URL. All the LHCb software stack relies on a common framework, Gaudi, which actually takes care of opening the files. Ten lines of code were sufficient so that Gaudi uses the XML file and loops over all the replicas until one is successfully opened. This naturally achieves the aim of a data federation, without the need of a heavy central infrastructure with the risk of the single point of failure, with very little development and no maintenance effort. We have been successfully running this in production for more than six months now.

7. Conclusion

Data management concepts are by definition simple: replicate, add, remove, etc. The DIRAC Data Management System and the whole LHCb data management policy aims at bringing this simplicity down to the logic and the core of the system. This is reflected by the single system we have for handling both productions and bulk data management operations, the very generic approach taken by the Request Management System that serves as a failover solution for jobs and as an asynchronous executor for any operation, or the dataset federation achieved with only ten lines of code. Keeping the logic simple is also what allows LHCb to run with a very limited amount of manpower: two developers and one part time operational role for what concerns the data management.

A simple logic does not translate into simple code. Lots of effort is put into the data management code in order to provide enough abstraction layers so that the users and the rest of the DIRAC systems are totally agnostic to any change that could happen in the underlying services used by LHCb: storage access protocol, FTS servers, etc. The best illustration of this decoupling is certainly the recent and very successful migration of the LHCb replica catalog from the LCG File Catalog to the DIRAC File Catalog. There was hardly any change needed in the code to prepare the transition, and users would probably not have realized the change if we did not announce a downtime for performing the migration.

The DIRAC and LHCb Data Management Systems keep evolving in order to stick with the latest services provided such as new FTS3 features, direct XROOT or HTTP storage accesses or underlying library changes like GFAL. Simplicity of use remains the primary criterion in our development. A good example is a new data popularity tool developed with Yandex that aims to dynamically adapt the number of replicas of each dataset based on their popularity among
physicists. This will unload the person in charge of the data management operations while allowing us to have a better usage of our disk and tape pledges.

References
[1] The LHCb Detector at the LHC, Alves, A. Augusto and others 2008 JINST S08005 doi:10.1088/1748-0221/3/08/S08005
[2] Status of the DIRAC Project, A Casajus et al 2012 J. Phys.: Conf. Ser. 396 032107 doi:10.1088/1742-6596/396/3/032107
[3] The LHCb Data Management System, J P Baud et al 2012 J. Phys.: Conf. Ser. 396 032023 doi:10.1088/1742-6596/396/3/032023
[4] The LHCb DIRAC-based production and data management operations systems, F Stagni et al 2012 J. Phys.: Conf. Ser. 368 012010 doi:10.1088/1742-6596/368/1/012010
[5] FTS3: New Data Movement Service For WLCG, A A Ayllon et al 2014 J. Phys.: Conf. Ser. 513 032081 doi:10.1088/1742-6596/513/3/032081
[6] DIRAC File Replica and Metadata Catalog, A Tsaregorodtsev and S Poss 2012 J. Phys.: Conf. Ser. 396 032108 doi:10.1088/1742-6596/396/3/032108
[7] Physics and Detectors at CLIC: CLIC Conceptual Design Report, L. Linssen et al, arXiv:1202.5940. CERN-2012-003. ANL-HEP-TR-12-01. DESY-12-008. KEK-Report-2011-7
[8] DIRAC File Catalog Structure and Content in preparation of the CLIC CDR, S. Poss, LCD-OPEN-2012-009, https://edms.cern.ch/document/1217804
[9] File and metadata management for BESIII distributed computing, C.Nicholson et al, 2012 Int. Conf. on Computing in High Energy and Nuclear Physics (New York)