DUNE Database Development

Ana Paula Vizcaya Hernandez[1*] and *Lino* Gerlach[2,**] for the DUNE collaboration

[1]Colorado State University
[2]Brookhaven National Lab (US)

Abstract. The DUNE experiment will produce vast amounts of metadata, which describe the data coming from the read-out of the primary DUNE detectors. Various databases will make up the overall DB architecture for this metadata. ProtoDUNE at CERN is the largest existing prototype for DUNE and serves as a testing ground for - among other things - possible database solutions for DUNE. The subset of all metadata that is accessed during offline data reconstruction and analysis is referred to as 'conditions data' and it is stored in a dedicated database. As offline data reconstruction and analysis will be deployed on HTC and HPC resources, conditions data is expected to be accessed at very high rates. It is therefore crucial to store it in a granularity that matches the expected access patterns allowing for extensive caching. This requires a good understanding of the sources and use cases of conditions data. This contribution will briefly summarize the database architecture deployed at ProtoDUNE and explain the various sources of conditions data. We will present how the conditions data is retrieved and streamed from the databases and how it is handled to match expected access patterns.

1 Introduction

Neutrinos are the most abundant mass particle in the universe, as it has been discovered in the last half century. They can shed light on questions such as the nature of matter and how the universe evolves. The Deep Underground Neutrino Experiment (DUNE) is a next generation long-baseline experiment that uses Liquid Argon Time Projection Chamber (LArTPC) technology. It aims to study topics such as neutrino oscillations, it will search for proton decays, and it will be able to observe supernova neutrinos [1].
DUNE, slated to begin data taking in late 2028, will consist of two neutrino detectors that will be placed on the path of a high-intensity (MW-scale) neutrino beam. The near detector, at the Fermi National Accelerator Laboratory (FNAL) will study particle interactions near the source of the beam. The far detector will be installed around 1,300 km from the source at the Sanford Underground Research Laboratory. The DUNE database structure will consist of several databases containing the large range of metadata produced.
Two prototypes of the far detector are currently deployed at the CERN neutrino platform. ProtoDUNE consists of two LArTPCs [2], their main distinction being how the charge is read out and having different particle detection techniques. They are critical to demonstrate viability of LArTPC technology. They are exposed to test beams and they also take cosmic ray data.

*e-mail: avizcaya@colostate.edu
**e-mail: lino.oscar.gerlach@cern.ch

2 ProtoDUNE databases

ProtoDUNE produces a large amount of data and metadata (Sec. 2.1), and as such presents an ideal testing ground for the offline database solutions and architecture that will be used in the DUNE experiment (Sec. 2.2). Nevertheless, the amount of metadata will be much larger in DUNE, so appropriate scaling studies will be done and R&D is under way (Sec. 3.3).

2.1 Metadata and conditions data

Generally, metadata describes the data coming from the read-out of the primary detectors. It helps to identify the nature of the data, its attributes, types, etc. It can come from different sources, like DAQ readouts, slow control parameters, or calibrations, to name a few.
The subset of all metadata that is accessed during offline data reconstruction and analysis is referred to as conditions data and it will be stored in a dedicated database [3]. It can be stored and indexed by time, run, or subrun. When the metadata is time-indexed, studies must be done to select the correct granularity, or the appropriate sample rate, at which the metadata must be stored, and a weight or an average can be given. Examples of time-indexed metadata can be found in the slow controls, one such example being the high voltage. Run/subrun-indexed metadata stores one value per run/subrun. Examples of them are run/subrun DAQ configuration settings.
The period for which given metadata is valid defines a Interval Of Validity (IoV). It is applied regardless of the indexing used to store the metadata. Alignment constants and calibration parameters are an example of conditions metadata that will likely have IoVs for longer time periods than that of a run.

2.2 Database architecture

In ProtoDUNE, the metadata is stored in dedicated databases based on the information they contain. These databases include: run or DAQ configuration, slow control, offline calibration, beam data (IFBeam), data quality, and hardware databases.
The database architecture used in ProtoDUNE is as follows. The several databases are depicted in Fig. 1 where the lines represent the direction in which the metadata streams, going towards the Master Store of Metadata (UconDB). There, a subset of metadata is selected, the conditions metadata, and it is sent to the conditions database. End users finally stream the metadata with the help of dedicated applications (Sec. 3).
The purpose of Master Store of Metadata is to have a centralized place to store all the metadata that will likely fall under the conditions metadata category. Interpolation or data reduction to match the desired granularity will be done here to decrease the load on the online databases. The metadata is stored as blobs in folders specific to their original database, where each blob contains the metadata in a JSON format. The blobs can be stored with run number and time as key values and their IoVs will be specified. This database is an unstructured PostgreSQL database which has the advantage of avoiding having a priori schema, this is desirable since the conditions data will change with time.
The conditions metadata will have a dedicated database, the Conditions database, with Application Programming Interfaces (APIs) to facilitate the extraction of the metadata without the users directly interacting with the database. The users will also have the option to use the File Metadata Catalog (Metacat [4]) to access the metadata. More on this can be found in the following section.

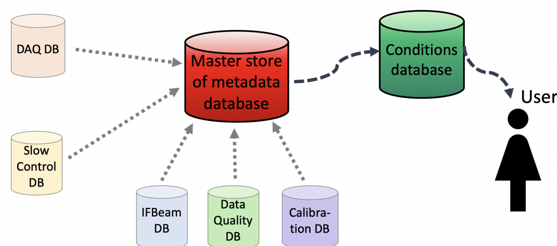

Figure 1. Diagram depicting the general architecture of the ProtoDUNE databases for offline processing. The arrows depict the direction of the metadata stream. The users access the metadata by interacting with the conditions db APIs.

3 Conditions Database

The purpose of the conditions database is to provide all of the non-primary data stream needed for offline data processing and analysis. There are challenges that need to be address when creating this database [5]. The metadata changes over time, new parameters can get added to the conditions list, or existing values of metadata can evolve, for example the calibration parameters can improve with better algorithms or the increased of the dataset. Another challenge is that the metadata is heterogeneous, in other words, there can be time-index and run-index metadata in the same database. The type and structure of the metadata can also vary, for example, the metadata can be a single number, a list of values, or a 3D map. Finally, the high access rates O(kHz). Offline data analysis is often run on distributed computing resources spread across thousands of jobs which can simultaneously request access.

3.1 ProtoDUNE conditions DB implementation

The conditions database for the ProtoDUNE experiment is a PostgreSQL relational database. It is implemented and supported by FNAL. It stores the conditions data with a run-by-run granularity, and users can access the data via python APIs or via Metacat.
After each run, the metadata is extracted from the different databases and sent to the Master Store of Metadata. A diagram of the workflow is shown in figure 2. A cron job runs every few minutes to get the DAQ metadata of the last 30 runs. The runs are compared with a list of the runs that haven't been transferred to the Master Store. If they haven't, all the metadata is extracted, a blob is created with the metadata in JSON format, and it is sent to the Master Store db. The metadata is then accessed via a python API [6] and a subset of it is sent to the conditions DB.
To access the metadata in the conditions database, users can employ a python API that allows either, direct access to the database, or remote access to the web server. The API also allows users to access the database with the command line interface [7].

3.2 Runs History Database

The purpose of the run history database, which is part of the conditions database, is to identify runs which fulfill specific configurations with the aim of file discovery. In other words, users will be able to find all the raw data files of runs where a certain configuration is applied, for example having the high voltage setting at a specified set voltage.

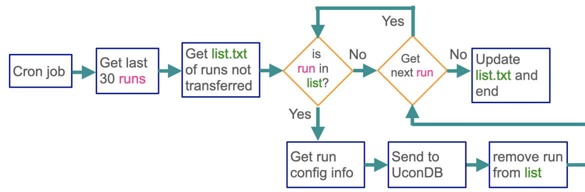

Figure 2. Diagram of the workflow to get the metadata, after each run, to the Master Store of Metadata and to the conditions database.

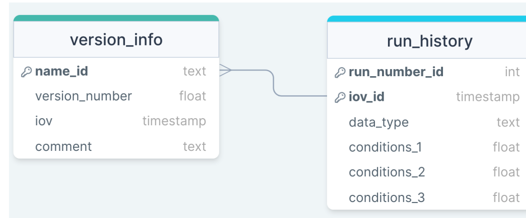

Figure 3. Runs history database schema. The *conditions_#* payloads in the **run_history** table represent the conditions parameters in the database and there are as many as there are conditions parameters. The *data_type* indicates which LArTPC the data comes from.

The chosen schema for the runs history relational database is the shown in Fig. 3. The **run_history** table has ***run_number_id*** and ***iov_id*** as key parameters, since the conditions data can change with time. The **version_info** table handles the versioning, where a tag and a name can be given to a new version. Changes of a single condition parameter in a run require a new entry in the database, the newest entry of each run is returned, except otherwise specified. A new condition metadata parameter can be added to the run history table as a new column, these will be tagged as a new table version. Nevertheless, that functionality will be left for the database experts since it will require direct access to the database.
The runs history database is then integrated with the File Metadata Catalog (MetaCat) [4] which allows for file discovery. An example query of how to find all the raw data files of runs having the high voltage set at 120 V is: `metacat filter dune_runsdb_incondb() (files from dune:all) where runs_history.hv = 120`

3.3 R&D: HSF-Inspired Conditions Database

As noted earlier, DUNE will produce significantly more metadata that will be accessed at a significantly higher rate. Therefore, the database group is exploring new approaches on conditions data handling. The concept of conditions data is something that many, if not all, High Energy Physics (HEP) and Neutrino Physics experiments share. For this reason, the High Energy Physics Software Foundation (HSF) has a dedicated 'activity' that deals with it. Its goal is to gather experience from various experiments and provide a set of best-practice guidelines. In 2019, they published a white paper [5], summarizing the typical workflows, use cases and challenges that experiments face when handling conditions data and giving the following recommendations regarding a possible experiment-agnostic implementation of a conditions database:

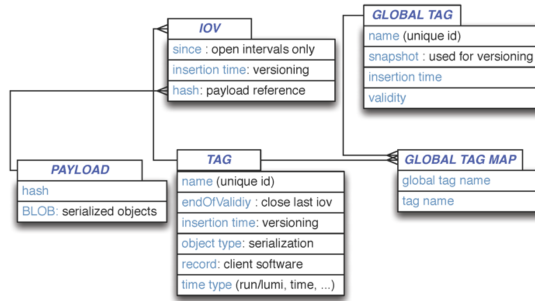

Figure 4. The data model for conditions data access management as recommended by the HSF [5].

- *Separation of payload queries from metadata queries.* This approach has several advantages. Firstly, it greatly facilitates caching. In particular when considering conditions data with greatly varying granularity where many combinations of initial parameters will eventually resolve to the same payload. Secondly, the actual payload query (a very simple mapping from a unique identifier to some chunk of data) can be handled effectively by existing solutions. The payloads can, for example, reside inside a DB as a blob or as a file on a file system. In particular, distributed file systems are well suited for this as they usually come with capable caching systems, great portability and the possibility to browse the payloads by hand without much overhead.
- *Payload agnostic design from first principles.* As mentioned in Sec. 3, the exact conditions data needed to achieve the best physics results from an experiment are often unknown a priori. A reliable, low-maintenance conditions database must take this into account. Therefore, no database schema evolution should be necessary to accommodate unforeseen changes to the nature of the conditions data. This guarantees that the system remains operational over the life time of the experiment with minimal person power from database experts.

Figure 4 shows the data access model recommended by the HSF. It takes the previously mentioned recommendations into account. The *Global Tag* is the single configuration parameter that will always resolve to the exact same set of payloads. The *Interval of Validity (IoV)* is an abstract concept based on some discrete unit of time (e.g. a run number), allowing to optimize the queries with respect to caching.

3.3.1 HSF-Inspired Reference Implementation

A reference implementation strictly according to the recommendations mentioned in 3.3 has been developed. It consists of a server-side application with the following component services:

- **nopayloaddb** REST API implemented using Django REST framework. The performance-critical read endpoint utilizes a postgres-optimized raw SQL query while other endpoints use the Django native Object Relational Model (ORM).
- **Postgres** database backend with persistent storage on NFS. Database indexes are established to enhance the responsiveness of read operations.
- **pgBouncer** serves as a database pooler, optimizing the efficiency of database connections by keeping them open for reuse by multiple requests from Django.

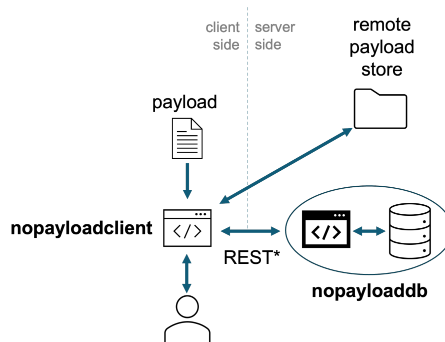

Figure 5. Overall architecture of the application. The client-side library **nopayloadclient** communicates with the server-side application, **nopayloaddb**, through a REST interface. The actual payloads reside in the remote payload storage (e.g. a /cvmfs/ [8] file system).

- **Nginx** web server.

The complete server-side application is containerized and can be deployed on a Kubernetes cluster with a single line of command, where the configuration is done via a helm chart.
A c++ client-side library, **nopayloadclient**, was also developed. It communicates with the server-side database through the REST interface, does client-side caching, and handles the payloads. One key feature is to ensure that the payload store and the database remain synchronized. The URL of individual payloads can furthermore be overridden locally, so that a user-specified file will be preferred over the one referenced in the database. This allows to locally test new calibrations before inserting them into the system indefinitely. The overall architecture of the application can be seen in Fig. 5.

3.3.2 Performance Testing

As mentioned in Section 3, very high request rates to the conditions database are expected for DUNE. Therefore, it is important to systematically test the system's performance for the expected conditions. These can be divided into two subjects: Firstly, the expected database occupancy. Secondly, the expected access patterns and rates.
In real-world usage, many (∼10,000) computing jobs running across the world-wide grids will make requests to the conditions database simultaneously. The vast majority of these jobs will access the data in a read-only fashion. More precisely, they will request the payload URLs for a given global tag, major- and minor IoV, which is why only that query was considered for the following tests. Furthermore, in order to minimize the influence of any hidden caching anywhere in the application, major- and minor IoV are chosen from a uniform random distribution between 0 and the largest possible value.
The performance of the server-side application is evaluated with the following procedure: On the client side, a random request is made. Timestamps of when the request was sent and when the answer was received are recorded. From this, the response time and -frequency can be calculated. It is worth noting that only the retrieval of the file URL through the REST API is timed. Retrieving the actual payload is not part of the performance testing. Higher request rates are achieved by submitting several grid jobs through HTCondor [9] that make subsequent requests, each. Alternatively, a single client-side node can make requests at a very high frequency with the help of asynchronous multithreading. The resulting maximal

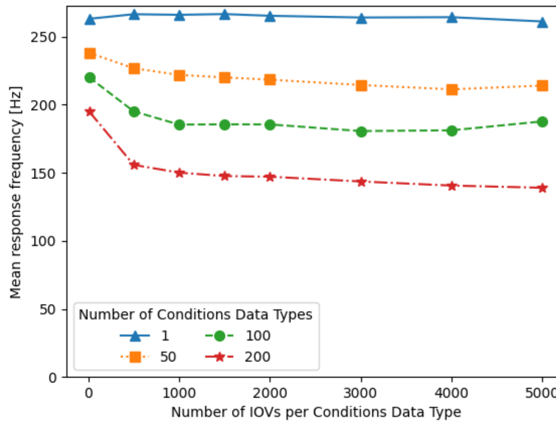

Figure 6. Mean response frequency for 10000 random requests as a function of the number of IoVs per Payload Type for different numbers of conditions data types. The last data point of the red line (dash-dotted line) corresponds to 1M entries in the queried Global Tag.

response frequency was found to be the same with both approaches, showing that the limiting factor was indeed the performance of the server-side application.
A test campaign typically consists of 10,000 random requests. Mean response time and -frequency are evaluated as a high-level metric, whose scaling with respect to various parameters can be investigated. As an example, Figure 6 shows the mean response frequency as a function of the number of IoVs per Payload Types for different numbers of Payload Types in the queried Global Tag. Other scaling tests were also conducted but are omitted in this document. For example, the number of IoVs outside the queried Global Tag was scaled up 70 million. No significant impact on the performance was observed.

3.3.3 Integration into DUNE Software Stack

The client-side library mentioned in Sec. 3.3.1 is designed to be experiment-agnostic. There is a DUNE-specific version of this library that inherits most functionality unchanged. The run-number was chosen as the major IoV. If needed, the minor IoV can be used for conditions data types that need finer granularity. The inclusion of both libraries into Fermilab's SciSoft stack has been approved and is underway. A dedicated directory in DUNE's *cvmfs* repository was created where the payloads will be stored. A test-instance of the **nopayloaddb** backend has been deployed at CERN. A first prototpe of an **art** [10] framework Service exists, although it can currently only provide the URL of the payloads. Yet to be implemented is the (partial) file reading to retrieve the data within the payload.

4 Summary

The vast DUNE metadata will be organized across various offline databases. They are specialized for different types of data as well as their typical access patterns. For ProtoDUNE, all relevant metadata is gathered in the Master Store of Metadata. It serves as the single data source for all subsequent applications and experts can browse it for detailed information. The

fraction of all metadata that is accessed by distributed computing is referred to as ‘conditions data’. Its management comes with a unique set of challenges, namely the high expected access rates in combination with the heterogeneous structure and granularity of the data. A dedicated conditions database is therefore used. It is based on a postgres backend deployed at FNAL resources. The conditions data is organized in tables and indexed by run number. Versioning of the tables helps with the configuration. In parallel, an alternative conditions database design is being investigated for future use. It is based on an experiment-agnostic conditions database developed in cooperation with the HSF and requires only minimal adjustments for usage in (Proto)DUNE. First performance tests and feedback from another experiment (sPHENIX [11]) deploying the same system show promising results.

Acknowledgements. This document was prepared by the DUNE collaboration using the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, Office of High Energy Physics HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359. This work was supported by CNPq, FAPERJ, FAPEG and FAPESP, Brazil; CFI, IPP and NSERC, Canada; CERN; MŠMT, Czech Republic; ERDF, H2020-EU and MSCA, European Union; CNRS/IN2P3 and CEA, France; INFN, Italy; FCT, Portugal; NRF, South Korea; CAM, Fundación “La Caixa”, Junta de Andalucía-FEDER, MICINN, and Xunta de Galicia, Spain; SERI and SNSF, Switzerland; TÜBİTAK, Turkey; The Royal Society and UKRI/STFC, United Kingdom; DOE and NSF, United States of America. The ProtoDUNE-SP and ProtoDUNE-DP detectors were constructed and operated on the CERN Neutrino Platform. We gratefully acknowledge the support of the CERN management, and the CERN EP, BE, TE, EN and IT Departments for NP04/ProtoDUNE-SP.

References

[1] D.U.N.E. Collabpration, *Dune, the deep underground neutrino experiment main webpage* (2020), https://www.dunescience.org/

[2] B. Abi, R. Acciarri, M.A. Acero, M. Adamowski, C. Adams, D.L. Adams, P. Adamson, M. Adinolfi, Z. Ahmad, C.H. Albright et al., *The single-phase protodune technical design report* (2017), 1706.07081

[3] T.D. collaboration, *Dune offline computing conceptual design report* (2022), 2210.15665

[4] I. Mandrichenko, *Metacat documentation* (2020), https://metacat.readthedocs.io/en/latest/

[5] P. Laycock, M. Bracko, M. Clemencic, D. Dykstra, A. Formica, G. Govi, M. Jouvin, D. Lange, L. Wood, *Hep software foundation community white paper working group – conditions data* (2019), 1901.05429

[6] I. Mandrichenko, *Ucondb client documentation* (2021), https://ucondb.readthedocs.io/en/latest/index.html

[7] I. Mandrichenko, *Condb client documentation* (2023), https://condb2.readthedocs.io/en/latest/index.html

[8] P. Buncic, C. Aguado Sanchez, J. Blomer, L. Franco, A. Harutyunian, P. Mato, Y. Yao, J. Phys. Conf. Ser. **219**, 042003 (2010)

[9] D. Thain, T. Tannenbaum, M. Livny, Concurr Comput **17**, 323 (2005)

[10] C. Green, J. Kowalkowski, M. Paterno, M. Fischler, L. Garren, Q. Lu, J. Phys. Conf. Ser. **396**, 022020 (2012)

[11] C. Dean (Sphenix), PoS **ICHEP2020**, 731 (2021)