ISSN 1547-4771, Physics of Particles and Nuclei Letters, 2026, Vol. 23, No. 5, pp. 671–673. © Pleiades Publishing, Ltd., 2026.

COMPUTER TECHNOLOGIES IN PHYSICS

Data Management System for SPD Online Filter

P. A. Korshunova[a], * and D. A. Oleynik[a], **

[a] *Joint Institute for Nuclear Research, Dubna, 141980 Russia*

**e-mail: menellyn@jinr.ru*

***e-mail: danila@jinr.ru*

Received December 5, 2025; revised March 20, 2026; accepted April 20, 2026

Abstract—This article outlines the architecture of the data management system (DMS), which is one of the core components of the SPD online filter's middleware. The DMS automates the lifecycle management of experimental data during primary processing. The article also examines the functionality of a system that emulates the operation of the data acquisition system (DAQ)—SPD DAQ emulator. It provides a continuous data stream entering the SPD online filter, which allows for load testing of all middleware.

DOI: 10.1134/S1547477126700706

INTRODUCTION

As part of the NICA mega-science project at the Joint Institute for Nuclear Research, the Spin Physics Detector (SPD) is currently under construction. A distinctive aspect of the SPD's design is the absence of a hardware trigger system for event selection, which necessitates the acquisition and handling of the complete set of produced signals. The resultant data flow is projected to be as high as 200 PB per year [1]. In order to reduce the amount of data for long-term storage and subsequent processing and analysis, it is necessary to design a software trigger—SPD online filter.

The SPD online filter should perform primary data processing, one of the aims of which is to reduce the amount of data for long-term storage and further precise processing and analysis [2]. To automate the primary processing, middleware is used, which contains the following components: a data management system, a workflow management system, and a workload management system [3]. Subsequent sections of this article will focus on the data management system.

PURPOSE OF THE DATA MANAGEMENT SYSTEM

The SPD online filter manages a high volume of data, organized into files and logical datasets to support its multi-step, high-throughput processing. The data management system provides control over the lifecycle of all data in the online filter.

Based on this, the DMS should provide the following functionalities:

— separation of metadata from the physical storage of files (i.e., cataloging), which implies the use of a database (DB). The database should contain information about files, datasets, and available storages;

— tracking the data lifecycle, which requires storing the status model of files and datasets;

— registration of files within the storage and datasets through set of APIs (application programming interface);

— ensuring consistency between the storage and the DB via background services for the correct functioning of the entire system.

ARCHITECTURE OF THE DATA MANAGEMENT SYSTEM

Based on the functionality of the data management system described above, it was split into three independent microservices (Fig. 1):

— **dsm-register**. This service processes requests for adding, removing, and uploading data asynchronously through a message queue. Upon handling a request, the dsm-register makes changes to the database via the API provided by the dsm-manager service;

— **dsm-manager**. This service provides a REST (Representational State Transfer) API to the database, serving both internal services (dsm-register and dsm-inspector) and external systems (workflow and workload management systems);

— **dsm-inspector**. A set of automated background services for ongoing data state monitoring. They perform critical tasks such as consistency checks, storage usage monitoring, deleting files, and transferring files to external storage.

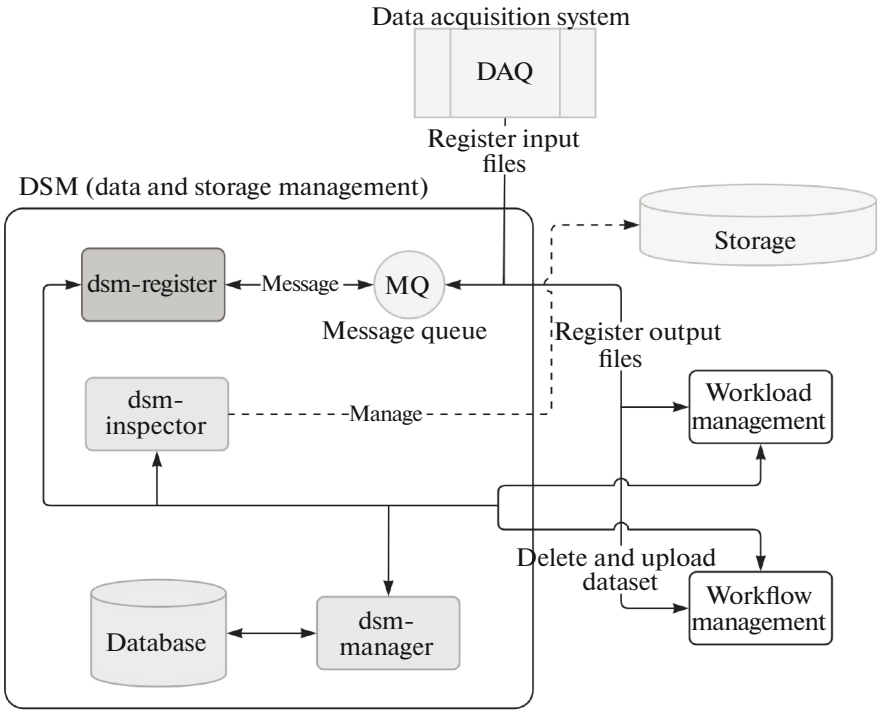


Fig. 1. Architecture of the Data Management System.

DATA MANAGEMENT SYSTEM PROTOTYPE

All services within the data management system are developed using Python 3.11. For deployment, the projects are hosted on a virtual machine and orchestrated using Docker Compose, with each application running in an isolated docker container.

Dsm-manager makes changes to the database and provides a REST API to this DB for all other services. PostgreSQL is used as the database, deployed on a separate virtual machine.

Table schemas are defined in the code using ORM (object-relational mapping) models from SQLAlchemy. The tables are created using Alembic migrations. The FastAPI framework is used to create the API.

The service provides the ability to get, change and delete information about files, datasets, storages, and "dark" files (files that exist in the storage, but information about them is missing from the system). In addition to standard operations, dsm-manager also allows you to obtain a list of files in a dataset, a list of datasets to which a certain file belongs, a list of files and datasets in a certain status, search for storage by its type, and clone the dataset.

Dsm-register accepts requests for adding and deleting data through the RabbitMQ message broker. It implements the configuration of message queues and their handlers. The following scenarios are implemented here:

(1) DAQ sends information about the input files to the queue. The dsm-register then consumes this information, creates a new dataset, registers the files within it, and finally notifies the workflow management system (WFMS) that a new dataset is available;

(2) Workload management system (WMS) sends information about the processed files, dsm-register registers them and informs the WMS about the registration status of each file;

(3) WFMS sends a request to delete datasets, and the dsm-register marks them for deletion;

(4) WFMS sends a request to upload output datasets to external storage, and the dsm-register marks them for upload.

Dsm-inspector works directly with files on storage, not just with records in the database.

The service consists of a set of background tasks: file consistency checking, deletion of datasets and files (executing dataset deletion requests and deleting "dark" files), storage control (monitoring storage for "dark" files and monitoring storage occupancy), and data upload control. Each task is an independent application running in a separate docker container.

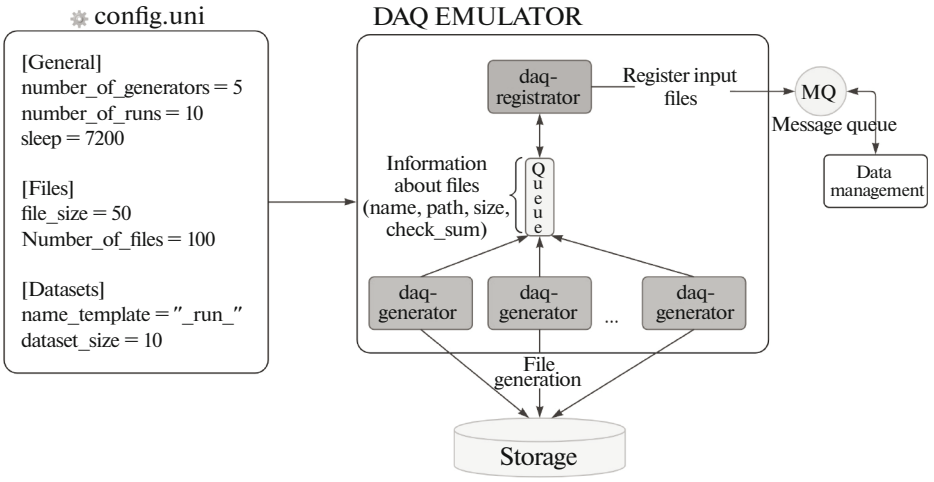


Fig. 2. Architecture of the SPD DAQ emulator.

ARCHITECTURE OF THE SPD DAQ EMULATOR

To fully test not only the data management system, but also the entire online filter, we need a continuous stream of data. Since the data from the detector will not be available for a while, it was decided to create SPD DAQ emulator. This emulator comprises two main components: the daq-generator and the daq-registrator (Fig. 2).

— **daq-generator** creates files on the storage system and places the corresponding information into a message queue. There can be several such generators at the same time;

— **daq-registrator** monitors the queue. Once the number of accumulated files meets the threshold required to form a dataset, it generates and routes a corresponding message to the DMS via a message broker.

All the necessary parameters are specified in the configuration file: the number of generators running in parallel, the number of runs, how many files we generate per run and their size, how many files should be in the dataset and the template we use to generate the dataset name.

CONCLUSIONS

This paper has presented the data management system, a component responsible for controlling the data lifecycle. The DMS is comprised of three microservices: dsm-register, dsm-manager, and dsm-inspector. During the development, the core functionality for both dsm-register and dsm-manager was successfully implemented, alongside a substantial portion of the dsm-inspector service.

To ensure a continuous data flow, the development of the SPD DAQ emulator was initiated. The initial phase involved formulating its requirements and creating a basic prototype.

FUNDING

This work was supported by ongoing institutional funding. No additional grants to carry out or direct this particular research were obtained.

CONFLICT OF INTEREST

The authors of this work declare that they have no conflicts of interest.

REFERENCES

1. V. M. Abazov et al. (SPD Collab.), "Conceptual design of the spin physics detector," arXiv Preprint (2021). arXiv:2102.00442.
2. V. Andreev, A. Belova, A. Galoyan, S. Gerassimov, G. Golovanov, P. Goncharov, A. Gribowsky, D. Maletic, A. Maltsev, A. Nikolskaya, D. Oleynik, G. Ososkov, A. Petrosyan, E. Rezvaya, E. Shchavelev, A. Tkachenko, V. Uzhinsky, A. Verkheev, and A. Zhemchugov, "Offline software and computing for the SPD experiment," in *Proceedings of the 9th International Conference "Distributed Computing and Grid Technologies in Science and Education"* (Crossref, 2021), pp. 26–31. https://doi.org/10.54546/mlit.2021.80.69.001
3. V. M. Abazov et al. (SPD Collab.), "Technical design report of the spin physics detector at NICA," Nat. Sci. Rev. **1**, 1 (2024). https://doi.org/10.54546/naturalscirev.100101

Publisher's Note. Pleiades Publishing remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.
AI tools may have been used in the translation or editing of this article.