
COMPUTER TECHNOLOGIES IN PHYSICS

Monitoring of the SPD Data Management System

A. S. Konak^{a,*} and A. Sh. Petrosyan^a

^a Joint Institute for Nuclear Research, Dubna, 141980 Russia

*e-mail: konak@jinr.ru

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

Abstract—Active work continues on the creation of the SPD (Spin Physics Detector) facility at the NICA accelerator complex, which is located at the Joint Institute for Nuclear Research. Since the detector or facility will collect a large amount of data, data processing and storage will be carried out in a distributed computing environment. Despite the fact that the detector will not be launched soon, active work is already underway on the infrastructure in which the accumulated data will be stored and processed. The data management system Rucio, a standard solution for data management in the field of high energy physics, is used as a management tool. The system has already been launched, integrated into an offline data processing system, and manages terabytes of data. There are also already active users. And there is a need to monitor the state of the system, its operability, as well as user activity. The report presents the experience of designing a monitoring system for the Rucio data management system.

DOI: 10.1134/S1547477126700755

INTRODUCTION

The SPD (Spin Physics Detector) experiment is one of the key projects of the NICA collider aimed at studying the spin structure of nucleons and other fundamental properties of matter [1]. The successful implementation of the scientific experimental program will generate a large amount of data that requires reliable, scalable, and efficient infrastructure for storage, processing and management.

Rucio [2] has been chosen to manage data in a distributed storage environment. The deployed data management system demonstrates stable and efficient operation. The critical interactions between the workload management system PanDA [3], the data management system Rucio, and the data transfer service FTS3 (File Transfer Service 3) are configured and functioning perfectly. This level of maturity and integration leads us to the next challenge: the need for a comprehensive monitoring system for the Rucio itself.

This article describes the design, implementation, and features of a new monitoring system developed to provide full visibility and control over all aspects of the distributed data system in the SPD experiment.

METRIC TYPES

Rucio collects a bunch of metrics that can be used to monitor different activities, but does not provide a ready-made monitoring solution. It was necessary to: design a monitoring system, deploy all the necessary infrastructure, configure each component, study

information provided by Rucio and finally build visualization panels.

Rucio provides different types of information for monitoring:

- **Metrics** provided by Rucio server and Rucio-daemons. Metric is a measurable numerical characteristic that shows how the system works.
- **Messages** provided by Rucio server. Generated during replication, deletion, transfer submission processes and stored in the Rucio DB (Rucio database). Messages consist of information about replication, deletion or transfer submission operations, e.g. scope, name, size, status (successful/failure), duration, etc in JSON (Java script object notation) format.
- **Traces** provided by Rucio-server. Trace is information about the execution of a single transaction in JSON format. Rucio gets GET, PUT traces from workload management system—PanDA, and UPLOAD, DOWNLOAD, TOUCH traces from Rucio clients.

COMPONENTS OF THE MONITORING SYSTEM

In order to organize monitoring for the Rucio data management system, it is necessary to deliver all types of information for monitoring to the storage systems of this type of information. Then build graphs by taking information from these systems.

Drawing upon Rucio documentation and the experience of colleagues, an integrated monitoring architecture was designed, leveraging Graphite, Open-

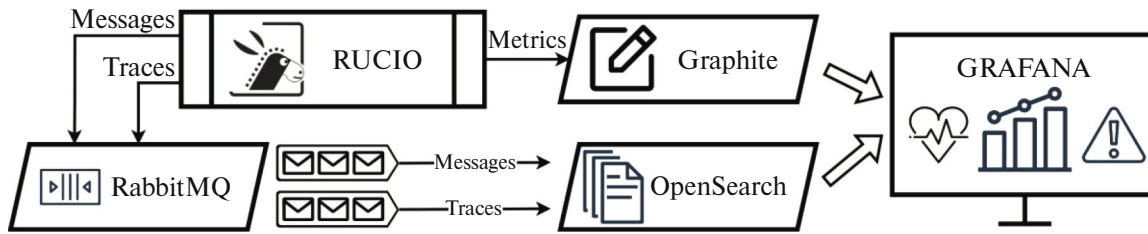


Fig. 1. Components of monitoring system.

Search, RabbitMQ, and Grafana (Fig. 1). Each component was deployed in a Docker container based on JINR cloud computing infrastructure [4], configured and tested.

The Rucio developers recommend using Graphite to store metrics and this particular solution was chosen. During the work with Graphite, no problems were observed and it was simply integrated with Rucio to store metrics.

OpenSearch was chosen to store messages and traces. It is an open source document-based search and analysis platform based on Elasticsearch. It provides high performance for full-text search for our monitoring system without licensing restrictions, which makes it an ideal solution for scientific projects requiring complete openness.

Queues in RabbitMQ are used for temporary storage of messages and traces before delivery to long-term storage in OpenSearch. RabbitMQ is a high-performance open source message broker. The Rucio developers recommend using ActiveMQ, but RabbitMQ offers better performance and stability. It is optimized to handle large streams of small messages.

Grafana is used to build visualisation panels. It is an open source platform for visualization and analysis of metrics, focused on working with time series. It supports multiple backends simultaneously (Graphite, OpenSearch, databases) and rich possibilities for creating custom dashboards. For our monitoring system, Grafana provides more specialized and flexible tools for analyzing time series and creating interactive dashboards.

COLLECTING OF METRICS FOR MONITORING

Metrics are being sent to Graphite automatically after appearing. The lifetime of metrics can be presented by three steps.

(1) Metrics generation. The Rucio code is instrumented using a StatsD library for collecting metrics. The metrics are collected in a memory buffer and sent in batches to the StatsD daemon on Rucio-server side.

(2) Aggregation and forwarding. The StatsD daemon pre-aggregates the received metrics and converts

them for transmission to Graphite. Finally daemon sends a ready-to-send metric to Graphite.

(3) Receiving and storage. On the Graphite side Carbon daemon receives metrics and writes them to disk in an efficient format in Whisper DB. Whisper is a fixed-size, time-series database format for storing numerical data over time.

Messages and traces temporarily stored in RabbitMQ. A Hermes (rucio-daemon) sends messages to RabbitMQ. Messages are taken from the Rucio DB and sent in batches of ten thousand. Traces are automatically sent to RabbitMQ after the Rucio-server receives them.

One unit of information in OpenSearch is a document. Documents are stored in Indices. Index is a logical space for similar documents in OpenSearch. There are different indices in OpenSearch in our setup for different event_type:

- For messages there are indices for transfer submission process (rucio_transfer_queued, rucio_transfer_submitted, rucio_transfer_submission_failed), transfer process (rucio_transfer_done, rucio_transfer_failed), and for deletion process (rucio_deletion_failed, rucio_deletion_done, rucio_deletion_not_found).
- For traces there are different indices for information provided by PanDA (rucio_trace_put, rucio_trace_get), and for information provided by rucio-clients rucio_trace_upload, rucio_trace_download, rucio_trace_touch).

Messages and traces were being delivered to OpenSearch for long term storage with special consumers. Consumer takes message or trace from the queue, pre-processes document and distributes it to the target index in OpenSearch by event_type of message or trace.

Load test of message consumer, when the Rucio daemon sends messages to the queue and the message consumer acknowledges messages at the same time, showed that the consumer can handle such a load. Incoming messages have time to be read out and do not accumulate in the queue. For the load trace of trace consumer, 550 thousand traces were queued, and after that the consumer was started. The consumer is able to cope with half a million traces in two minutes, which is quite enough to ensure fast delivery of

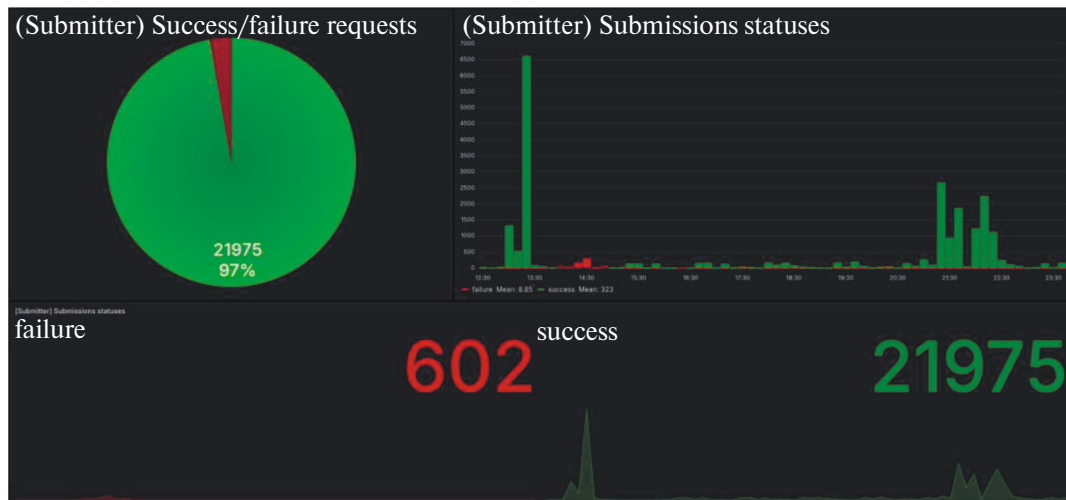


Fig. 2. Transfer submission dashboard.

information from Rucio to the monitoring system during production.

VISUALISATION PANELS

The collected metrics, messages and traces are aggregated into a series of specialized visualization panels in Grafana, providing a comprehensive operational overview of the Rucio data management system. These dashboards are categorized by their functional purpose as follows:

Core system and daemons performance. This category focuses on the health and efficiency of the Rucio infrastructure itself. Metrics obtained from the Rucio server and daemons are used as source of information for this category.

- **Transfer submission monitoring:** Tracks the volume and success rate of transfer submissions, including the number of successful/failed requests (Fig. 2) and the time spent per submission.
- **Deletion monitoring:** Provides insight into the data deletion process, monitoring the number of deletion requests, unlocked replicas, the protocols used, and the average duration from daemon request to storage system response.
- **Daemons activity dashboard:** Offers a unified view of all running Rucio daemons, confirming their continuous operation, activity levels, and absence of critical errors.

User and production activity. These panels offer visibility into how the system is utilized by users and production workflows.

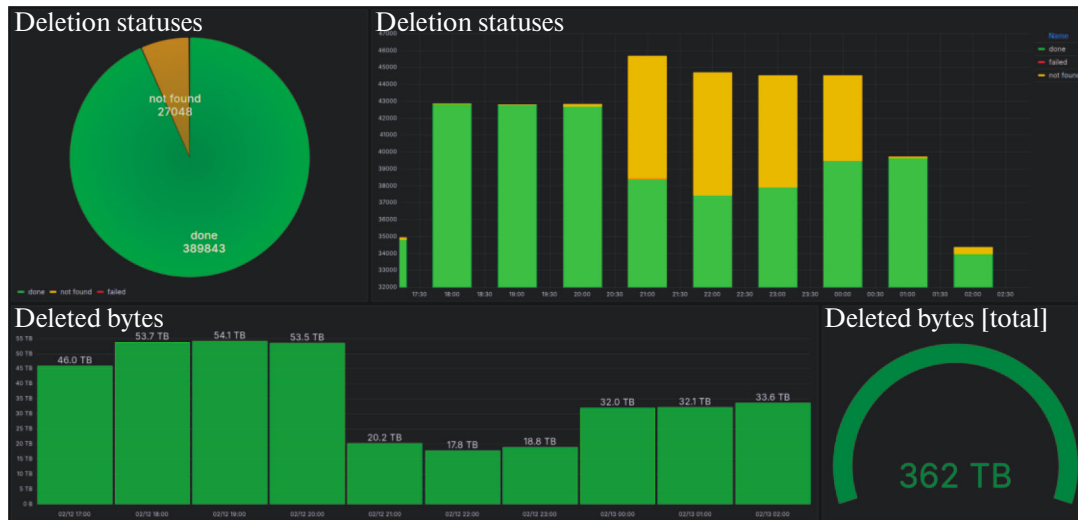
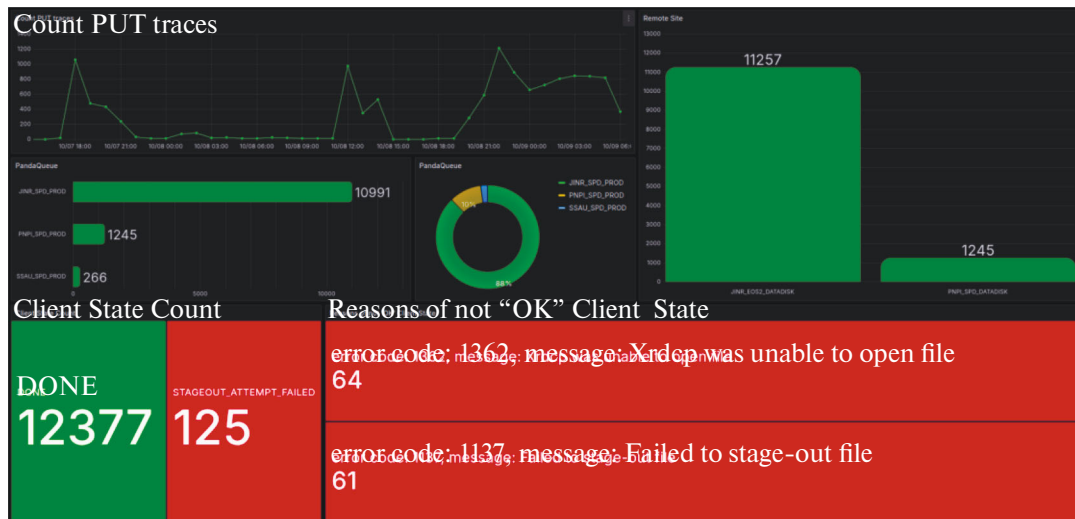
- **User activity monitoring:** Tracks authentication and authorization events (with metrics) in a depersonalized manner, ensuring no information security issues while providing data on system access patterns. With trace data (e.g., upload, download) these dashboards

monitor users data flow. They display number of traces, client states (OK/error), error types and counts, activity per scope/dataset/local site, file sizes uploaded/downloaded with rucio-client, and statistics about using protocols, such as number of activities and average duration of transaction.

- **Production workflow traces:** Based on trace data (e.g., get, put operations), these dashboards monitor the data flow from production jobs. They display the number of traces, jobs per Panda queue/remote site, task ID distribution, client states (OK/error), error types and counts, activity per scope/dataset/compute node, and average file sizes used in tasks. Figure 3 shows an example of this dashboard.

Data lifecycle operations. This set of panels delivers detailed analytics on the core data management operations: transfer and deletion. Messages are used for this category.

- **Transfer submission analysis:** Focuses on the submission queue, showing counts of successful and failed submissions, broken down by scope and dataset.
- **Transfer execution analysis:** Monitors the actual file transfers, reporting the count of success/failure, the total volume of bytes transferred (overall and per scope/dataset), average transfer duration, average file size, and a matrix of successful transfers and data volume between RSEs.
- **Deletion execution analysis:** Tracks the outcome of deletion requests, including counts of success, not_found, and failure outcomes, the total volume of data deleted, average deletion duration, average deleted file size, and deletion statistics per RSE. Figure 4 shows an example of the deletion process dashboard.



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.

OPEN ACCESS

This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>

REFERENCES

1. 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>
2. M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, et al., “Rucio: Scientific data management,” *Comput. Software Big Sci.* **3**, 11 (2019).
<https://doi.org/10.1007/s41781-019-0026-3>
3. T. Maeno, A. Alekseev, F. H. Barreiro Megino, K. De, W. Guan, E. Karavakis, A. Klimentov, T. Korchuganova, F. Lin, P. Nilsson, et al., “PanDA: Production and distributed analysis system,” *Comput. Software Big Sci.* **8**, 4 (2024).
<https://doi.org/10.1007/s41781-024-00114-3>
4. N. Balashov, I. Kuprikov, N. Kutovskiy, A. Makhalkin, Ye. Mazhitova, I. Pelevanyuk, R. Semenov, and D. Shpotya, “Changes and challenges at the JINR and its member states cloud infrastructures,” *Phys. Part. Nucl.* **55**, 366–370 (2024).
<https://doi.org/10.1134/S1063779624030092>

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.