ISSN 1063-7796, Physics of Particles and Nuclei, 2024, Vol. 55, No. 3, pp. 617–620. © Pleiades Publishing, Ltd., 2024.

SPD Event Index

F. Prokoshin[a, *], **I. Tvauri**[b, **], **Z. Budtueva**[b, ***], **R. Gurtsiev**[b, ****], **and A. B. Gazzaev**[b, *****]

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

[b] *North Ossetian State University, Vladikavkaz, Russia*

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

***e-mail: ateia@mail.ru*

****e-mail: zbudtueva@list.ru*

*****e-mail: vilovnok@gmail.com*

******e-mail: gazzaev1999@yandex.ru*

Received September 18, 2023; revised November 6, 2023; accepted December 1, 2023

Abstract—The SPD experiment has to collect large amount of data: up to trillion events (records of a collision results) have to be stored and analyzed, producing tens of petabytes of data. This information will be distributed between a number of computing sites on a various storage locations, with duplication to avoid data loss and improve performance. An information system is necessary to efficiently access all instances of the events, and the SPD Event Index is being developed for this purpose. It is a catalog of all events obtained from the detector or simulated, in all permanent instances of different formats and versions.

DOI: 10.1134/S1063779624030699

INTRODUCTION

The SPD experiment [1] is a research project aimed at studying the physics of elementary particles and searching for new fundamental knowledge about the interactions and structure of particles. The main goal of this project is to analyze the data obtained from the SPD detector installed on the NICA particle accelerator, created at the Joint Institute for Nuclear Research, located in the city of Dubna, Moscow region.

To store and process a large amount of data obtained during the SPD experiment, the creation of complex information systems will be required. One of such systems should be an Event Index—a catalog of physical events received from the detector or modeled for data analysis and processing.

SPD INFORMATION ECOSYSTEM

At this stage of the creation of the SPD experiment, information and computing systems are in the initial stage of development. The creation of an Event Index is planned to start with parts that do not depend on other components.

SPD Event Index is being developed as a comprehensive information system that should provide:

• obtaining information about experimental events and simulated data by indexing data files containing information about these events;

• transfer of this information and write to databases;

• access to information for data processing and analysis programs via API and applications;

• access to information to users through interactive and asynchronous interfaces.

The choice of a platform for data storage and management was made based on the expected flows and volumes of information, the content of the record and the expected ways of using Event Index.

The estimated data flow at the output of the online filter will be from tens thousand events per second in the early stage up to 150 thousand at maximum machine performance, giving us from hundreds of billion (10^{11}) to few trillion (10^{12}) events per year.

An entry for an event in Event Index must contain the following fields:

• event IDs: Run number (run_number) and event number in Run (event_number);

• information about online filter solutions, in the form of a bit mask (olf_result);

• unique identifier of the RAW data file containing this event (fuid_raw). Using the UUID of a file, you can access it through a distributed storage system;

• ID of the dataset this file is included in (dsid_raw).

As the data is processed, new instances of the recovered events will be created in a format optimized for physical analysis (AOD). Pointers to different ver-

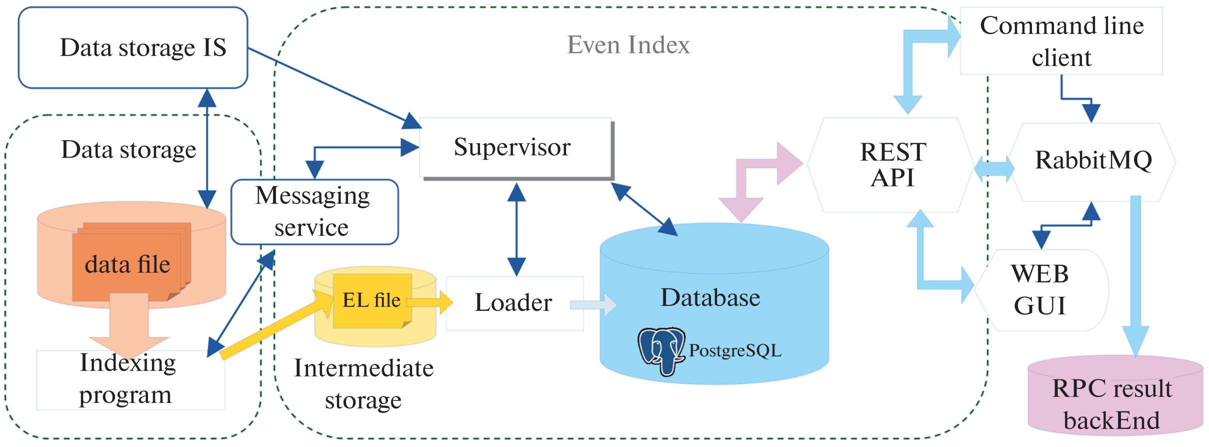
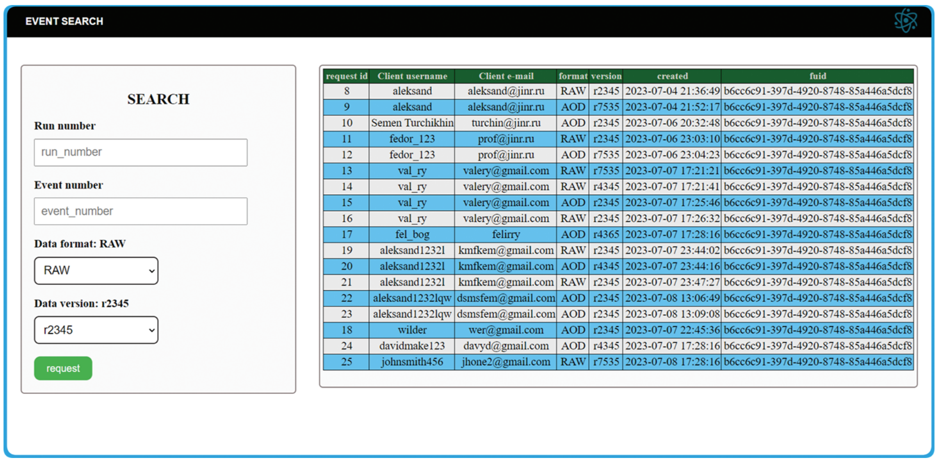


Fig. 1. SPD Event Index data flow architecture and schema.



request id	Client username	Client e-mail	format	version	created	fuid
8	aleksand	aleksand@jinr.ru	RAW	r2345	2023-07-04 21:36:49	b6cc6c91-397d-4920-8748-85a446a5dcf8
9	aleksand	aleksand@jinr.ru	AOD	r7535	2023-07-04 21:52:17	b6cc6c91-397d-4920-8748-85a446a5dcf8
10	Semen Turchikhin	turchin@jinr.ru	AOD	r2345	2023-07-06 20:32:48	b6cc6c91-397d-4920-8748-85a446a5dcf8
11	fedor_123	prof@jinr.ru	RAW	r2345	2023-07-06 23:03:10	b6cc6c91-397d-4920-8748-85a446a5dcf8
12	fedor_123	prof@jinr.ru	AOD	r7535	2023-07-06 23:04:23	b6cc6c91-397d-4920-8748-85a446a5dcf8
13	val_ry	valery@gmail.com	RAW	r7535	2023-07-07 17:21:21	b6cc6c91-397d-4920-8748-85a446a5dcf8
14	val_ry	valery@gmail.com	RAW	r4345	2023-07-07 17:21:41	b6cc6c91-397d-4920-8748-85a446a5dcf8
15	val_ry	valery@gmail.com	AOD	r2345	2023-07-07 17:25:46	b6cc6c91-397d-4920-8748-85a446a5dcf8
16	val_ry	valery@gmail.com	RAW	r2345	2023-07-07 17:26:32	b6cc6c91-397d-4920-8748-85a446a5dcf8
17	fel_bog	felirry	AOD	r4365	2023-07-07 17:28:16	b6cc6c91-397d-4920-8748-85a446a5dcf8
19	aleksand12321	kmfkem@gmail.com	RAW	r2345	2023-07-07 23:44:02	b6cc6c91-397d-4920-8748-85a446a5dcf8
20	aleksand12321	kmfkem@gmail.com	AOD	r4345	2023-07-07 23:44:16	b6cc6c91-397d-4920-8748-85a446a5dcf8
21	aleksand12321	kmfkem@gmail.com	RAW	r2345	2023-07-07 23:47:27	b6cc6c91-397d-4920-8748-85a446a5dcf8
22	aleksand1232lqw	dsmsfem@gmail.com	AOD	r2345	2023-07-08 13:06:49	b6cc6c91-397d-4920-8748-85a446a5dcf8
23	aleksand1232lqw	dsmsfem@gmail.com	AOD	r2345	2023-07-08 13:09:08	b6cc6c91-397d-4920-8748-85a446a5dcf8
18	wilder	wer@gmail.com	AOD	r2345	2023-07-07 22:45:36	b6cc6c91-397d-4920-8748-85a446a5dcf8
24	davidmake123	davyd@gmail.com	AOD	r4345	2023-07-07 17:28:16	b6cc6c91-397d-4920-8748-85a446a5dcf8
25	johnsmith456	jhone2@gmail.com	RAW	r7535	2023-07-08 17:28:16	b6cc6c91-397d-4920-8748-85a446a5dcf8

Fig. 2. Web interface.

sions of such files will be added to the event record. Also, important event parameters can be added to the record, which can be used for classification and selection.

For reliable storage and processing of structured data, a PostgreSQL [2] database was chosen, the features of which are multithreading and the ability to process large amounts of data.

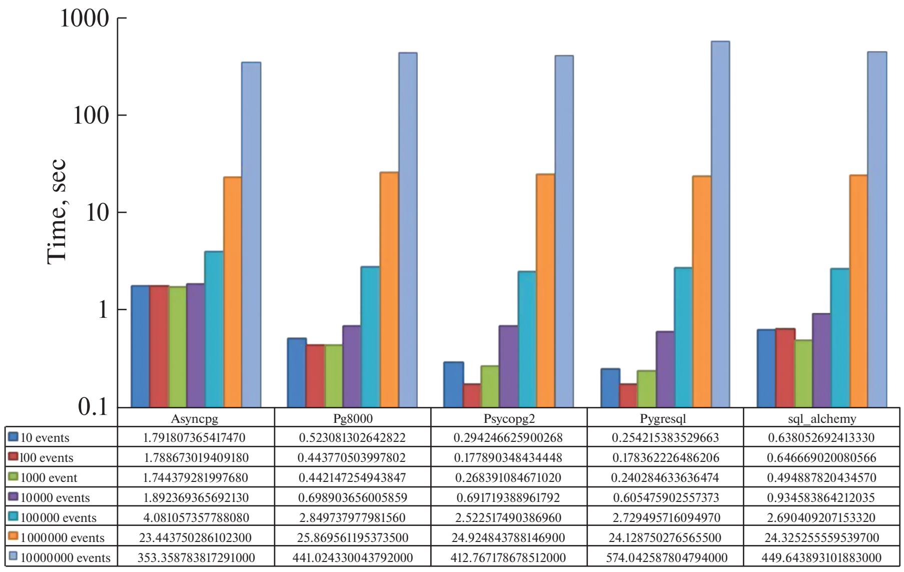
Within the framework of this study, a convenient and efficient program interface was developed and implemented that performs data exchange using the RESTful API [3]. To ensure the flexibility and reliability of the server side, Flask [4] was chosen—a microframework for the Python language. The frontend part of the client interface was developed using the Angular [5] framework, which provides tools for creating modern dynamic user interfaces, as well as providing effective interaction with the server and data manipulation.

RabbitMQ [6] is a message broker that allows you to send, receive and route messages between application components asynchronously. Celery [7] is a system that allows you to perform operations in the background. The joint use of RabbitMQ and Celery when creating a system for asynchronous task processing helps to improve its performance. Flask can focus on HTTP request processing while task processing is passed to RabbitMQ and Celery, providing a modular and flexible application architecture.

The integration of all these technologies made it possible to create a powerful and flexible interface for data exchange, contributing to more efficient processing of information in the EventIndex system.

DATA GENERATION

To test the prototype of the system, sets of generated intermediate Event Index data are created. The format of these sets is JSON, and it does not depend



	Asyncpg	Pg8000	Psycopg2	Pygresql	sql_alchemy
🟦 10 events	1.791807365417470	0.523081302642822	0.294246625900268	0.254215383529663	0.638052692413330
🟥 100 events	1.788673019409180	0.443770503997802	0.177890348434448	0.178362226486206	0.646669020080566
🟩 1000 event	1.744379281997680	0.442147254943847	0.268391084671020	0.240284633636474	0.494887820434570
🟪 10000 events	1.892369365692130	0.698903656005859	0.691719388961792	0.605475902557373	0.934583864212035
🟦 100000 events	4.081057357788080	2.849737977981560	2.522517490386960	2.729495716094970	2.690409207153320
🟧 1000000 events	23.443750286102300	25.869561195373500	24.924843788146900	24.128750276565500	24.325255559539700
🟦 10000000 events	353.358783817291000	441.024330043792000	412.767178678512000	574.042587804794000	449.643893101883000

Fig. 3. Recording time of data sets using various modules.

on the format in which the data from the detector will be stored. For each event, identifiers are generated (run_number and event_number), a random of_result, as well as fuji_raw and psd_raw. In the future, pseudo-data for AOD files is created based on these sets for the same events.

This data is then written to tables in the database: event records are stored in the "events" table, and information about datasets is stored in the "datasets" table. The dataset ID serves as a foreign key for the event table.

OPTIMIZATION OF DATA RECORDING SPEED

The procedure used in the first version of the interface for recording each event with a separate PUT instruction showed insufficient speed when testing even on relatively small data arrays. For example, 1000 events, on average, are recorded for 1 min 55 s. A system capable of coping with a flow of tens of thousands of events per second is needed. To solve this problem, various optimization methods were investigated to speed up the process of writing data to tables.

Along with using the common psycopg2 driver for interacting with PostgreSQL, the following modules were tested: asyncpg, pg8000, PyGreSQL, SQLAlchemy.

To optimize the data write speed, a COPY query was used instead of an INSERT query. The test results are shown in Fig. 3.

Using COPY accelerated the writing of data to the table by about 2 times. It follows from the graph that when working with a small number of events (10–100000), using the PyGreSQL module is effective, but for a large number of events (from 1000000), the asynchronous asyncpg module should be used.

At the moment, work is underway to optimize the data write speed by changing PostgreSQL parameters, the effect of block size on download speed is being investigated. The system tries to load all the data at once, but at high load it adapts using a cluster approach for efficient data processing. Additionally, the possibility of parallel data loading is being considered to improve performance.

CONCLUSIONS

In the course of further development of the Event Index project, the following tasks are expected to be performed:

(1) Development of user authorization and authentication mechanisms using single sign-on technology and group access policies;

(2) API development (REST, Python, C++, ...);

(3) Optimization of user request processing, with synchronous or asynchronous output of results, depending on the volume of requested data;

(4) Development of mechanisms for transmitting "Event Index" data obtained by indexing files located on remote nodes of a distributed computing network;

(5) Development a supervisor—software for managing, collecting and importing data into "Event Index";

(6) Development of an Event Index component monitoring system, with graphical representation of data based on popular platforms (Grafana, etc.).

The implementation of these tasks will be carried out in parallel with the development of others FROM the installation.

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. Abazov et al. (SPD Collab.), "Conceptual design of the Spin Physics Detector," arXiv:2102.00442 [hep-ex].
2. Official PostgreSQL Website. https://www.postgresql.org/. Accessed September 13, 2023.
3. L. Richardson and M. Amundsen, *RESTful Web APIs* (O'Reilly Media, 2013).
4. Flask's documentation. https://flask.palletsprojects.com/. Accessed September 13, 2023.
5. Introduction to Angular concepts. https://angular.io/guide/architecture. Accessed September 13, 2023.
6. RabbitMQ. https://www.rabbitmq.com/. Accessed September 13, 2023.
7. Celery—Distributed Task Queue. https://docs.celeryq.dev/. Accessed September 13, 2023.

SPELL: 1. ok