ISSN 1547-4771, Physics of Particles and Nuclei Letters, 2026, Vol. 23, No. 5, pp. 682–685. © Pleiades Publishing, Ltd., 2026.

COMPUTER TECHNOLOGIES IN PHYSICS

The Design and Implementation of the Pilot Application for the SPD Online Filter

L. R. Romanychev[a, b, *], D. A. Oleynik[a, **], and N. V. Greben[a, *]**

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

[b] *St. Petersburg State University, St. Petersburg, 198504 Russia*

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

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

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

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

Abstract—This paper presents the design and implementation of the Pilot application for the SPD online filter. We describe its architecture, components, and operational workflow, which leverages a late-binding mechanism for efficient resource utilization. A key feature is an adaptive workload management strategy that maximizes resource utilization by enabling GPU-node pilots to dynamically execute CPU-bound jobs. The results of testing using a DAQ emulator are presented.

DOI: 10.1134/S1547477126700731

1. INTRODUCTION

The Spin Physics Detector (SPD) at the NICA collider facility [1] will produce a vast amount of data, requiring an advanced high-throughput computing (HTC) infrastructure for its processing. The SPD Online Filter is being developed to handle this challenge through multi-step data processing workflows. This paper describes the design and implementation of a key component of this system: the Pilot agent application.

Pilot-based systems are a proven solution for HTC infrastructure, providing a flexible layer for executing jobs on heterogeneous resources [2–4]. In this paradigm, a lightweight "pilot" acts as an intermediary: once active on a compute node, it contacts a workload management system (WMS) to fetch a payload. This approach is widely used in major experiments, but many projects develop their own systems, leading to a variety of implementations. Our work builds upon this established concept by introducing an adaptive layer for resource management tailored to the needs of the SPD experiment [5].

2. ARCHITECTURE OF THE SPD PILOT SOFTWARE

The overall architecture of the Pilot and its integration with the workload management system is described in detail in [6]. On each compute node, a persistent background daemon starts and monitors pilot agents.

A key concept is *Late Binding*, which assigns jobs to pilots only when they are active and ready, reducing queue times. The WMS manages distinct queues for different job types. GPU pilots can fetch jobs from CPU queues when their primary queue is empty, maximizing resource utilization (Fig. 1). This adaptive behavior ensures efficient use of computational resources while maintaining system responsiveness and fault tolerance.

3. IMPLEMENTATION AND WORKFLOW

The Pilot implementation uses a multi-level process and thread architecture for reliable job execution. The execution flow is illustrated in Fig. 2.

3.1. Process and Thread Architecture

The pilot employs a three-level hierarchy: the main process handles communication with the WMS, a payload process executes the job in isolation, and a job thread within the payload process manages the execution stages. This separation ensures that failures in job execution do not affect the main communication loop.

The main process continuously sends heartbeats to the WMS and receives commands. When a job is obtained from RabbitMQ, the main process creates a payload process using Python's multiprocessing module. The payload process spawns a job thread that executes the job stages sequentially: pre-processing,

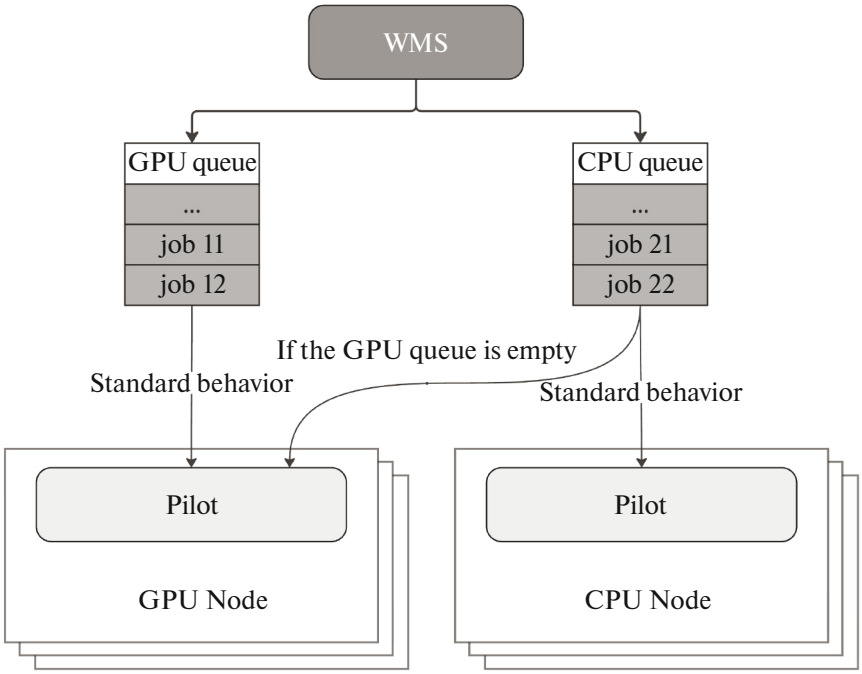


Fig. 1. Adaptive queue management: GPU pilot fetching from CPU queue.

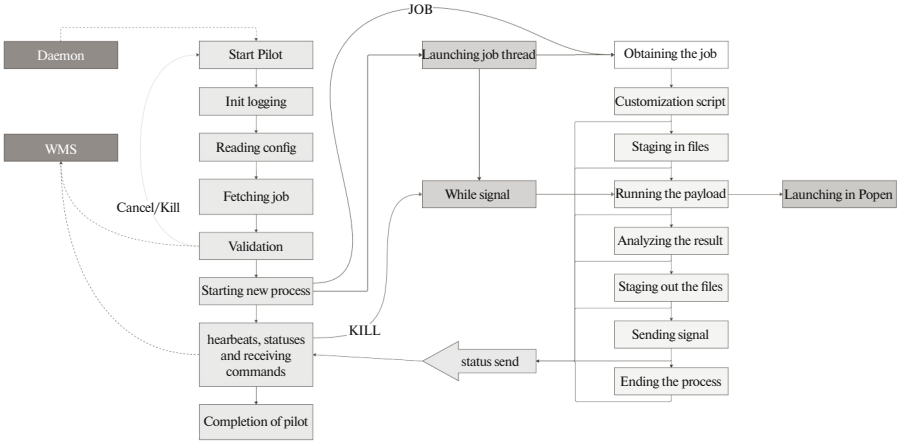


Fig. 2. Pilot execution pipeline and inter-thread communication.

stage-in, payload execution, post-processing, and stage-out.

3.2. Thread Synchronization and Cancellation

Inter-process and inter-thread communication uses queues and events. The main process and payload process share two queues: one for passing job data and another for status updates. A multiprocessing. Event object provides a synchronization mechanism for cancellation.

The event-based cancellation works as follows: when the WMS sends a “stop” command, the main process sets the event by calling event.set(). The payload process monitors this event in two places: (1) in the monitoring loop that checks log file size, and (2) within the payload execution method, where it periodically checks event.is_set() while the subprocess is running.

When event.is_set() returns True, the payload process terminates the job subprocess gracefully and sets the job status to Failed. Similarly, the job thread

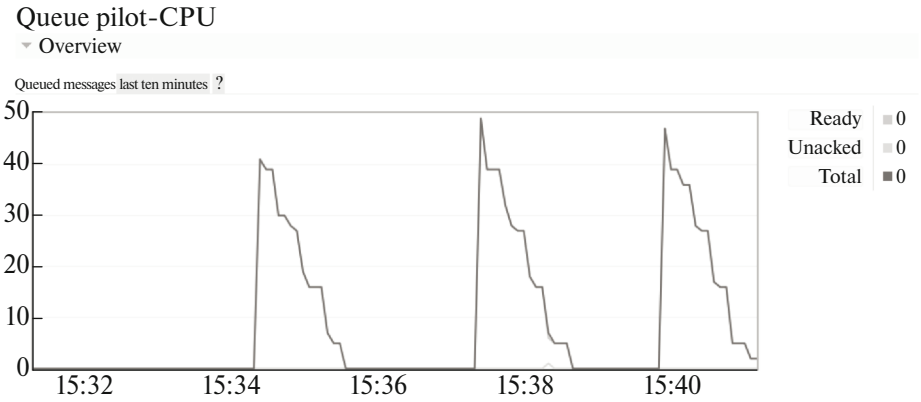


Fig. 3. Queue monitoring: three successful processing cycles.

checks the event after each stage completion. If the event is set, the thread stops processing remaining stages and exits. This ensures that all child threads and processes terminate promptly when cancellation is requested, preventing resource leaks and zombie processes.

3.3. Job State Model

The job lifecycle is tracked through states: *Registered*, *Extracted* (managed by WMS), *Obtained*, *Preprocessing*, *Stage-in*, *Running*, *Post-processing*, *Stage-out*, *Completed* (managed by Pilot). Any error moves the job to *Failed* state. Each stage method is wrapped with error handling that automatically uploads logs on failure.

3.4. Testing and Validation

Testing was conducted on a JINR Cloud virtual machine [7]. Due to limited hardware, 10 daemons were launched on a single node, each spawning pilots. At the current stage, the processor type is configured through a configuration file. A RAND processor type option was added for flexible testing without dedicated GPU hardware, allowing the pilot to randomly assign itself either CPU or GPU type. Fig. 3 shows the queue monitoring dashboard demonstrating three successful processing cycles, where messages are queued, processed, and cleared.

4. CONCLUSIONS

The Pilot application for the SPD online filter has been designed, implemented, and successfully tested. The proposed approach provides a unified, adaptable, and low-overhead interface for job execution. The key features are:

• An adaptive workload management system that maximizes resource utilization by allowing GPU pilots to execute CPU jobs.

• Reduced overhead and high efficiency thanks to the lightweight pilot and late-binding mechanism.

• Robust inter-process and inter-thread communication with event-based cancellation mechanism ensuring proper termination of all child processes and threads when jobs are cancelled or errors occur.

Future work will focus on comprehensive joint testing of SPD online filter subsystems, where the pilot software serves as one component of the integrated system.

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.), "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. Sadiku, K. Eze, and S. Musa, "High-throughput computing," Int. J. Trend Res. Dev. **5**, 1 (2018).
3. M. Turilli, M. Santcroos, and S. Jha, "A comprehensive perspective on pilot-job systems," ACM Comput. Surv. **51**, 1–32 (2018). https://doi.org/10.1145/317785
4. A. Merzky, M. Turilli, M. Maldonado, M. Santcroos, and Sh. Jha, "Using pilot systems to execute many task workloads on supercomputers," in *Workshop on Job*

Scheduling Strategies for Parallel Processing (Springer, 2018), pp. 61–82. https://jsspp.org/papers18/JSSPP_2018_paper_11.pdf.

5. N. Greben, L. Romanychev, D. Oleynik, and A. Degtyarev, "SPD on-line filter: Workload management system and pilot agent," Phys. Part. Nucl. **55**, 612–614 (2024). https://doi.org/10.1134/S1063779624030419

6. N. Greben, D. Oleynik, and L. Romanychev, "Workload management system for SPD online filter," Phys. Part. Nucl. Lett. **22**, 1044–1047 (2025). https://doi.org/10.1134/s1547477125701067

7. 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.