Workload management system for SPD Online Filter

N. Greben[a, 1], *D. Oleynik*[a, 2], *L. Romanychev*[a, b]

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

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

The paper provides details and an update on the Workload Management System Middleware alongside Pilot Agent, that are part of SPD Online Filter — computing system dedicated for multi-step high-throughput processing of data from Data Acquisition System of SPD NICA detector.

INTRODUCTION

«SPD Online filter» is a computing system that performs multi-step high-throughput processing of primary data from the SPD NICA detector, that will serve as a software trigger [1]. In order to perform one step in the processing chain, tasks will be created on the specified dataset, which is defined as a logical grouping of files designed to serve as the unit of data processing. Given that each file, aggregated by the data acquisition system (DAQ), can be processed simultaneously, the high-throughput paradigm can be readily applied, with a focus on the efficient execution of a substantial number of loosely coupled jobs [2]. A task is defined as a set of input data and a handler that performs the requisite processing step and generates a dataset with the resulting output data. The objective of the entire task is to fully process the input dataset. The total output data from all jobs constitutes the output of the entire task.

The Workload Management System is currently being developed with the objective of creating an adequate number of jobs and controlling their execution on compute nodes [3]. It is expected to follow a number of requirements:

- *Task registration:* formalized task description, including job options and required metadata registration;
- *Jobs definition*: generation of required number of jobs to perform task by controlled loading of available computing resources;
- *Jobs execution management*: continuous job state monitoring by communication with Pilot, job retries in case of failures, job execution termination;
- *Consistency control*: control of the consistency of information in relation to the tasks, files and jobs;
- *Scheduling*: implementing priority-based job scheduling to ensure that high-priority jobs get access to compute resources faster.

MIDDLEWARE

SPD DAQs divides the sensor signals into time blocks and transmits the data to the input buffer of the «SPD Online filter» as files of a consistent size. Concurrently, the Data & Storage Management System (DSM) receives information about these files for the purpose of initial registration [4]. Workflow Management System (WfMS) acquires

[1] E-mail: ngreben@jinr.ru

[2] E-mail: danila@jinr.ru

the registered input datasets and initiates the processing chain based on the predefined CWL-template.[4]. This information is then supplied to the Workload Management System (WMS) as part of the task description. WMS requests information regarding the contents of the dataset and subsequently generates job instances that are dispatched to Pilots. In essence, WMS populates datasets with information about the resulting files, which serve as an input dataset for the subsequent step of the workchain. Throughout the data processing stages, Pilots engage in reading and writing files to storage, thereby creating secondary data. Based on this dataflow, the following inter-service interaction scenarios have been defined with the WfMS:

- Registration of a task for processing: WfMS passes the task description into the message broker to WMS;
- Summary of current intermediate properties of jobs/files in the system: aggregated information about the status of each jobs/files for further decision making on what to do with current task and correspondent dataset;
- Task cancellation: based on the decision made by WfMS;
- Change priority of a task: to accelerate the rate at which the corresponding dataset is being processed.

WMS generates a set of jobs by receiving the necessary file metadata from the DSM Manager (Data Catalog REST API). WMS is also responsible for creating new files in the system after a job payload has been executed by Pilot, which must be tagged in the file catalog. Another obligation is to close the dataset after the critical number of successfully processed files has been reached. The processing of requests to delete/close datasets and register new files in the system is handled by DSM-Register — a data event processor microservice that processes incoming messages from RabbitMQ [5], validates them, and calls the data catalog.

WORKLOAD MANAGEMENT SYSTEM

The system has been split into two similarly structured subdomains (see Fig. 1):

- *Task Management*: manages tasks, with Task Manager for creating and managing tasks in PostgreSQL database [6], Task Register that acts as a message gateway with RabbitMQ, and a Task Executor that generates jobs from a task-based template and interacts with a Data & Storage Management System.
- *Job Management*: similar in structure to the Task Management, but at the job level, with components such as Job Manager (handles job creation, updates, and priority), Job Register (queues and registers jobs), Job Executor (distributes jobs to pilots), and Job Watchdog (monitors jobs statuses).

Such decomposition allows for modularization, as each module can encapsulate a set of specific functionalities that are developed, maintained, and deployed separately. This is done with respect to message gateway microservices (Task/Job Register) and data validation, which are put into different Python modules.

Task/Job Manager implements a data access layer, the remaining microservices can be thought of as stateless. They only read data from a database, but do not maintain any internal session or persistent state between requests, so that improves flexibility, as the underlying database or its structure can be changed without impacting dependent services. The Job Manager (DB API microservice) can be scaled independently to handle increasing write/update loads, while other microservices focus on their specific respon-

sibilities. For example, the Job Executor retrieves chunks of jobs and dispatches them to Pilots. Once dispatched, it does not maintain in-memory information about those jobs, so it merely acts as a coordinator between the database and the pilots.

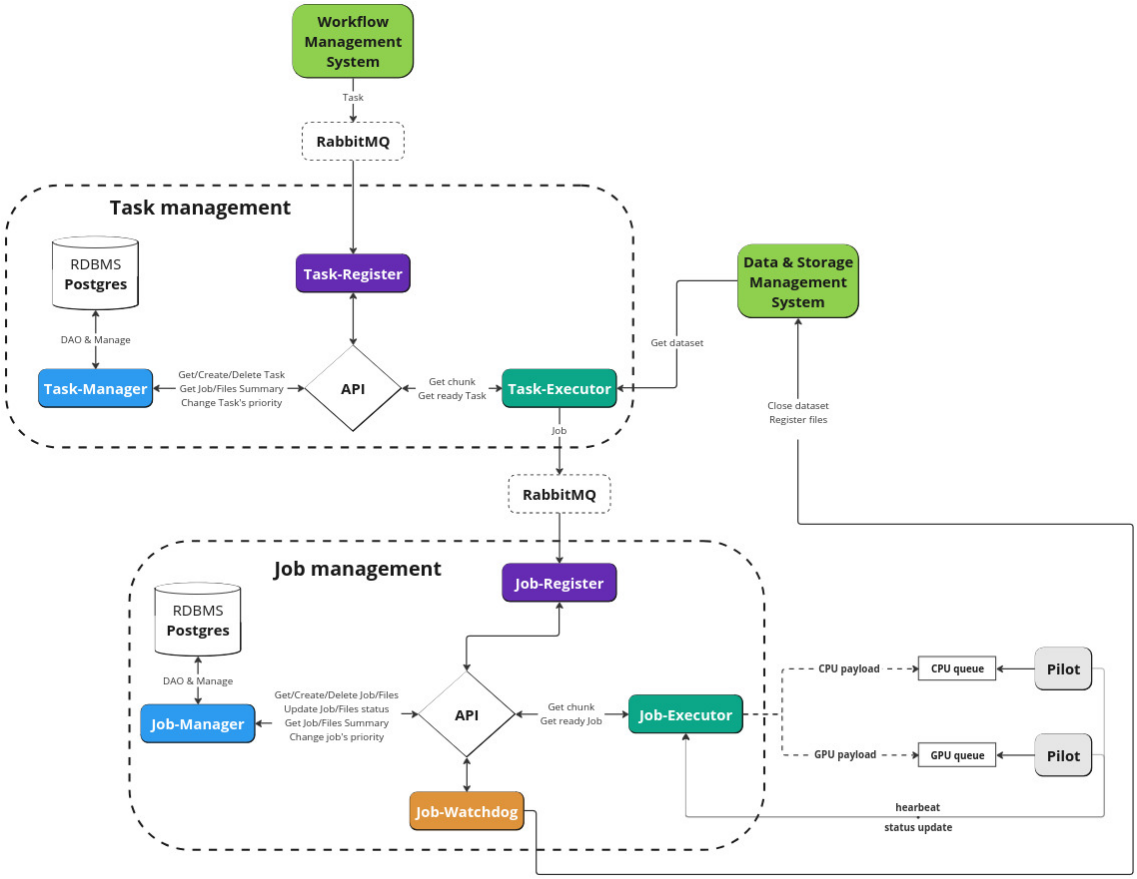

Fig. 1. Workload Management System High-Level Architecture

PILOT AGENT

The Pilot is a lightweight agent script that is executed and managed by a persistent UNIX daemon on compute nodes (see Fig. 2). Its main goals are to orchestrate the execution of jobs by preparing the environment, running jobs in isolated subprocesses, analyzing results, and reporting statuses to the Workload Management System. This involves setting up and tearing down the execution environment, handling input/output file transfers, and storing logs and results. Additionally, Pilot aims to provide continuous monitoring and reporting of job state, handle external control commands (cancelling jobs), validate jobs before execution, manage unexpected failures, and maintain system resilience despite individual job errors. The daemon ensures Pilot is always up-to-date by downloading the latest version from a repository and restarting it after each execution or failure, enabling fault-tolerant and continuously operational job processing.

During the preprocessing stage Pilot performs the following operations: initializes logging mechanisms to capture operational details and job-specific information; loads necessary configuration parameters, including node specifics; retrieves a job from the message queue provided by the Job Executor which corresponds to available co-proces-

sors on a node; and ensures the retrieved job is valid before processing (e.g., correct job format; dependencies available).

After the preprocessing stage Pilot starts a new thread to manage its execution. It starts a job initialization or environment set up, executes scripts to create the environment for job execution, after which retrieves input files from the designated input storage system. The job is started in a separate subprocess, isolating its execution from the main Pilot process. After execution, the job's output is analyzed to determine if it was successful or if errors occurred. On any significant job status change or internal update, a Pilot sends a status update message to the WMS, which is collected by the ASGI web server [7]. The payload for a job at this stage is artificial, as it requires applied software and simulated data to be ready.

Workload Management System can terminate the active job thread of a Pilot if the corresponding task is canceled by the Workflow Management System or if an error occurs that prevents the job from being completed successfully.

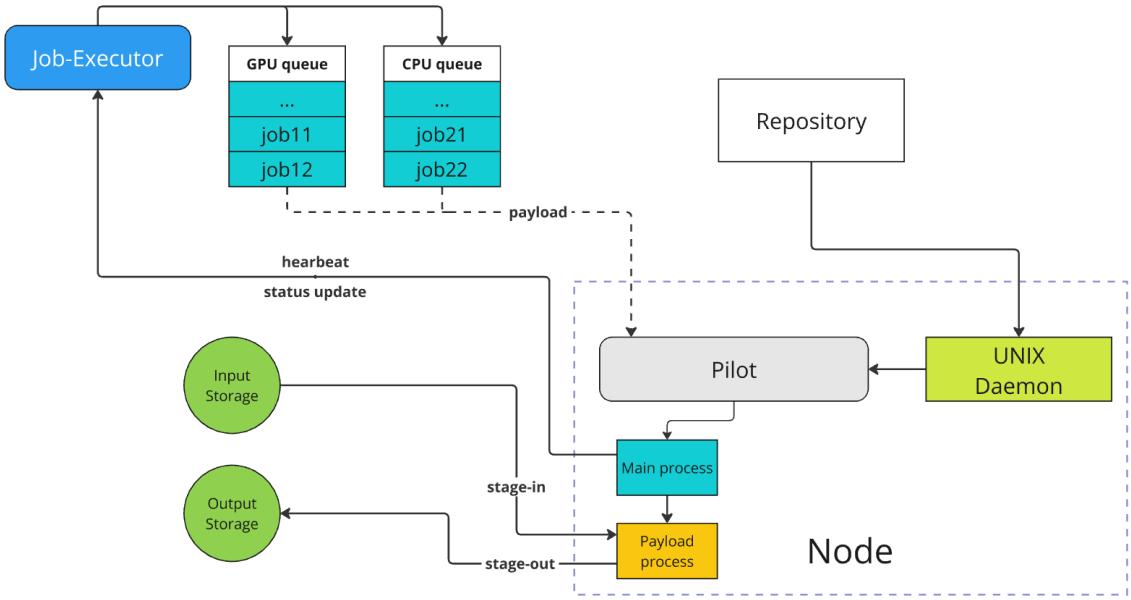

Fig. 2. Pilot Agent Architecture

CONCLUSIONS

Some clarifications and changes were made to the inter-service interaction scenarios, which entailed defining API contracts with the Workflow Management System and the Data & Storage Management System. The architecture of the Workload Management System underwent a slight change, introducing message gateway and job monitoring microservices. The middleware backbone and main microservices were implemented, and the project was decomposed into a set of reusable modules that can be shared not only within the Workload Management System, but also within the «SPD Online filter» as a whole. The internal architecture of the pilot agent was designed and implemented in conjunction with the UNIX daemon.

The next phase of development will entail the execution of the entire workflow, which constitutes a chain of interdependent tasks established at the level of the Work-

flow Management System. Another objective is to integrate with the applied software and execute a workflow on a simulated data.

REFERENCES

1. *V.M. Abazov et.al* [SPD Collaboration] Technical Design Report of the Spin Physics Detector at NICA // arXiv: 2404.08317 [hep-ex]
2. *Sadiku M., Eze K., Musa S.*, High-Throughput Computing. // International Journal of Trend in Research and Development. – 2018 – Volume 5 – Issue 4 – ISSN: 2394-9333
3. *Greben, N., Romanychev, L., Oleynik, D., Degtyarev, A.* SPD On-Line Filter: Workload Management System and Pilot Agent. Phys. Part. Nuclei 55, 612–614 (2024)
4. *Tereschenko, D., Ponomarev, E., Oleynik, D., Korkhov, V.* SPD On-Line Filter: Workflow and Data Management Systems. Phys. Part. Nuclei 55, 603–605 (2024)
5. RabbitMQ Message Broker [Electronic resource] // URL: https://www.rabbitmq.-com/ (accessed: 22.11.2024)
6. PostgreSQL: The World's Most Advanced Open Source Relational Database [Electronic resource] // URL: https://www.postgresql.org/ (accessed: 22.11.2024)
7. ASGI (Asynchronous Server Gateway Interface) Specification [Electronic resource] // URL: https://asgi.readthedocs.io/en/latest/specs/main.html (accessed: 22.11.2024)