SPD Online Filter

High-Throughput Processing Middleware

Nikita Greben

Meshcheryakov Laboratory of Information Technologies

SPD experiment at NICA collider

Detector

- Polarized proton and deuteron beams.
- Collision energy up to 27 GeV.
- Luminosity up to 10^{32} cm^{-2} s^{-1}.
- Bunch crossing every 80 ns = crossing rate 12.5 MHz.

Key Challenges

- Number of registration channels in SPD $\approx$ 500000.
- Physics signal selection requires momentum and vertex reconstruction $\rightarrow$ no simple **trigger** is possible.
- The goal of the online filter is to reduce the data stream so that the annual increase in data, including modeled samples, does not exceed 10 PB.

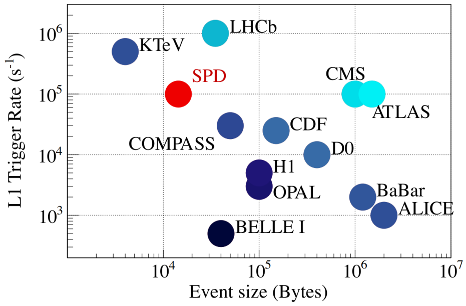


Рис. 1: Expected event size and event rate of the SPD setup after the online filter, compared with some other experiments.

SPD DAQ

Triggerless DAQ

Triggerless DAQ means that the output of the system is not a set of raw events, but a set of signals from sub-detectors organized into time slices.

- DAQ provide data organized in time frames which placed in **files** with reasonable size (a few GB).
- Each of these file may be processed independently as a part of top-level **workflow chain**.
- No needs to exchange of any information during handling of each initial file, but results of may be used as input for next step of processing.

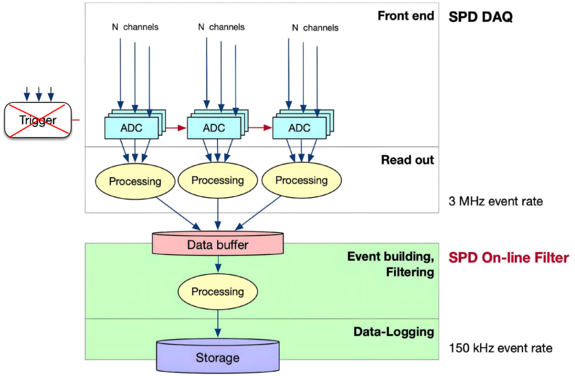


Рис. 2: Triggerless dataflow in SPD

SPD Online Filter

SPD Online Filter is a **primary** data processing facility designed for the high-throughput, multi-step processing of data from the SPD detector.

Hardware component

Compute cluster with two storage systems and set of working nodes: multi-CPU and hybrid multi CPU + Neural Network Accelerators (GPU, FPGA etc.)

Middleware component

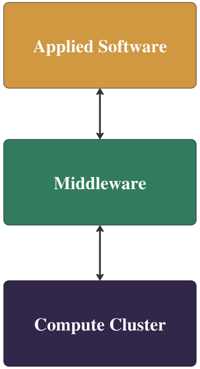
Software complex for management of multistep data processing and efficient loading (usage) of computing facility.

Applied software

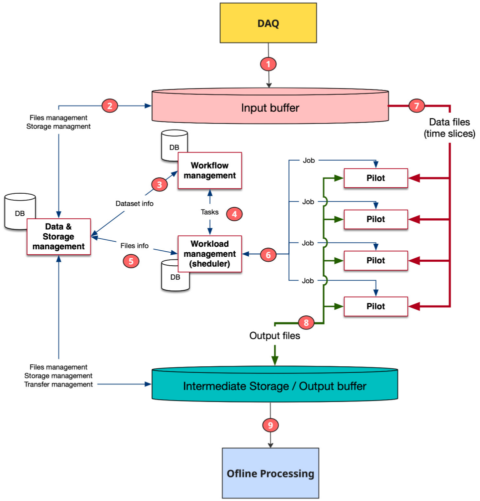
Performs informational processing of data.

SPD Online Filter Middleware

- **Data Management System**
  - Data lifecycle support (data catalog, consistency check, cleanup, storage);
- **Workflow Management System**
  - Define and execute data processing chains by generating the required number of computational tasks;
- **Workload management System**
  - Create the required number of processing jobs to perform the task;
  - Control job execution through pilots working on compute nodes;
  - Handles efficient use of resources.

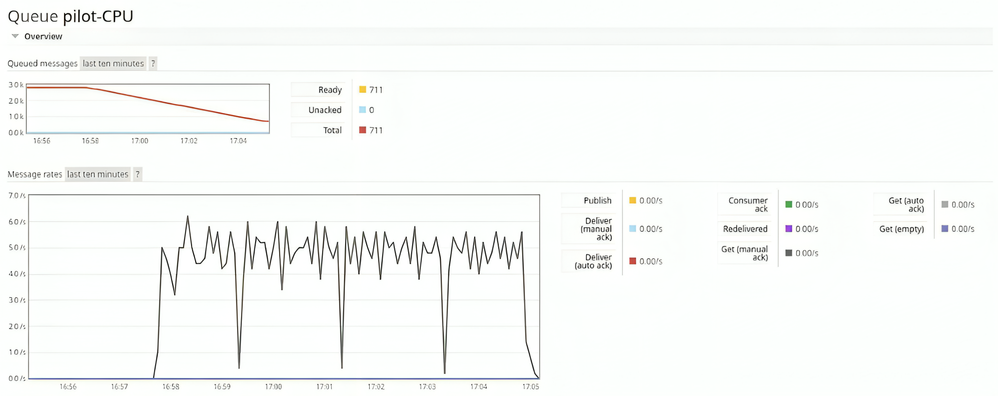


High-level architecture of SPD Online Filter



1. Data taking from DAQ;
2. Primary datasets registering;
3. Workflows initiating;
4. Task generation;
5. Acquisition the files metadata;
6. Job generation and dispatching;
7. Data stage-in;
8. Data stage-out;
9. Uploading for long-term storage and analysis.

First “load testing”



Queue pilot-CPU

Overview

Queued messages last ten minutes ?

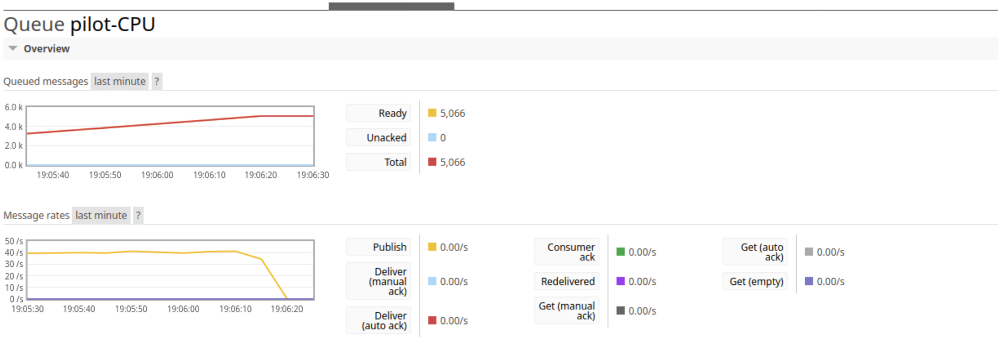
Ready	🟨 711
Unacked	🟦 0
Total	🟥 711

Message rates last ten minutes ?

Publish	🟨 0.00/s	Consumer ack	🟩 0.00/s	Get (auto ack)	⬜ 0.00/s
Deliver (manual ack)	🟦 0.00/s	Redelivered	🟪 0.00/s	Get (empty)	🟦 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s		

Рис. 3: 100 concurrent Pilots processed ≈2,100 jobs in 7 minutes (≈15 s/job including stage-in/out) on standard JINR Cloud VMs using a simplified synthetic payload.

First “load testing”



Queue **pilot-CPU**

Overview

Queued messages last minute ?

Ready	🟨 5,066
Unacked	🟦 0
Total	🟥 5,066

Message rates last minute ?

Publish	🟨 0.00/s	Consumer ack	🟩 0.00/s	Get (auto ack)	⬜ 0.00/s
Deliver (manual ack)	🟦 0.00/s	Redelivered	🟪 0.00/s	Get (empty)	🟪 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s		

Рис. 4: **Workload Management System** generates ≈ 5000 jobs in less than a minute.

Conclusions

- **Codebase and deployment**
  - Around ≈25 000 lines of code for the entire SPD Online Filter Middleware;
  - Full deployment requires ≈16 Docker containers: one container per microservice;
  - Integration infrastructure deployed on JINR Cloud resources on 9 VM's.
- **Task and workflow processing has been achieved**
  - Execution of the entire workflow set up on the level of Workflow Management System;
  - The major cycle of refactoring and test coverage is required.

Next steps and milestones

- **Middleware and applied software integration**
  - Requires prototyped applied software and simulated data;
  - Non-functional requirements for applied software;
  - Move to the execution of the jobs on the pilot with a "real"payload.
- **Middleware deployment and release management**
  - Focus on shipping SPD Online Filter as standalone software;
  - Work on the deployment on the **upcoming testbed** (256 CPU Cores, 1TB RAM, 120TB HDD);
  - Select the appropriate release management strategy.

Thank you for your attention!

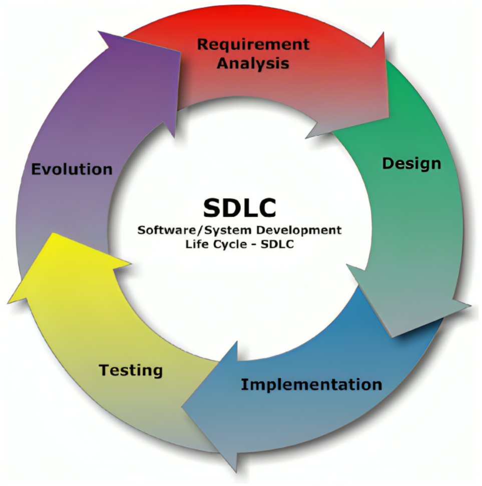


Рис. 5: The never-ending cycle of everyday life

Backup slides

SPD Online Filter

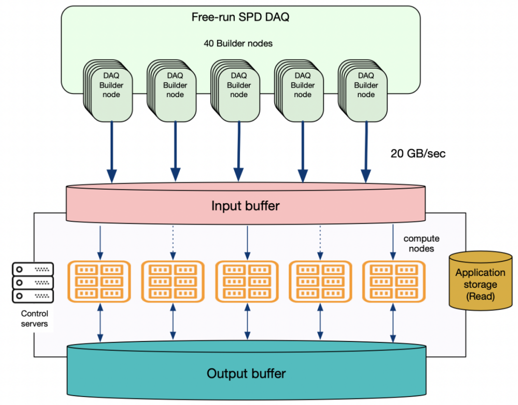


Рис. 6: SPD Online Filter Facility

Middleware

Definition

An intermediate software layer that connects hardware resources and application services. **Primary purpose** is to abstract the complexity of the compute cluster and provide a unified interface for application software.

- **Key Functions**
  - Data management
  - Coordination of multi-stage workflows
  - Efficient workload management, usage of computing resources.
- **Role in SPD Online Filter:** bridges the dedicated compute cluster and applied software, enabling a configurable and scalable data-processing pipeline.

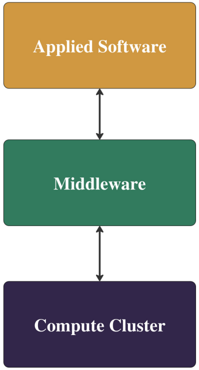


Рис. 7: Middleware

SPD DAQ

Data Acquisition System

The **DAQ** system takes raw data from detector sensors and ensures that only the most interesting events (collisions) are recorded for later analysis, while discarding unimportant ones due to limitations in storage and processing.

Triggerless DAQ

Triggerless DAQ means that the output of the system is not a set of raw events, but a set of signals from sub-detectors organized into time slices.

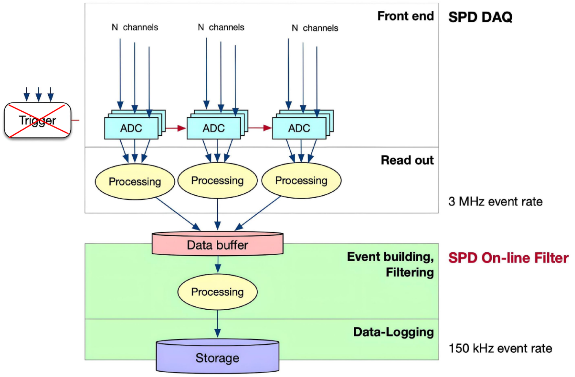


Рис. 8: Triggerless dataflow in SPD

SPD DAQ data

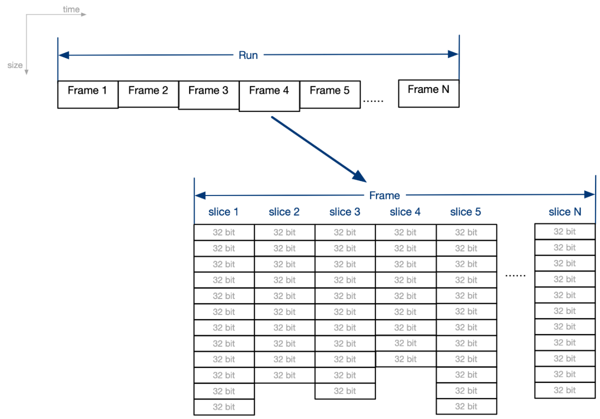


Рис. 9: Structure of SPD DAQ data

- DAQ provide data organized in time frames which placed in **files** with reasonable size (a few GB).
- Each of these file may be processed independently as a part of top-level **workflow chain**.
- No needs to exchange of any information during handling of each initial file, but results of may be used as input for next step of processing.

SPD DAQ data 2

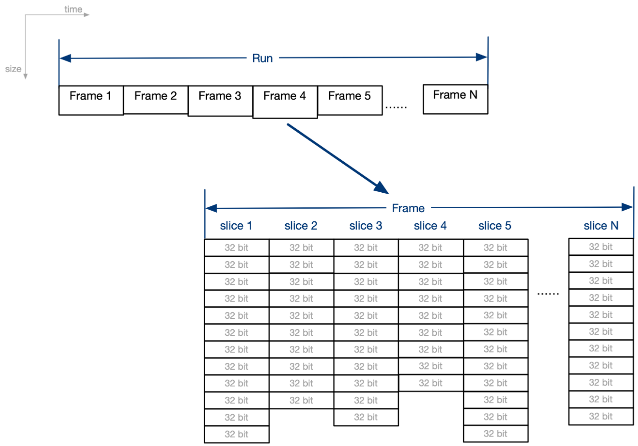


Рис. 10: Structure of SPD DAQ data

- The output of the system will not be a dataset of raw events, but a set of signals from detectors organized in time slices
- Primary data unit: time slice (10 s) Time slices combined in time frames (1-10 sec.)
- Every slice will contain signals from a few to many collisions (events)
- Event building have to unscramble events from a series of time slices

Data Processing in SPD Online Filter

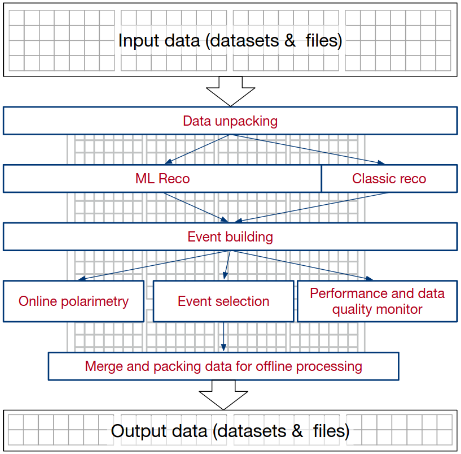

Input data (datasets & files)
Data unpacking
ML Reco
Classic reco
Event building
Online polarimetry
Event selection
Performance and data quality monitor
Merge and packing data for offline processing
Output data (datasets & files)

Рис. 11: Example of multi-step processing scheme

High-throughput computing

Definition

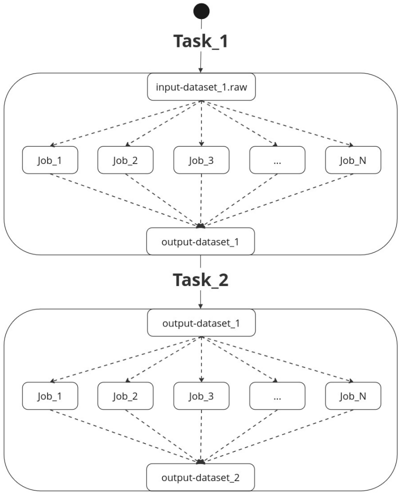
The European Grid Infrastructure defines **HTC** as "a computing paradigm that focuses on the efficient execution of a large number of loosely-coupled tasks".

Focus

Maximizing the number of tasks processed per unit of time.

Reliability

HTC systems are mostly designed to provide high reliability and make sure that all tasks run efficiently even if any one of the individual components fails.



Data Management System

dsm-register (data registration)

A service that receive requests for adding/deleting data in the system asynchronously (via Message Broker). Then the service makes changes to the data catalog via the API of the dsm-manager.

dsm-manager (REST API of data catalog)

File and dataset management (adding data to a database, changing data, deleting data).

dsm-inspector (daemon tasks)

Delete files on storage, check consistency of files, file upload control, monitoring the use of storage (for example, "dark"data).

Data Management System

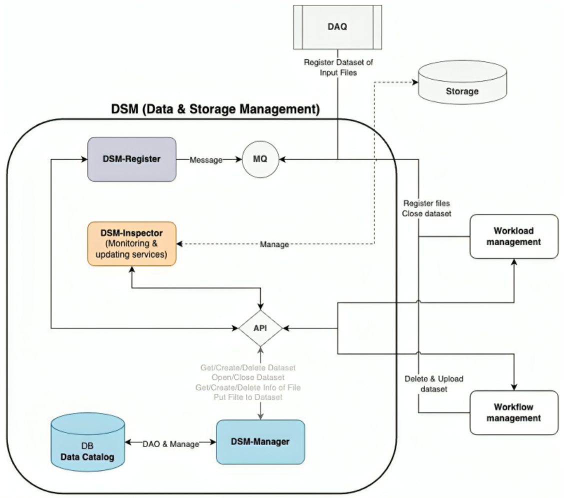

DAQ
Register Dataset of Input Files
Storage
DSM (Data & Storage Management)
DSM-Register
Message
MQ
Register files Close dataset
Workload management
DSM-Inspector (Monitoring & updating services)
Manage
API
Get/Create/Delete Dataset
Open/Close Dataset
Get/Create/Delete Info of File
Put Fiile to Dataset
Delete & Upload dataset
Workflow management
DB Data Catalog
DAO & Manage
DSM-Manager

Рис. 12: Architecture of Data Management System

Workflow Management System

Responsibilities

Workflow Management System is a top-level component responsible for defining and orchestrating data-processing workflows and for managing both intermediate and final datasets. It retrieves input datasets, maps them to CWL templates, generates and dispatches tasks for execution, and oversees the entire dataset lifecycle.

- Retrieves input datasets from Data Management System;
- Maps these datasets with the appropriate CWL template;
- Generates the workchain from this template;
- Generates tasks and sends them to the Workload Management System for further execution;
- Oversees datasets: decision making for creation, closure, deletion;
- Manages the concurrent execution of workchains and tasks.

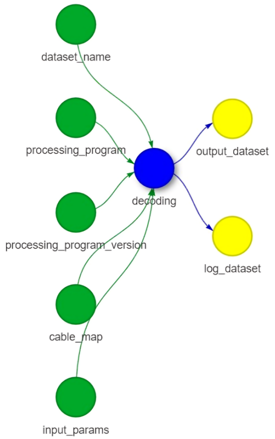


Рис. 13: Task description

Workflows execution

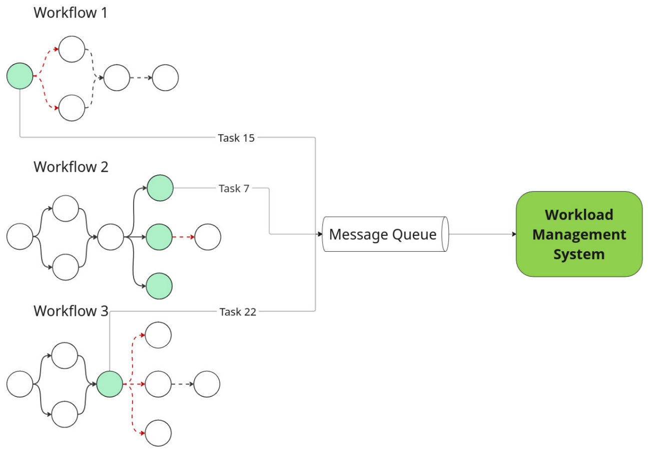


Рис. 14: An example of the concurrent execution of several workflows.

Workload Management System

Responsibilities

The **Workload Management System** is responsible for partitioning each task into individual processing jobs, dispatching those jobs to **Pilot** agents on compute nodes, and monitoring their execution. It ensures efficient resource utilization by generating the appropriate number of jobs, tracking status, handling retries or failures, and aggregating output files into new datasets.

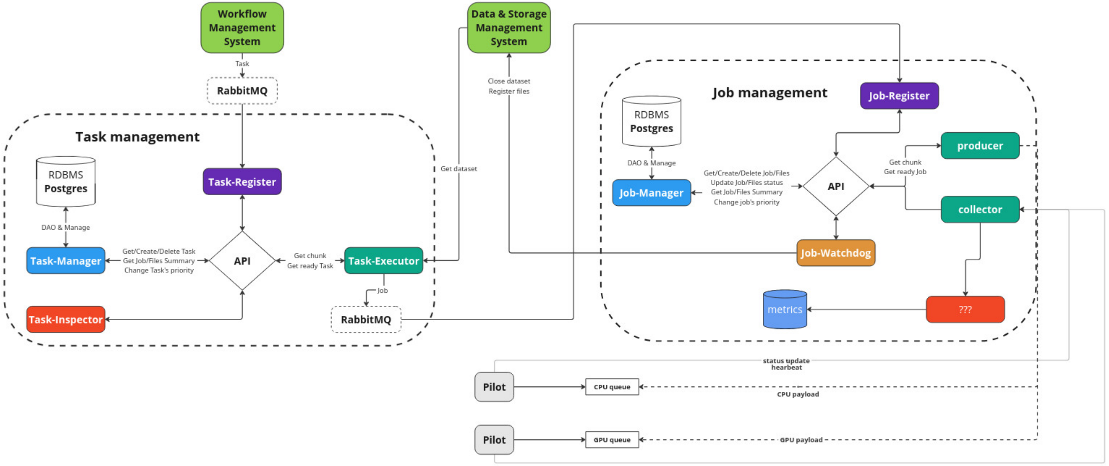
Workload Management System as Main Scheduling Mechanism

Role of Scheduling in WMS

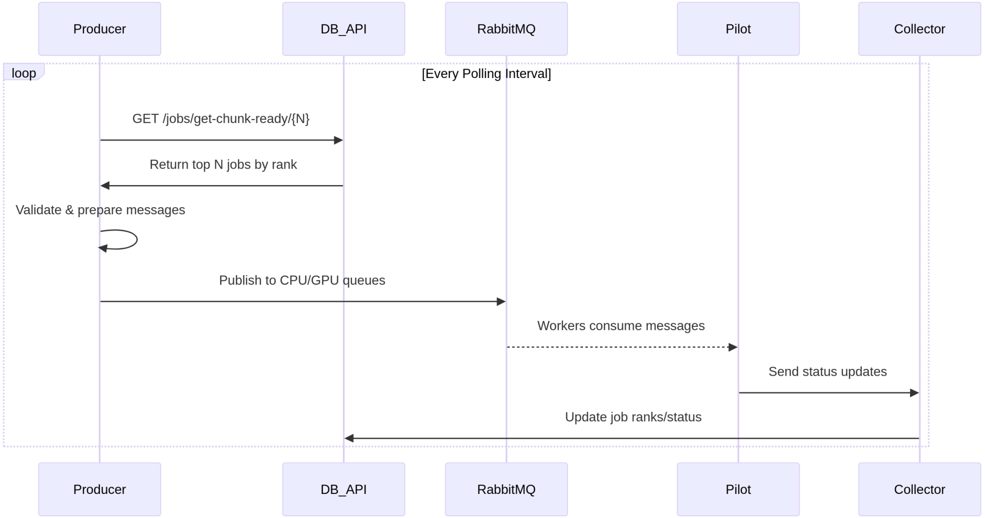
In the Workload Management System (WMS), scheduling fulfills two primary functions:

- Partitioning each task (dataset) into quanta for job generation based on dataset priority – IWRR based scheduler.
- Distributing ready jobs to compute nodes (Pilots) according to job priority – rank-based scheduler.

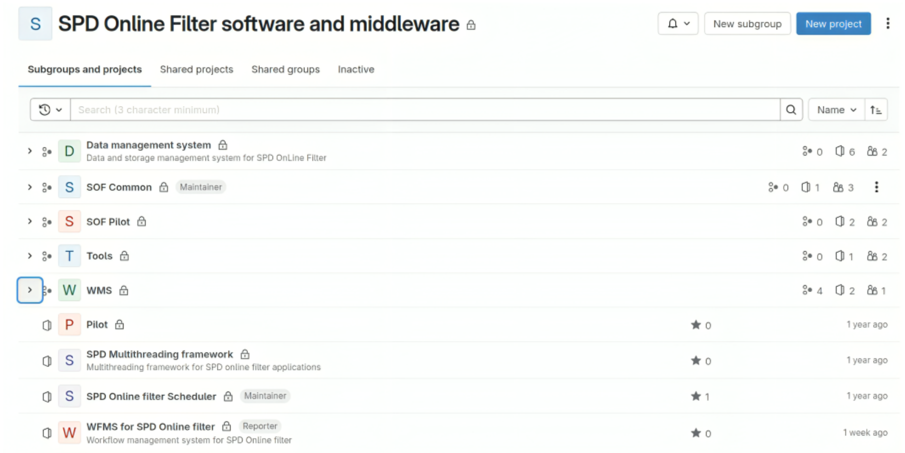
Workload Management System Architecture



Rank-Based Job Distribution Scheduler



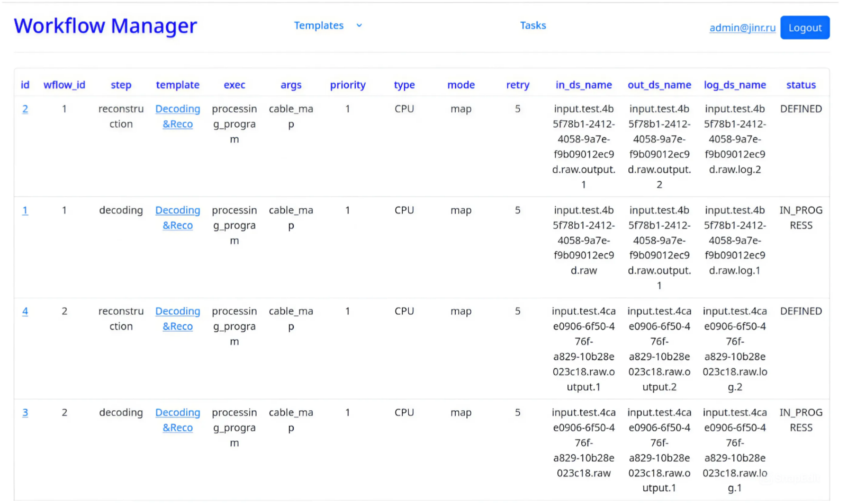
SPD Online Filter middleware codebase



S SPD Online Filter software and middleware

Data management system — Data and storage management system for SPD OnLine Filter	0	6	2	
SOF Common (Maintainer)	0	1	3	
SOF Pilot	0	2	2	
Tools	0	1	2	
WMS	4	2	1	
Pilot	★ 0			1 year ago
SPD Multithreading framework — Multithreading framework for SPD online filter applications	★ 0			1 year ago
SPD Online filter Scheduler (Maintainer)	★ 1			1 year ago
WFMS for SPD Online filter (Reporter) — Workflow management system for SPD Online filter	★ 0			1 week ago

Workflow Management System Control Panel



id	wflow_id	step	template	exec	args	priority	type	mode	retry	in_ds_name	out_ds_name	log_ds_name	status
2	1	reconstruction	Decoding &Reco	processing_program	cable_map	1	CPU	map	5	input.test.4b5f78b1-2412-4058-9a7e-f9b09012ec9d.raw.output.1	input.test.4b5f78b1-2412-4058-9a7e-f9b09012ec9d.raw.output.2	input.test.4b5f78b1-2412-4058-9a7e-f9b09012ec9d.raw.log.2	DEFINED
1	1	decoding	Decoding &Reco	processing_program	cable_map	1	CPU	map	5	input.test.4b5f78b1-2412-4058-9a7e-f9b09012ec9d.raw	input.test.4b5f78b1-2412-4058-9a7e-f9b09012ec9d.raw.output.1	input.test.4b5f78b1-2412-4058-9a7e-f9b09012ec9d.raw.log.1	IN_PROGRESS
4	2	reconstruction	Decoding &Reco	processing_program	cable_map	1	CPU	map	5	input.test.4cae0906-6f50-476f-a829-10b28e023c18.raw.output.1	input.test.4cae0906-6f50-476f-a829-10b28e023c18.raw.output.2	input.test.4cae0906-6f50-476f-a829-10b28e023c18.raw.log.2	DEFINED
3	2	decoding	Decoding &Reco	processing_program	cable_map	1	CPU	map	5	input.test.4cae0906-6f50-476f-a829-10b28e023c18.raw	input.test.4cae0906-6f50-476f-a829-10b28e023c18.raw.output.1	input.test.4cae0906-6f50-476f-a829-10b28e023c18.raw.log.1	IN_PROGRESS

High-throughput computing

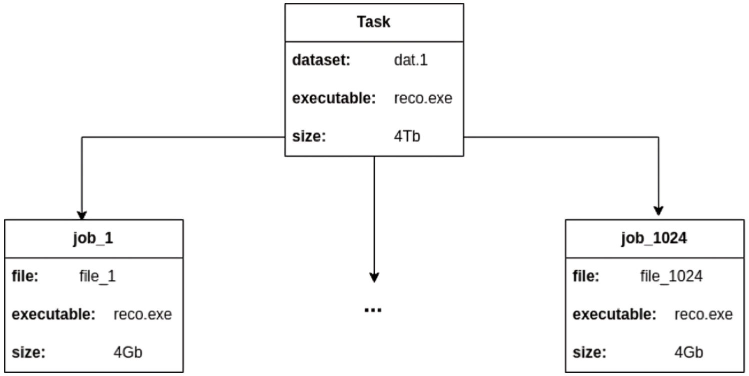


Рис. 15: Task-job relationship

Interleaved Weighted Round-Robin (IWRR) Based Scheduler

Algorithm Description

The **IWRR scheduler** apportions “job-generation quanta” among active datasets in proportion to their assigned weights (ranks). Let

$$\mathcal{D} = \{ D_1, D_2, \ldots, D_k \}$$

denote the set of datasets marked `running`, each with integer rank $\boldsymbol{w_i} \geq \mathbf{1}$. Define

$$W = \max_i w_i .$$

The scheduler iterates over rounds

$$r = 1, 2, \ldots, W.$$

In round r, it selects every dataset D_i satisfying $\boldsymbol{w_i} \geq \boldsymbol{r}$. For each such D_i, exactly one job is created for the next unprocessed file in D_i's input partition. That job is immediately enqueued into RabbitMQ. Once all files belonging to D_i have been assigned, D_i is removed from the active set. By construction, higher-ranked datasets participate in more of the W rounds, thereby receiving a proportionally larger share of job-generation slots, while lower-ranked datasets still obtain at least one slot each. This ensures a balance between **fairness** (each dataset D_i appears in exactly $\min(w_i, W)$ rounds) and **priority** (frequency of job creation scales linearly with $\boldsymbol{w_i}$).

Interleaved Weighted Round-Robin (IWRR) Based Scheduler

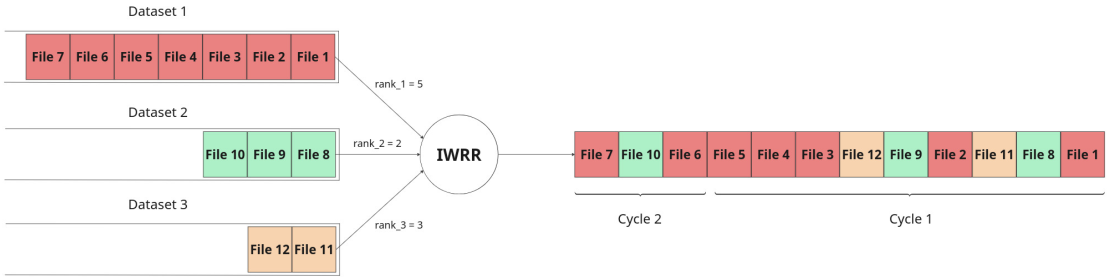


Рис. 16: An example of the job generation procedure across several datasets being processed concurrently.

Rank-Based Job Distribution Scheduler

Algorithm Description

The **Rank-Based Job Distribution** scheduler repeatedly fetches a batch of `ready` jobs from the database, ordered by descending rank. Formally, let

$$\mathcal{J} = \{ J \mid \text{status}(J) = \text{“ready”} \}.$$

At each polling interval, the Producer issues a query to the Job Manager API:

```
GET /jobs/get-chunk-ready/{N},
```

requesting the top N jobs sorted by rank r_j (higher means more urgent). The API returns a list $\{ J_1, J_2, \ldots, J_N \}$ with

$$r_{J_1} \geq r_{J_2} \geq \ldots \geq r_{J_N}.$$

The Producer then validates each J_i and encapsulates it into a message, which is published to the appropriate RabbitMQ queue (e.g., `cpu` or `gpu`). Pilot agents consume these messages, upon completion, a Pilot sends a status update to the Collector, which invokes the Job Manager API to update the job's status and, if necessary, adjust its rank. Scheduler guarantees that more critical computations are dispatched before less critical ones, achieving a **priority-driven** workflow.

Rank-Based Job Distribution Scheduler

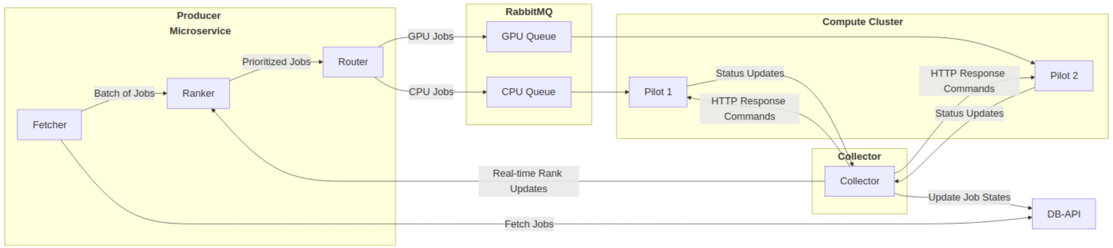


Рис. 17: Simplified diagram of the Job Distribution Scheduler's working process.

Next Life Plan - Control Theory's Dynamic Adaptability Scheduler

- Each dataset has a rank (priority) that determines its processing order;
- Tasks are processed in priority order, with dynamic updates to maintain system responsiveness;
- **Priority-based job scheduling mechanism** is expected, with rank update scheme involving **Control Theory** (option to be explored later);
- Not applicable at this stage of the development process.

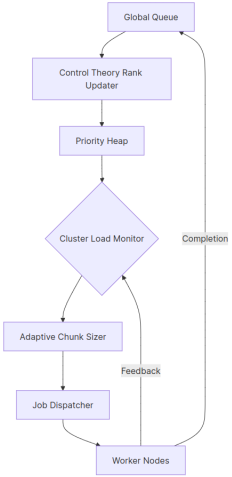
$$r_{i+1} = \underbrace{\alpha \ln(x_i + 1)}_{\text{Aging}} - \underbrace{\beta 2^{y_i}}_{\text{Retry Penalty}} + \underbrace{\gamma r_i}_{\text{History}} + \underbrace{\delta(1 - L)}_{\text{Load}}$$

$$\mathbf{r}_{i+1} = \Gamma \mathbf{r}_i + \alpha \ln(\mathbf{x}_i + \mathbf{1}) - \beta \cdot 2^{\mathbf{y}_i} + \delta(1 - L)\mathbf{1}$$

$\Gamma = \mathrm{diag}(\gamma_1, \dots, \gamma_N)$ (job-specific history weights)

$\mathbf{x}_i = [x_i^{(1)}, \dots, x_i^{(N)}]^\top$ (job ages)

$\mathbf{y}_i = [y_i^{(1)}, \dots, y_i^{(N)}]^\top$ (retry counts)


Global Queue
Control Theory Rank Updater
Priority Heap
Cluster Load Monitor
Completion
Adaptive Chunk Sizer
Feedback
Job Dispatcher
Worker Nodes