A Unified Middleware

for Data, Workflow, and Workload Management in SPD Online Filter

Nikita Greben

Meshcheryakov Laboratory of Information Technologies

29th International Scientific Conference of Young Scientists and Specialists (AYSS-2025)

October 27-31, 2025

SPD experiment at NICA collider

Specifications

- Collision energy up to 27 GeV.
- Luminosity up to 10^{32} cm^{-2} s^{-1}.
- Bunch crossing every 76 ns = crossing rate 13.1 MHz.
- Collision rate 3–4 MHz (1st stage about 100 kHz).

Key Challenges

- The wide SPD physical program eliminates a possibility of rejecting events at a hardware level.
- Physics signal selection requires momentum and vertex reconstruction → 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.

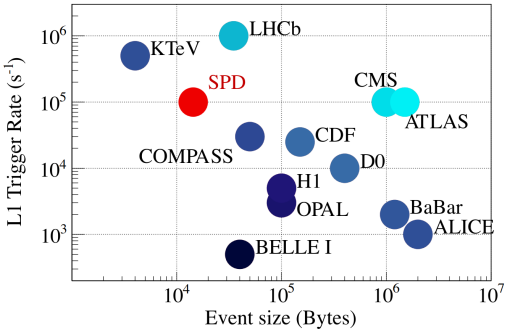


Figure 1: Expected event size and event rate of the SPD setup after the online filter, compared with some other experiments.

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.

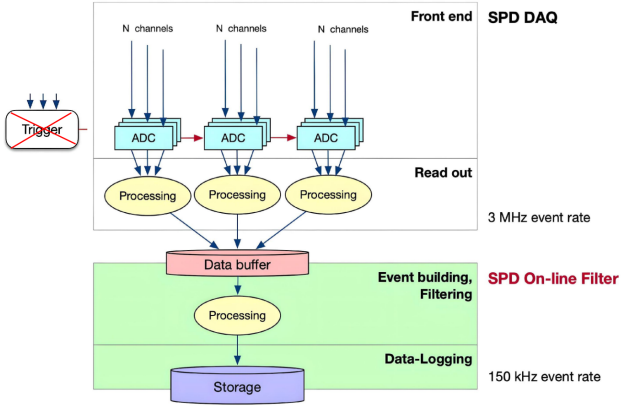


Figure 2: Triggerless dataflow in SPD

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

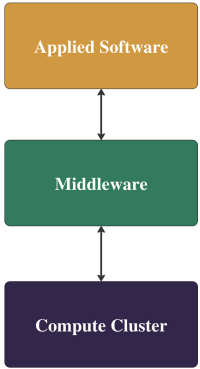
Software complex for management of multistep data processing and efficient loading (usage) of computing facility.

Applied software

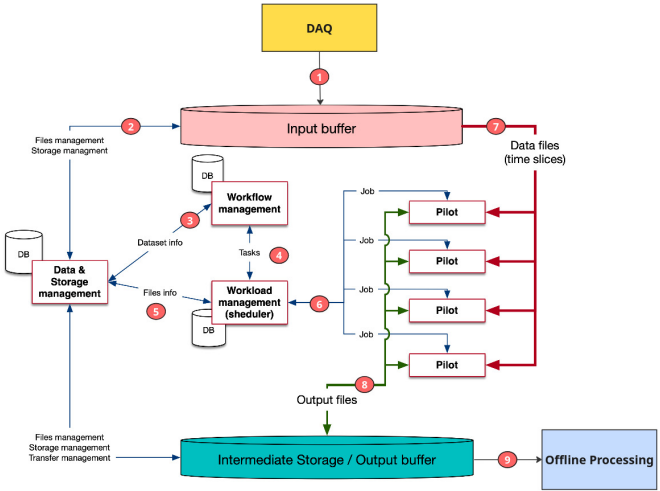
Performs informational processing of data.

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



High-throughput computing

Definition

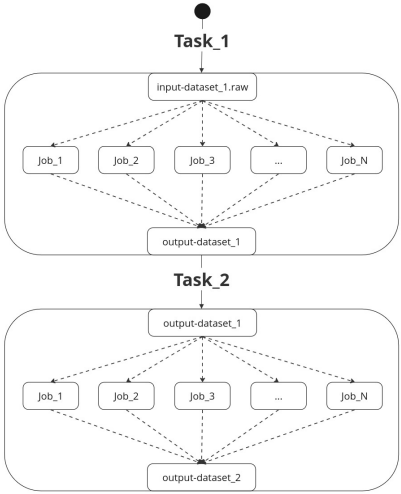
The European Grid Infrastructure defines **HTC** as "a computing paradigm that focuses on the efficient execution of a large number of loosely-coupled tasks".

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 Processing in SPD Online Filter

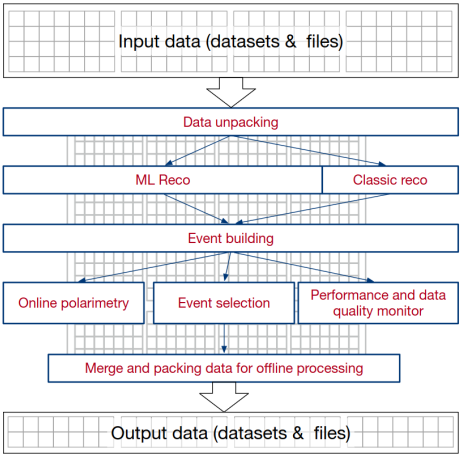
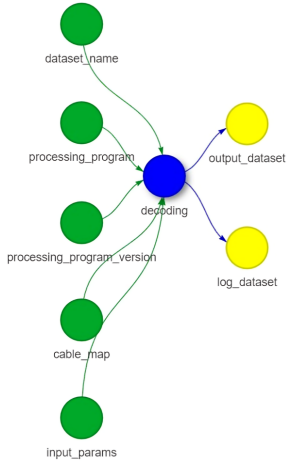


Figure 3: Example of multi-staged processing scheme

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.



Workflow Management System

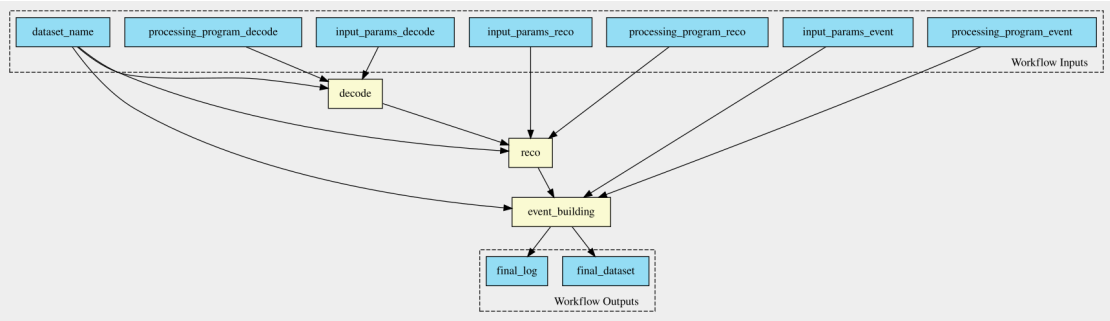


Figure 4: Workflow Definition via CWL (Common Worfklow language). Workflow Management System serves as a sort of high level interpreter, executing the **DAGs** in large-scale.

Workflow Execution Scheme

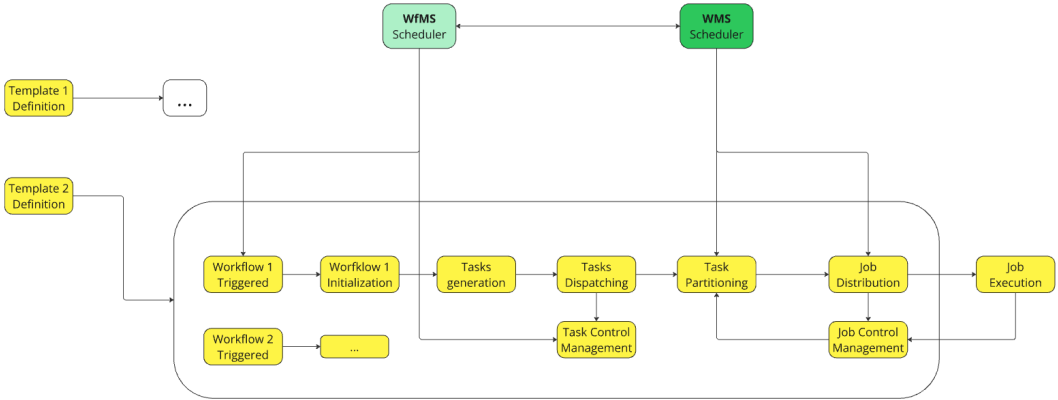


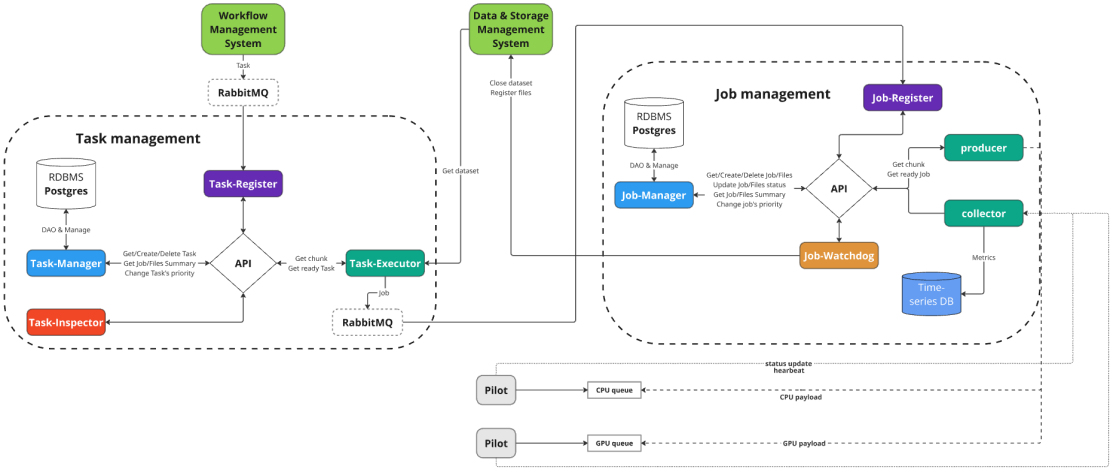
Figure 5: An example of the state execution of several workflows triggered by different templates and managed by schedulers.

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 Architecture



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.

Interleaved Weighted Round-Robin (IWRR) Based Scheduler

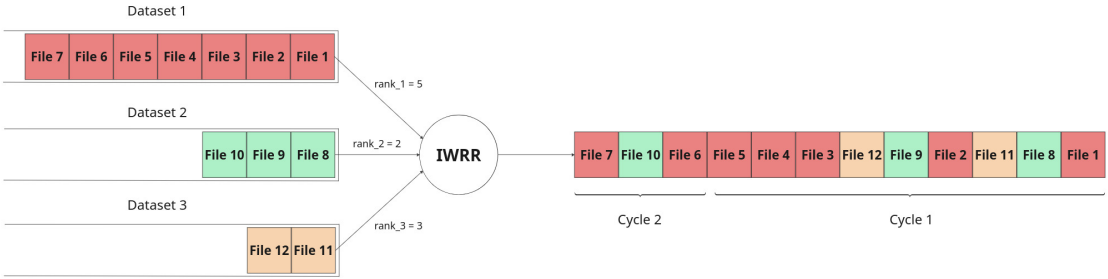


Figure 6: An example of the job generation procedure across several datasets being processed concurrently.

Rank-Based Job Distribution Scheduler

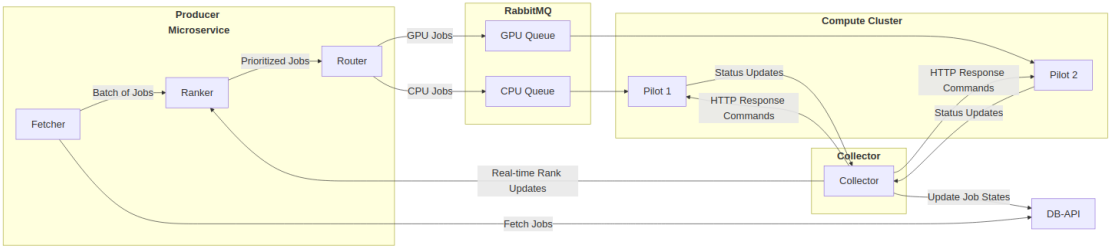


Figure 7: Simplified diagram of the Job Distribution Scheduler's working process.

First "end-to-end testing"

Figure 8: Completed one three-step workflow on standard JINR Cloud VMs using a simplified synthetic payload.

First "end-to-end testing"

Figure 9: An example of the simultaneous execution of **ten workflows** on **ten primary** datasets. Each bump represents a new batch of jobs that are ready to be executed and have been published by the scheduler.

Summary

- **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.
- **Workflow processing has been achieved**
  - Execution of the entire workflow set up on the level of Workflow Management System;
  - Simultaneous execution of several workflows.

Next steps and milestones

- **Observability and monitoring**
  - Collecting requirements for monitoring system;
  - Started development of “SPD DAQ Emulator service” which should allow us to perform complex loading tests;
  - Perform tests on the **WMS scheduler**.
- **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.
- **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.

Thank you for your attention!

Backup slides

First “load testing”

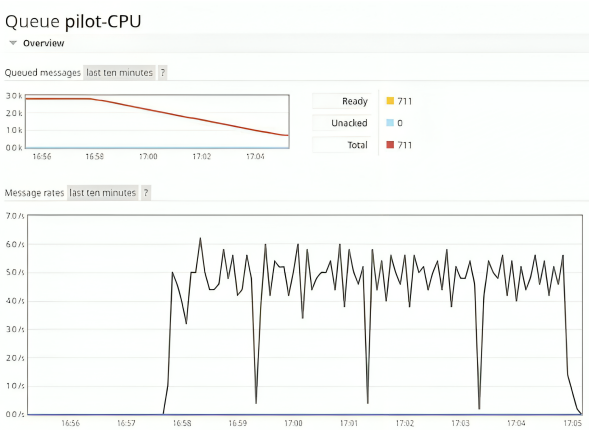


Figure 10: 100 concurrent Pilots processed ≈2,100 jobs in 7 minutes (≈15 s/job including stage-in/out) on standard JINR Cloud VMs using a simplified synthetic payload.

First “load testing”

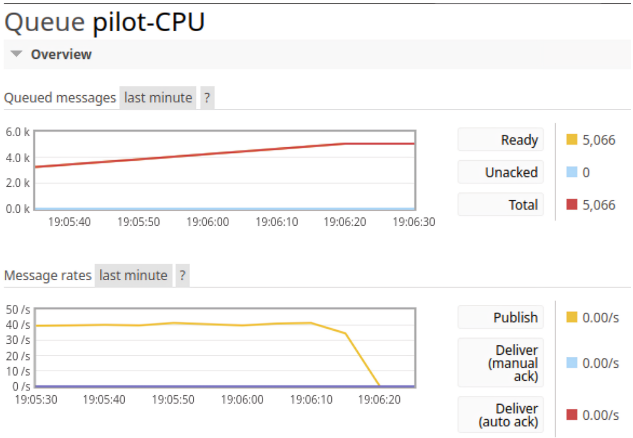


Figure 11: **Workload Management System** generates $\approx$ 5000 jobs in less than a minute.

SPD Online Filter

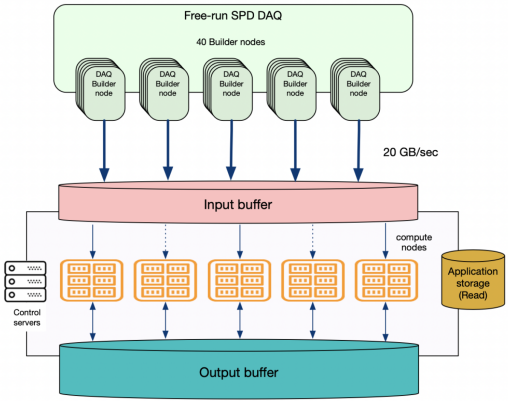


Figure 12: SPD Online Filter Facility

Tasks and Jobs relationship

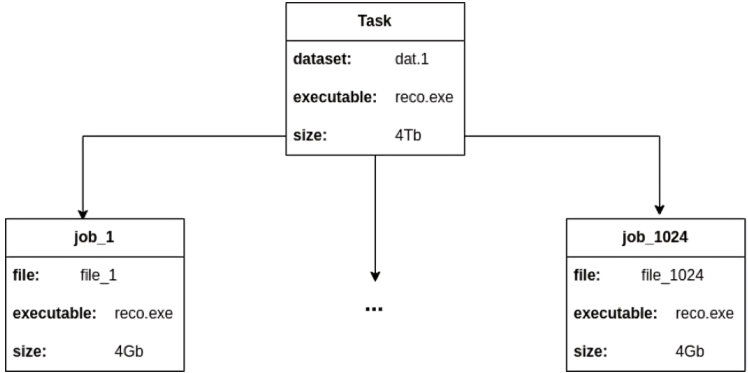


Figure 13: The goal of the entire task is to completely process the input dataset. Each job is assigned a file for further processing. The sum of the resulting output data from all jobs is the output of the entire task

Analogy with an Operating System

OS Component	SPD Online Filter	Explanation
Kernel	Middleware (WMS, WfMS, DSM)	The middleware manages resources, schedules tasks/jobs, and handles data flow, similar to an OS kernel coordinating system operations.
Scheduler	Task Executor in WMS	The task executor distributes jobs to pilots using IWRR based on task ranks, similar to an OS scheduler's role in allocating CPU time to processes/threads.
File System	Data & Storage Management (DSM)	DSM manages data lifecycle (cataloging, consistency, storage), similar to an OS file system handling file storage and access.
Inter-Process Communication (IPC)	RabbitMQ/Message Broker	RabbitMQ facilitates communication between components (e.g., task executor to job register, producer to pilots).
System Calls	REST APIs (to WfMS, WMS, DSM)	The middleware uses REST APIs for communication between its microservices. This is similar to how an OS uses system calls or kernel APIs to coordinate between kernel components or provide services to user processes.

Workflows execution

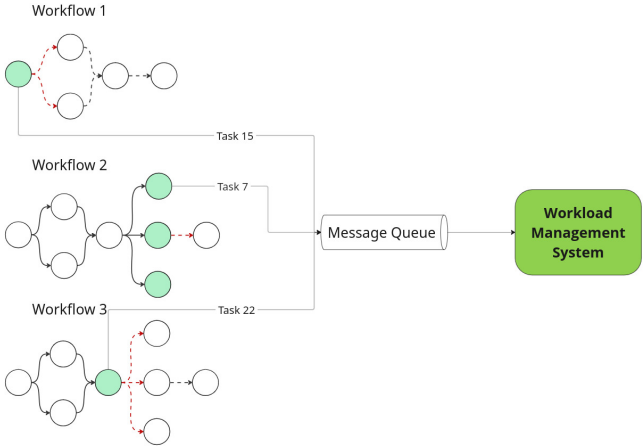
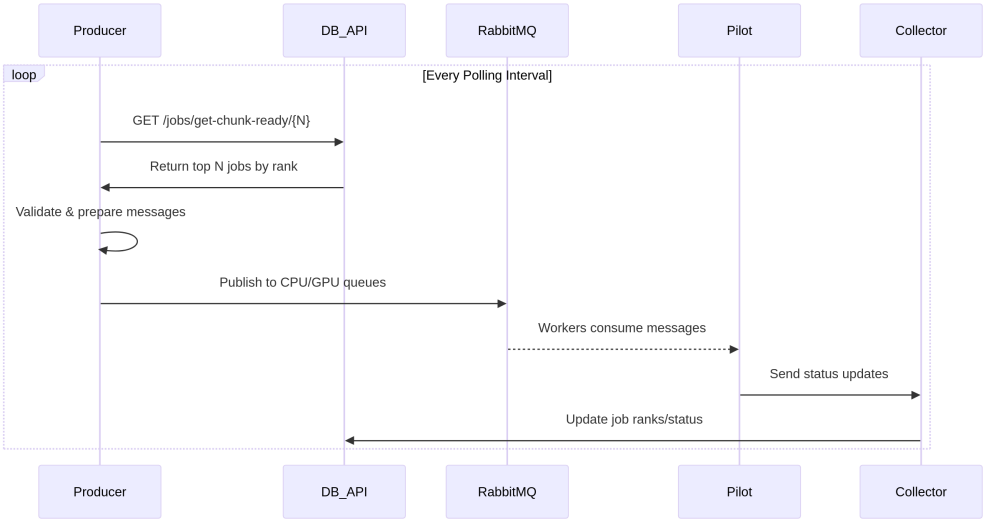


Figure 14: An example of the concurrent execution of several workflows.

Rank-Based Job Distribution Scheduler



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.

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

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

$\Gamma = \text{diag}(\gamma_1, ..., \gamma_N)$ (job-specific history weights)
$\mathbf{x}_i = [x_i^{(1)}, ..., x_i^{(N)}]^\top$ (job ages)
$\mathbf{y}_i = [y_i^{(1)}, ..., y_i^{(N)}]^\top$ (retry counts)