MESHCHERYAKOV LABORATORY of INFORMATION TECHNOLOGIES

SPD NICA

Заявка на соискание гранта ОМУС, стипендий им. М.Г. Мещерякова и Н.Н. Говоруна

Разработка системы управления рабочей нагрузкой
SPD Online Filter

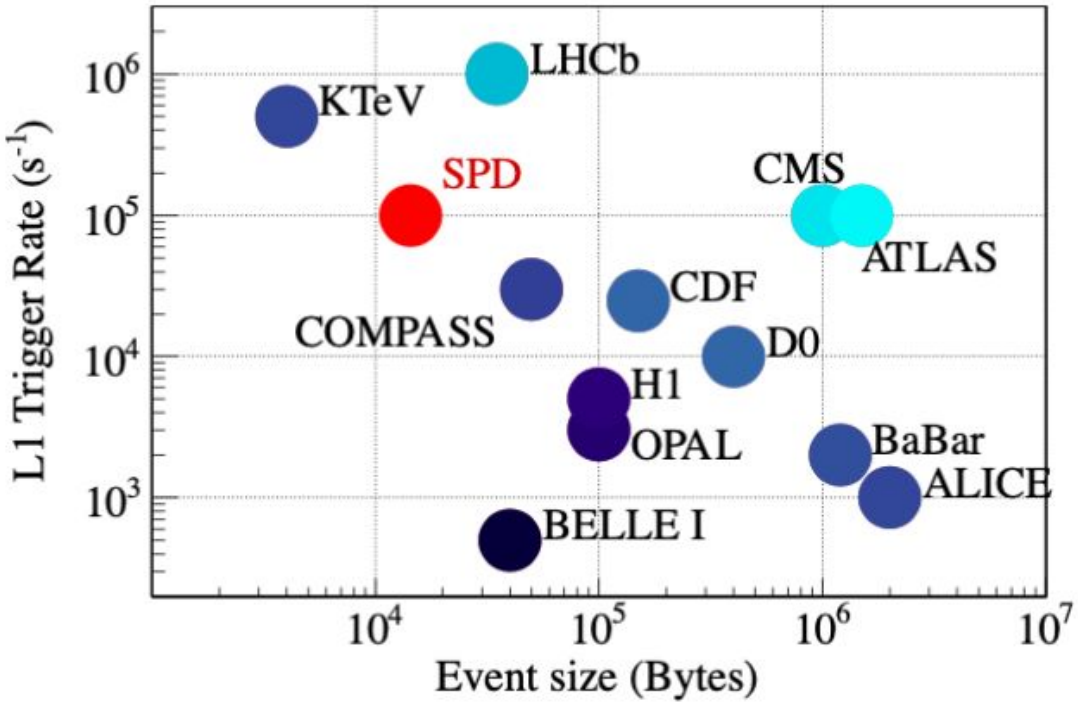
Гребень Никита
стажер-исследователь
НТО ВКиРИС

Непосредственный руководитель
Олейник Данила Анатольевич
старший научный сотрудник ЛИТ ОИЯИ

Актуальность работы

«SPD Online filter» будет представлять собой программно-аппаратный комплекс высокопропускной обработки первичных данных эксперимента SPD коллайдере NICA, с целью уменьшения их объема для последующей обработки и долговременного хранения. Аппаратная часть будет состоять из совокупности многоядерных вычислительных узлов, высокопроизводительных систем хранения данных и ряда управляющих серверов; программная часть будет состоять не только из прикладного программного обеспечения, но и из комплекса промежуточного ПО — «SPD Online filter» «Visor», роль которого заключается в реализации многоступенчатой обработки данных.

Система управления нагрузкой, является ключевым компонентом комплекса промежуточного ПО

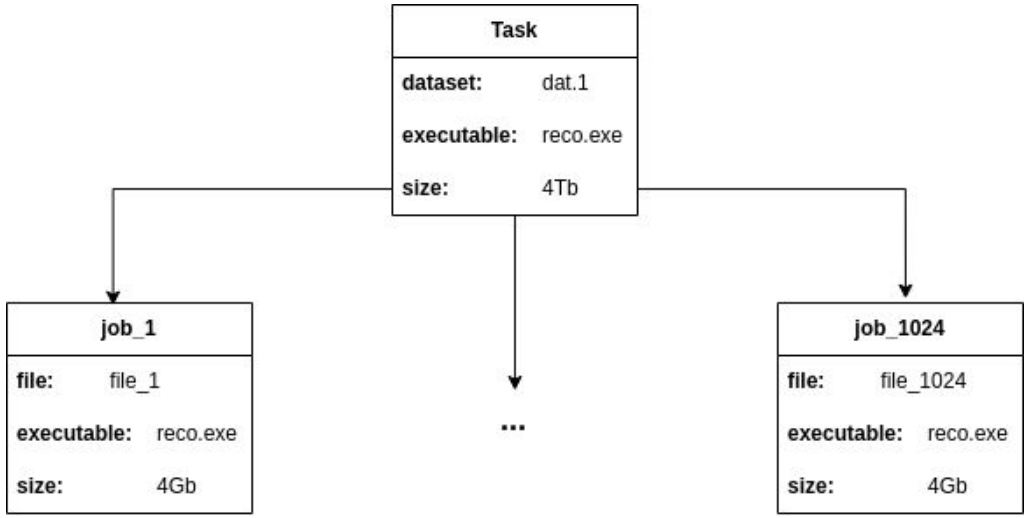


- ~ 20 ГБ/с (или 200 ПБ/год) «сырых» данных
- безтриггерный DAQ (Data Acquisition System)

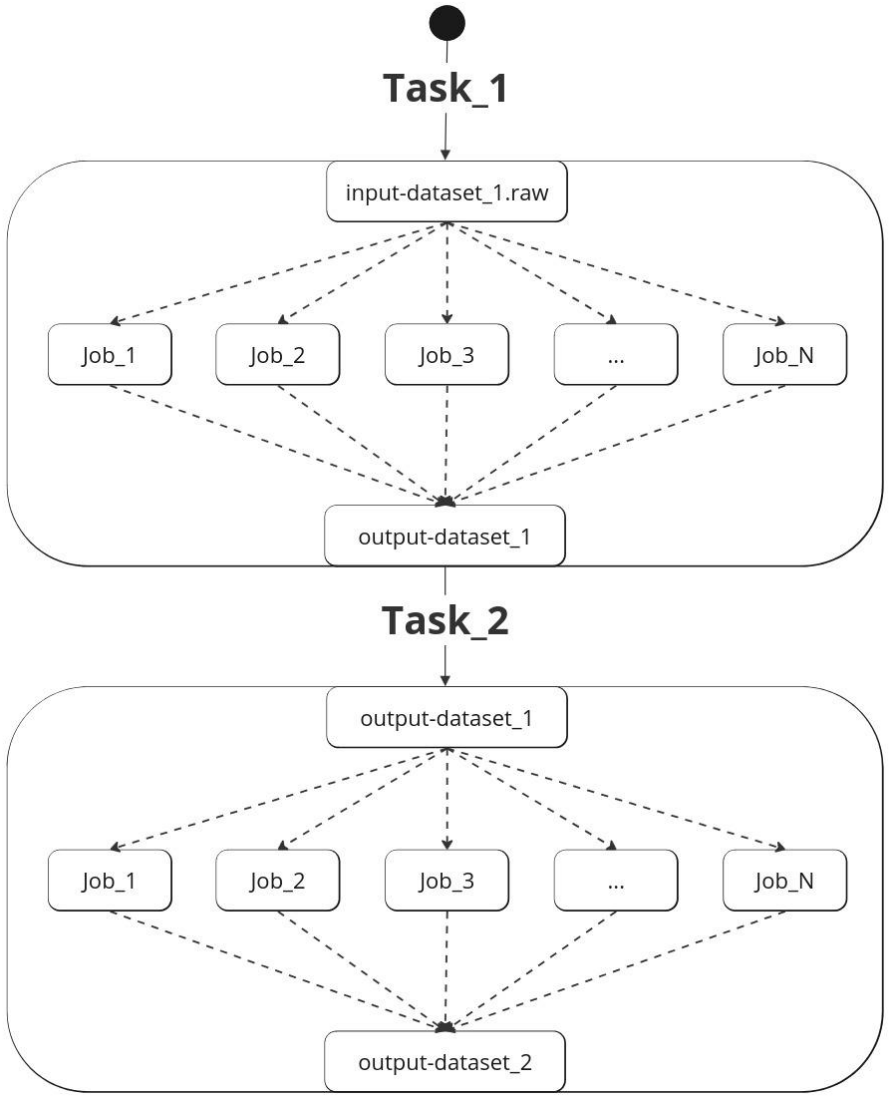
Высокопропускные вычисления

MESHCHERYAKOV LABORATORY of INFORMATION TECHNOLOGIES

- **HTC** определяется как тип вычислений, при котором одновременно выполняется множество простых и независимых друг от друга задач для выполнения задания обработки данных.
- Поскольку каждый элемент данных может обрабатываться одновременно, это может быть применено к данным, агрегированным системой сбора данных (DAQ).
- Обработка данных многоступенчатая:
  - Один этап обработки → **task**
  - Обработка блока данных (файла) → **job**



Task-job relationship



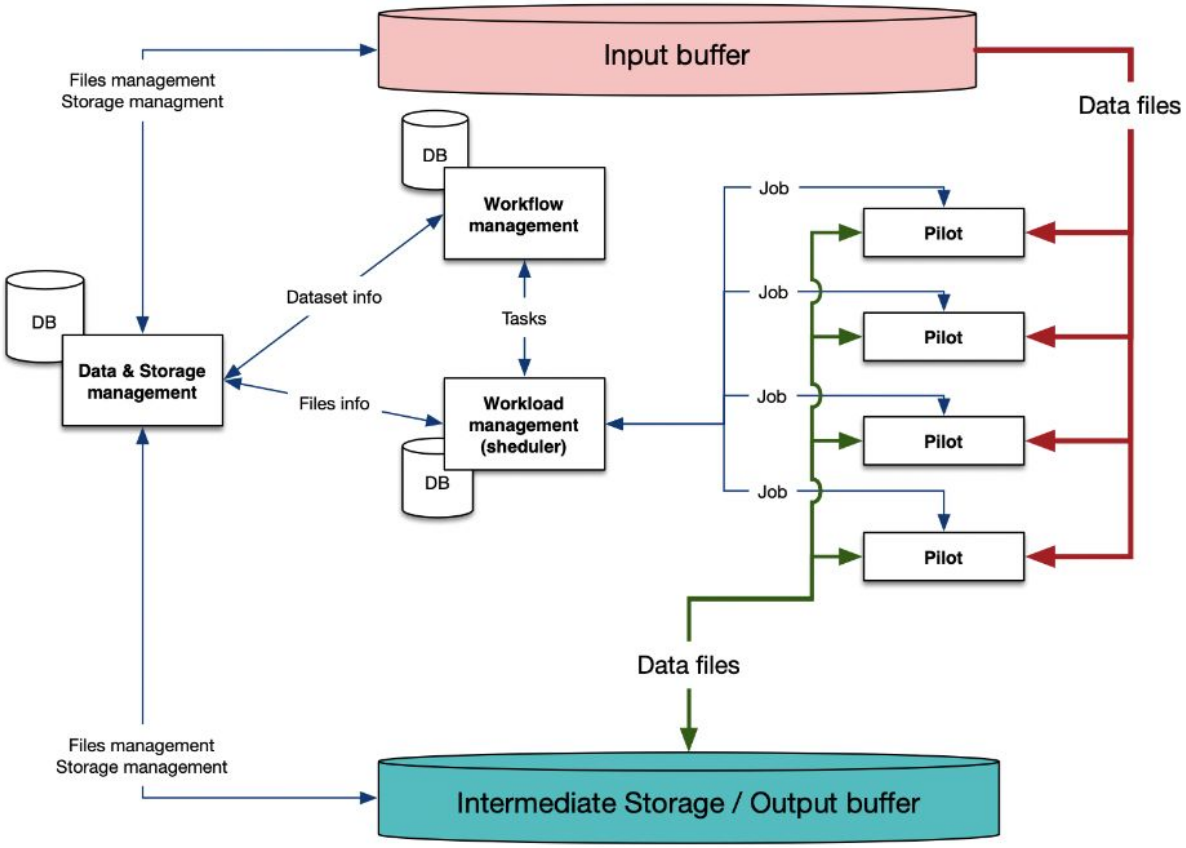
Data processing workflow example

Промежуточное программное обеспечение

MESHCHERYAKOV LABORATORY of INFORMATION TECHNOLOGIES

«SPD OnLine filter» – аппаратно-программный комплекс, обеспечивающий многоступенчатую высокопропускную обработку первичных данных эксперимента **SPD**.

- **Система управления данными**
  - Поддержка жизненного цикла данных (каталог данных, проверка согласованности, очистка, хранение);
- **Система управления процессами**
  - Определение и выполнение цепочки обработки, сгенерировав необходимое количество вычислительных заданий;
- **Система управления нагрузкой**
  - Генерация необходимого количество задач для выполнения задания;
  - Контроль выполнения задач с помощью пилотов, работающих на вычислительных узлах;
- Координатор: **Данила Олейник**

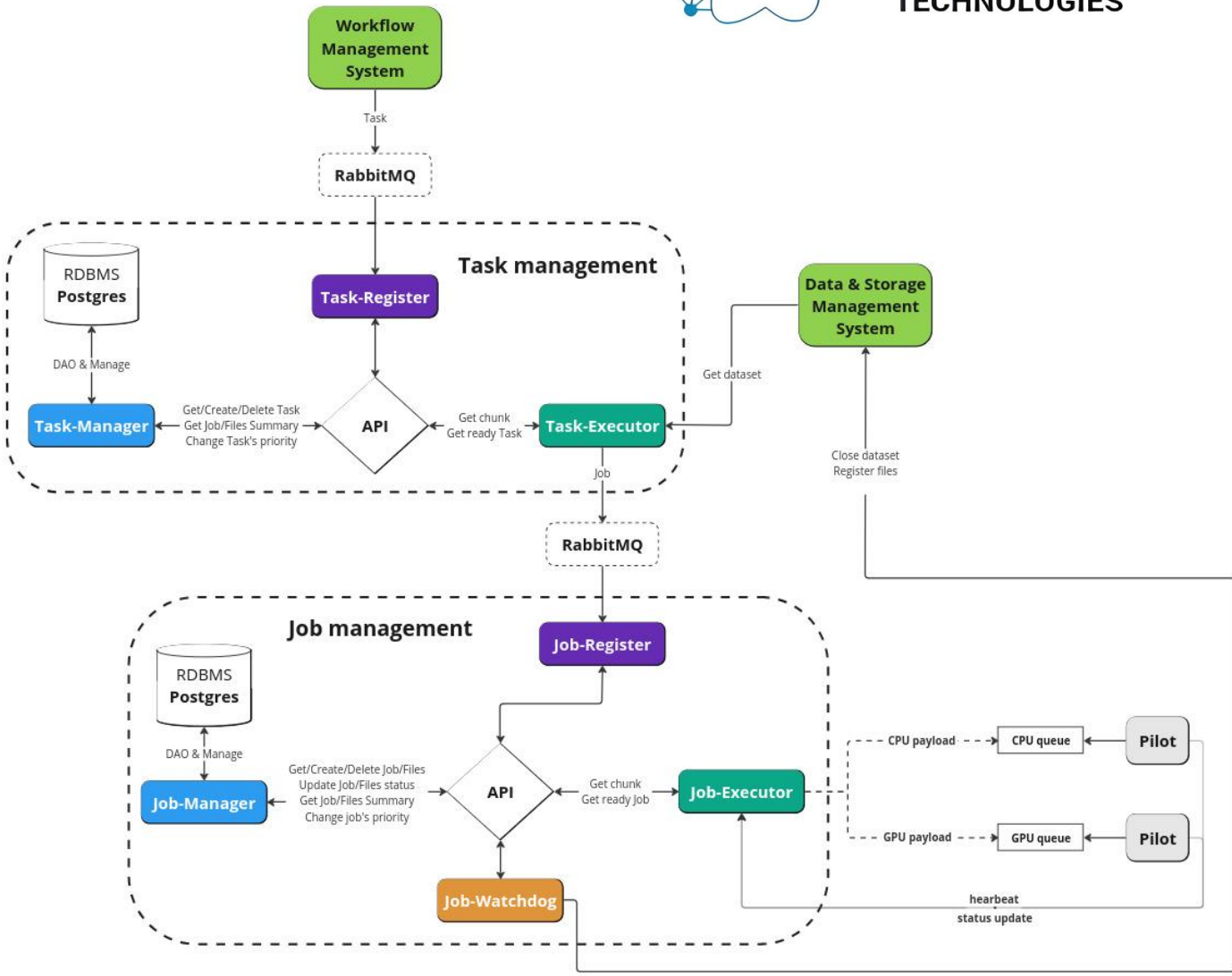


Architecture of SPD Online Filter

Требования к системе управления нагрузкой

MESHCHERYAKOV LABORATORY of INFORMATION TECHNOLOGIES

- **Регистрация заданий:** формализованное описание задания, включая параметры задач и требуемые метаданные;
- **Определение задач**: генерация необходимого количества задач для выполнения задания путем контролируемой загрузки доступных вычислительных ресурсов;
- **Управление исполнения задач**: контроль состояния задач, повторные попытки выполнения задач в случае сбоев, завершение выполнения задачи;
- **Контроль согласованности**: контроль согласованности данных в отношении заданий, файлов и задач;
- **Планировщик:** реализация алгоритма планирования для распределения заданий/задач;



SPD Workload Management System High Level Architecture

Pilot Agent

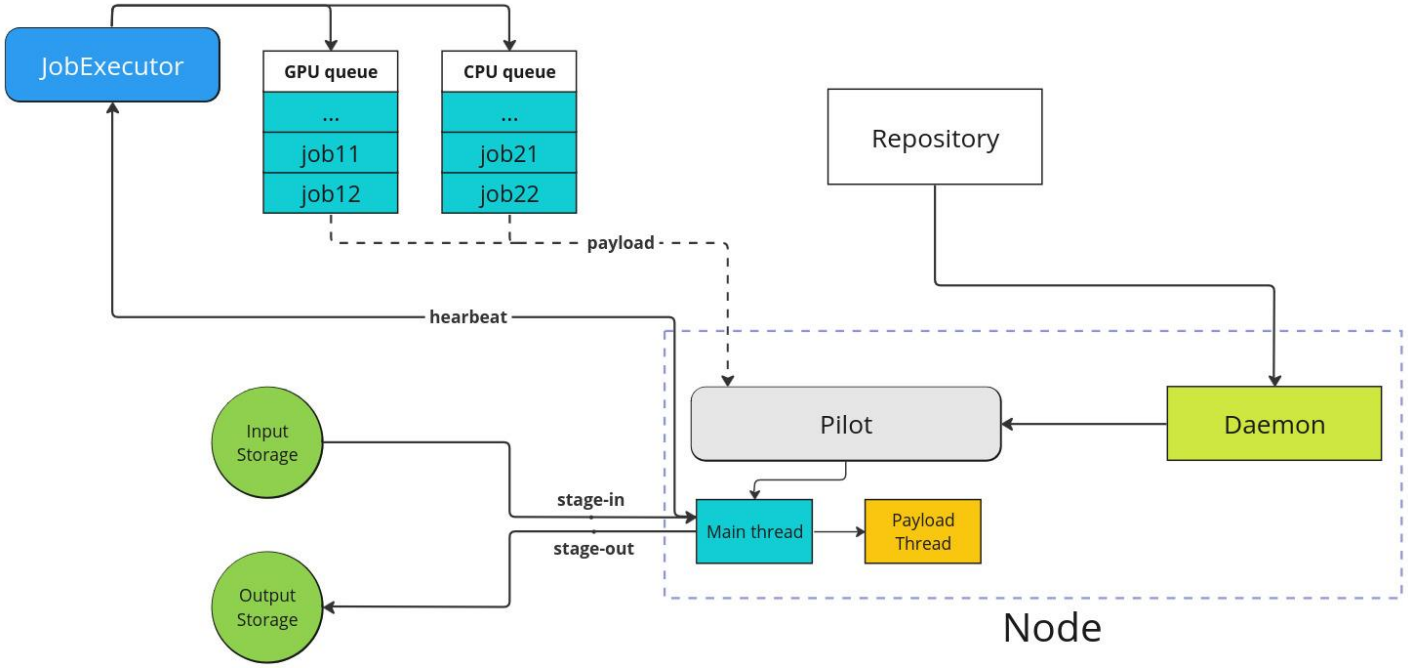
- Агентское приложение развертывается на вычислительном узле и состоит из следующих двух компонентов: UNIX демона и самого пилота.
- Задача демона UNIX - запустить следующий пилот, загрузив актуальную версию из хранилища.
- Сам Pilot представляет собой многопоточное приложение, отвечающее за
  - Получение и валидацию задач из брокера сообщений;
  - Загрузка входных файлов и выгрузка файлов с результатами в выходное хранилище;
  - Запуск подпроцесса для выполнения полезной нагрузки (декодирование формата DAQ, алгоритм распознавания треков и т. д.)
  - Информирование вышестоящей системы о текущем статусе полезной нагрузки и самого пилота

Два типа узлов:

- Multi-CPU
- Multi-CPU + GPU

Два канала связи:

- HTTP (aiohttp)
- AMQP (message broker - RabbitMQ)



Текущий статус

MESHCHERYAKOV LABORATORY of INFORMATION TECHNOLOGIES

Проектирование:

- ✅ Разработан и реализован набор необходимых REST API методов (DB API сервис);
- ✅ Реализован механизм объявления модели данных в базе данных на основе ORM и скриптов миграции;
- ✅ Настройка инструментов CD (сборка и развертывание) в облачной инфраструктуре ЛИТ;
- ✅ Разработка сценариев межсервисного взаимодействия - определение контрактов API;
- ✅ Реализована модель взаимодействия с пилотом на базе описанной статусной модели задач.

Прототипирование:

- ✅ Большинство микросервисов реализовано;
- ✅ Подсистема управления задачами является наиболее продвинутой: реализовано и тестируется большинство взаимодействий;
- ✅ Переезд на производственный СУБД Postgres;
- ✅ Пилот обрабатывает все этапы выполнения задачи;
- ↻ Реализуется разбиение задания;
- ↻ Переезд на производственный сервер RabbitMQ.

Публикации

1. 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).
2. V.M. Abazov et.al [SPD Collaboration] Technical Design Report of the Spin Physics Detector at NICA // arXiv: 2404.08317 [hep-ex]

Выступления

1. 59th meeting of the PAC for Particle Physics – «Workload Management System for SPD Online Filter» – ОИЯИ – Дубна – Россия – 2024. – Постерная сессия
2. VII SPD Collaboration meeting – «Workload Management System for SPD Online filter» – Institute of Nuclear Physics – Almaty – Kazakhstan – 2024. – Устный доклад
3. JINR Association of Young Scientists and Specialists Conference "Alushta-2024" – «Система управления нагрузкой специализированной вычислительной системы SPD Online Filter» – JINR – Alushta – Russia – 2024. – Устный доклад
4. The 28th International Scientific Conference of Young Scientists and Specialists (AYSS-2024) – «Workload Management System Development for SPD Online Filter» – JINR – Dubna – Russia – 2024. – Устный доклад
5. VIII SPD Collaboration Meeting – «SOF Middleware development status» – JINR – Dubna – Russia – 2024. Устный доклад

План работы на 2025 год

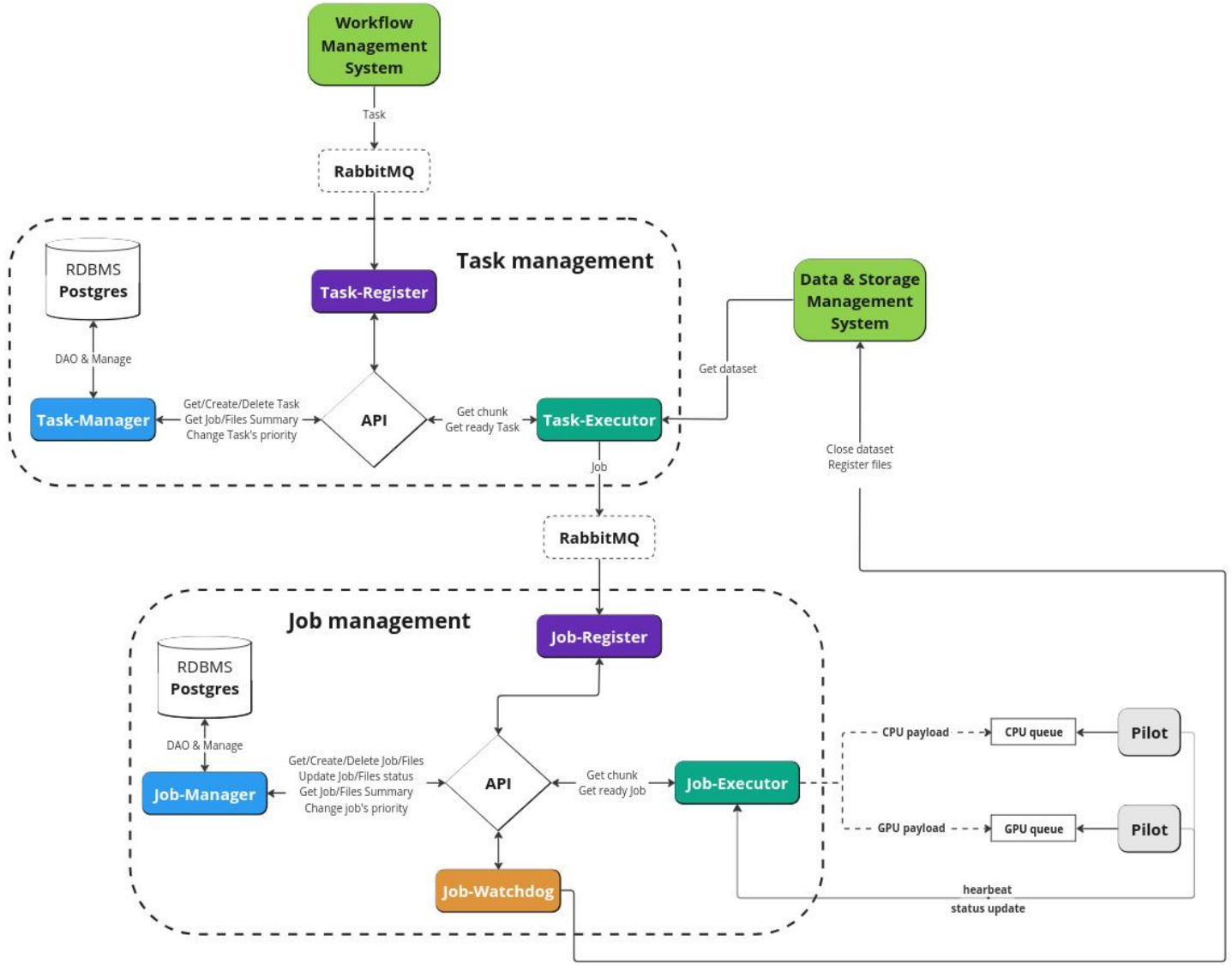
- **Обработка заданий и рабочих процессов**
  - Выполнение всего рабочего процесса, установленного на уровне системы управления процессами;
  - Весь рабочий процесс - цепочка зависимых заданий.
- **Интеграция с прикладным программным обеспечением**
  - Требуется прикладное программное обеспечение (в процессе разработки) и смоделированные данные;
  - Нефункциональные требования к прикладным программам.

Спасибо за внимание!

Backup slides

Architecture and functionality of Workload Management System

- **task-manager** – implements both external and internal REST APIs. Responsible for registering tasks for processing, cancelling tasks, reporting on current output files and tasks in the system.
- **task-executor** – responsible for forming jobs in the system by dataset contents.
- **job-manager** – accountable for storing jobs and files metadata, as well as providing a REST API for the executed jobs.
- **job-executor** – responsible for distribution of jobs to pilot applications, updating the status of jobs
- **pilot** – responsible for running jobs on compute nodes, organizing their execution, and communicating various information about their progress and status.



Workload management system requirements - reminder

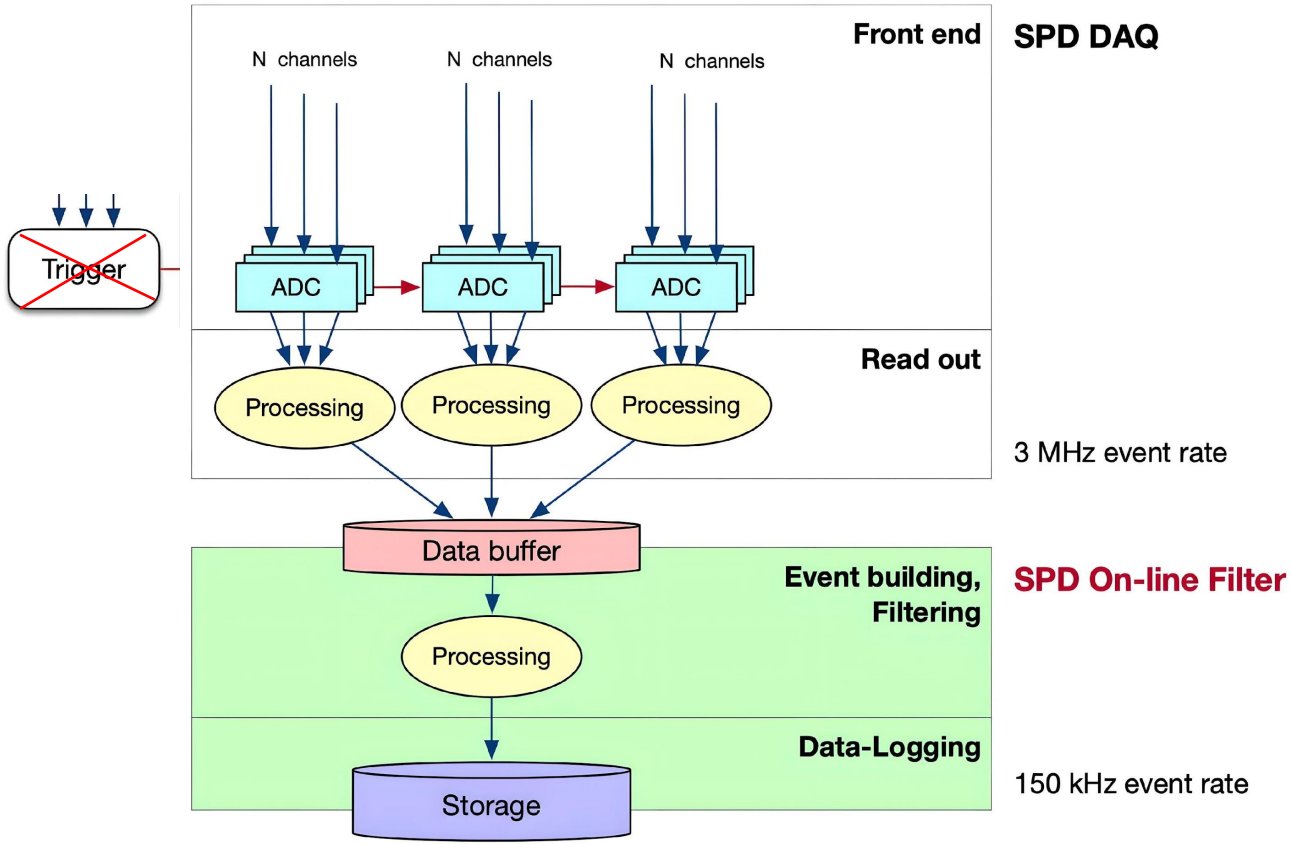
The key requirement - systems must meet the **high-throughput paradigm**.

- **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 a scheduling principle for task/job distribution;

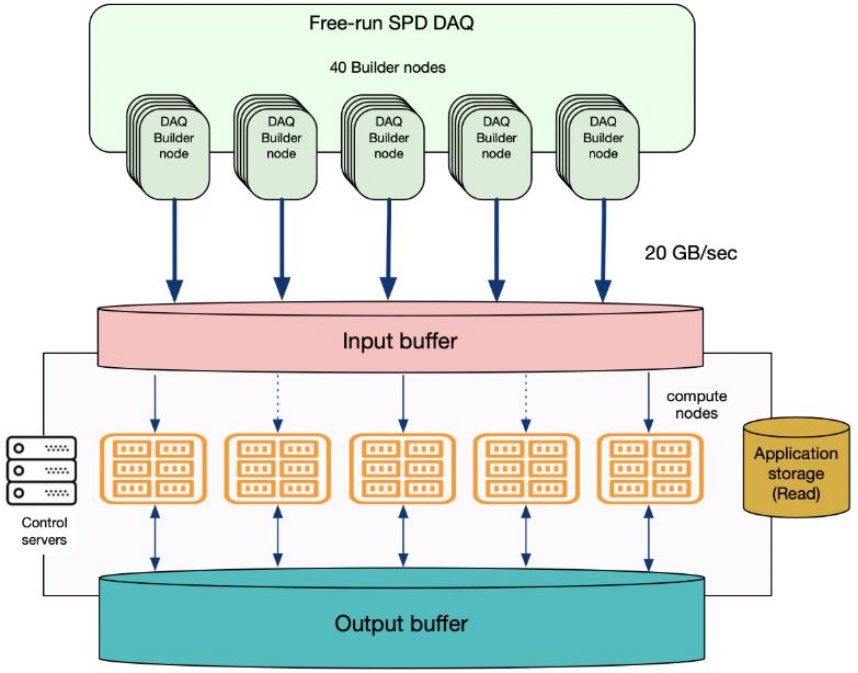
Forming jobs based on dataset contents, one file per one job

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.



«triggerless» DAQ



- 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.

Database design

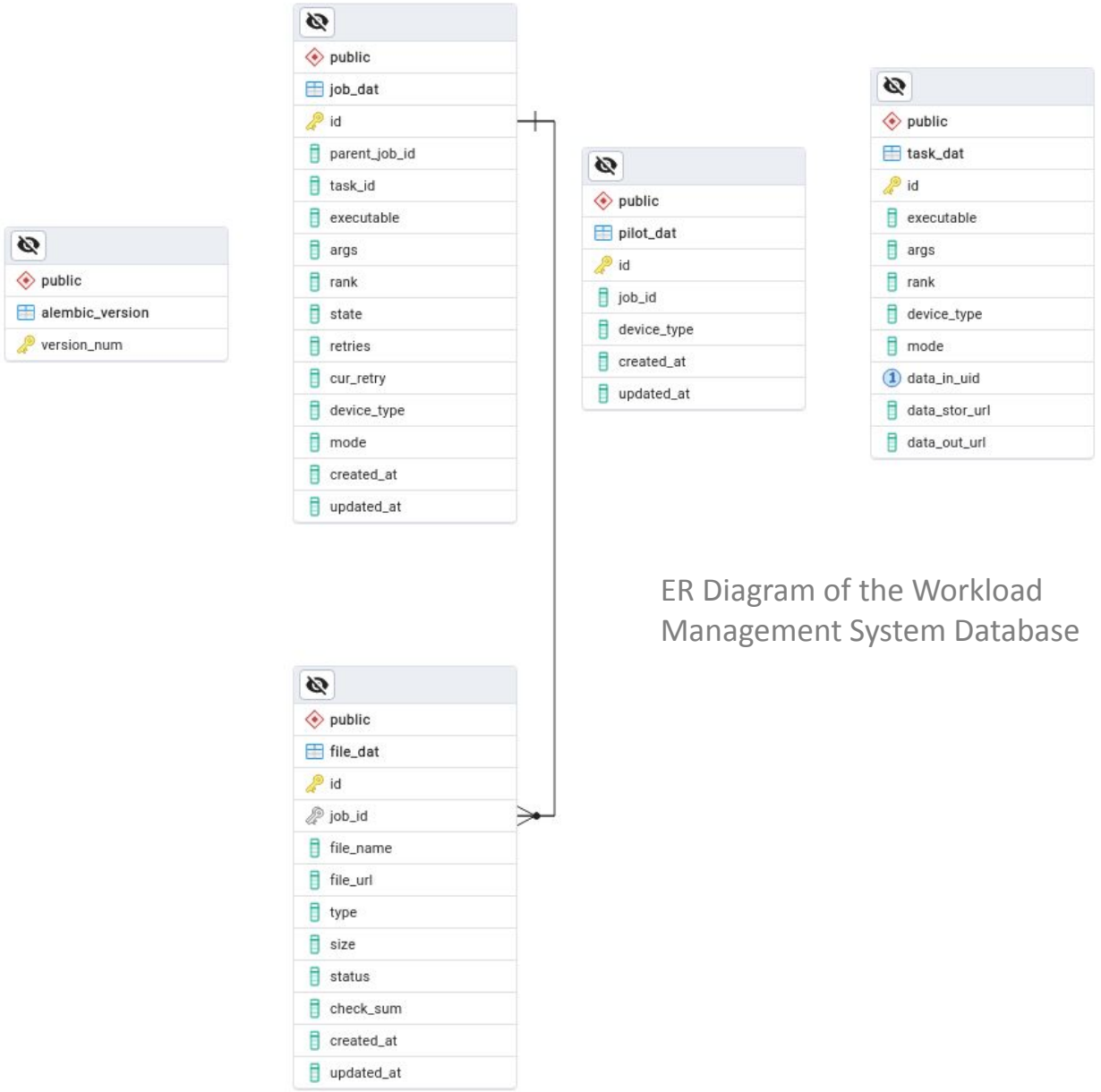
RDBMS - PostgreSQL 16

Tables:

- **alembic_version** – managing and tracking database schema changes
- **file_dat** – a directory specifying the output files and logs generated on the pilot
- **job_dat** – jobs currently being processed in the system
- **task_dat** – current tasks in the system

Extra mechanisms:

- **Indexes** – on filter fields for optimization of operations (B-tree);
- **Procedures** – task and job generation for test purposes;
- **Triggers** – rank update logic;
- **Decomposition** – single database per microservice (Postgres in Docker initially)



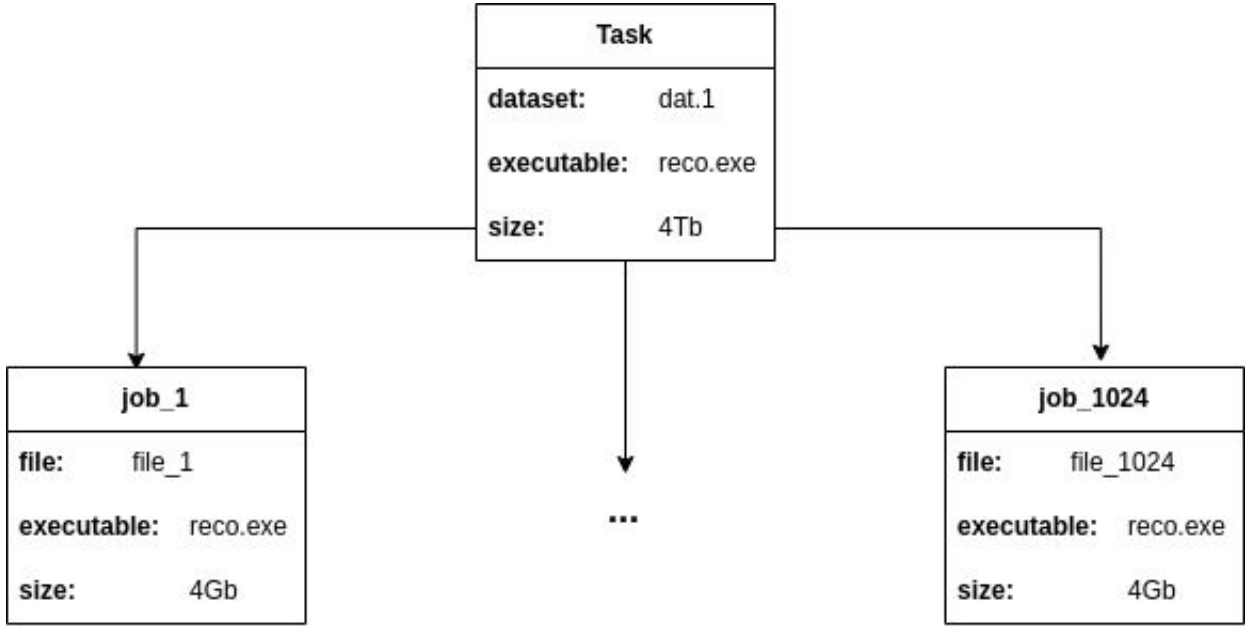
ER Diagram of the Workload Management System Database

Tech stack

Common ➢ Python 3.12 ➢ docker compose - running multi-container applications	**Frameworks** ➢ aio-pika (RabbitMQ + asyncio) - asynchronous API with RabbitMQ ➢ FastAPI + uvicorn
DB ➢ PostgreSQL - RDBMS ➢ Alembic (Migration) ➢ SQLAlchemy 2.0 ➢ asyncpg - Postgres DBAPI	**Extra** ➢ aiohttp - asynchronous HTTP client/server framework ➢ Pydantic - validate and serialize data schemes ➢ pytest-asyncio - test purposes

Task and job definition

- A **task** is a workload unit responsible for processing a block of homogeneous data - **dataset**.
- A processing request is a set of input data, which may consist of multiple files, and a handler.
- The criterion for the completion of the task is the processing of the entire block of data.
- The **Workflow Management System** is responsible for defining and executing workflows, as well as defining a processing request, which is a **task**.



Task-job relationship

- A **job** (payload) is a unit of work that processes a unit of data (**file**).
- The unit responsible for processing a single **file** in terms of workload is called a **job**.
- The **Workload Management System** is responsible for generating **jobs**, sending them to compute nodes, and executing them.

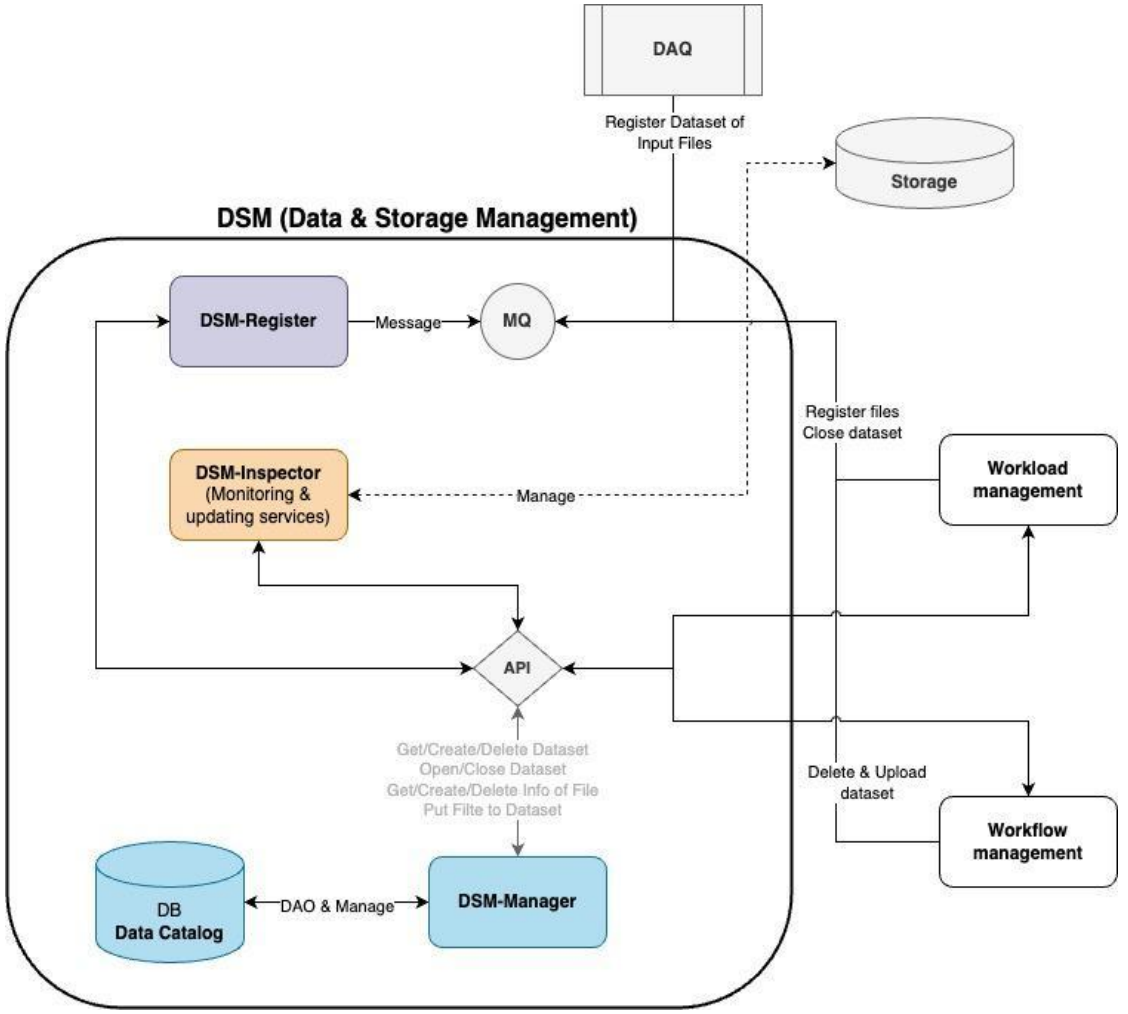
Interaction with Data Management System

Routing Key	Msg	Algo
dataset.close	Dataset info • Dataset UID • File check list (file names)	Request the registered files in the dataset. If they match the checklist, set the status to **CLOSED**. Otherwise, return the messages back to the queue for deferred execution.
dataset.upload	Dataset UID	Marking dataset for uploading (**TO_UPLOAD**)
dataset.delete	Dataset UID	Marking dataset for deletion (**TO_DELETE**)

Signature and algorithm of message receiving gateways for the **dsm-register** service

Within a **Workload Management System**, there are several scenarios for interacting with the data management system:

- ➢ Obtain information about dataset contents for forming jobs from **DSM-Manager (Data Catalog REST API)**
- ➢ Register files in datasets after executing payload on compute node – **DSM-Register (Data Registration)**
- ➢ Close dataset after cancellation or sufficient number of successfully processed files – **DSM-Register***

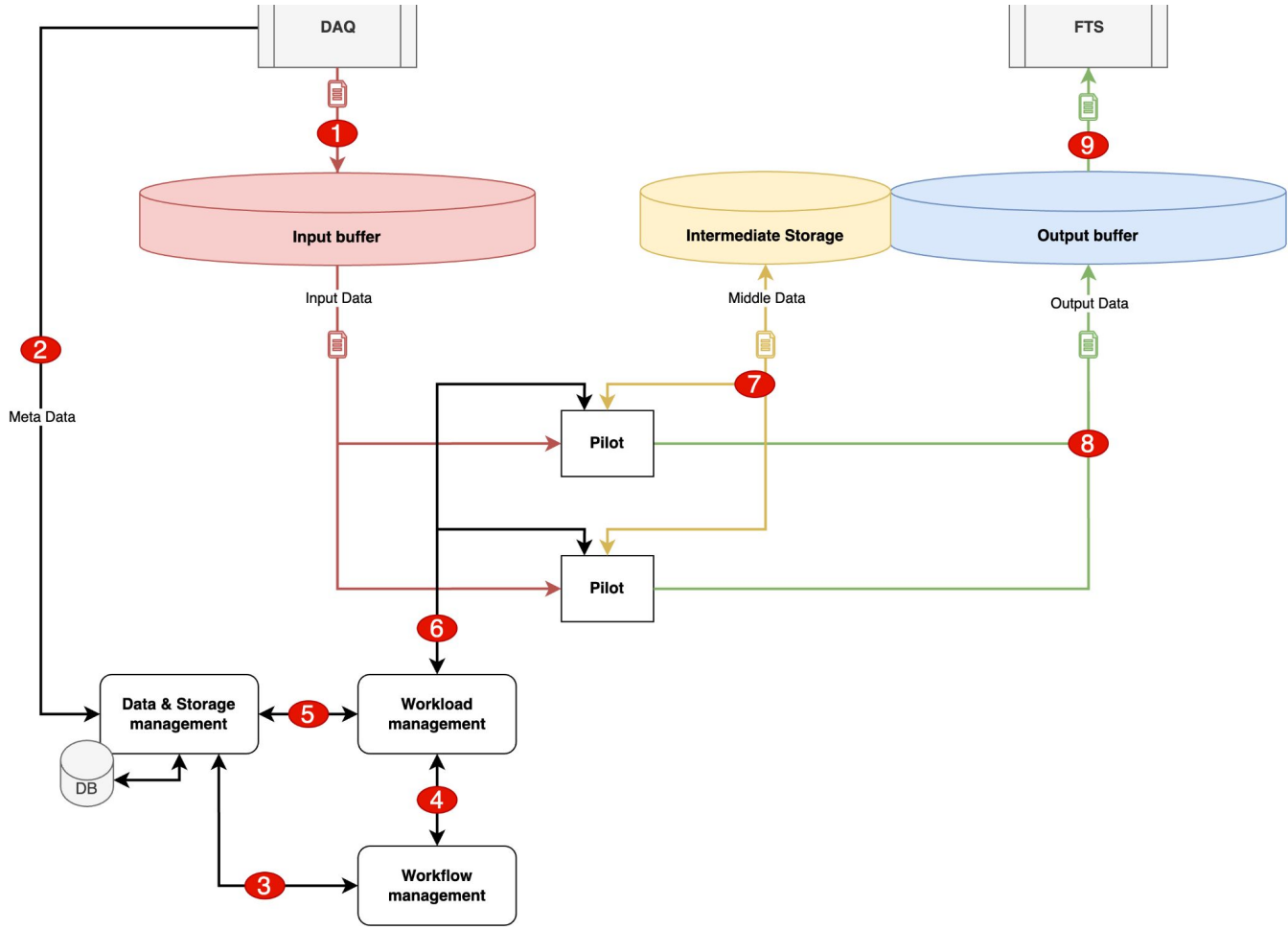


Architecture of Data Management

Dataflow and data processing concept

Main data streams:

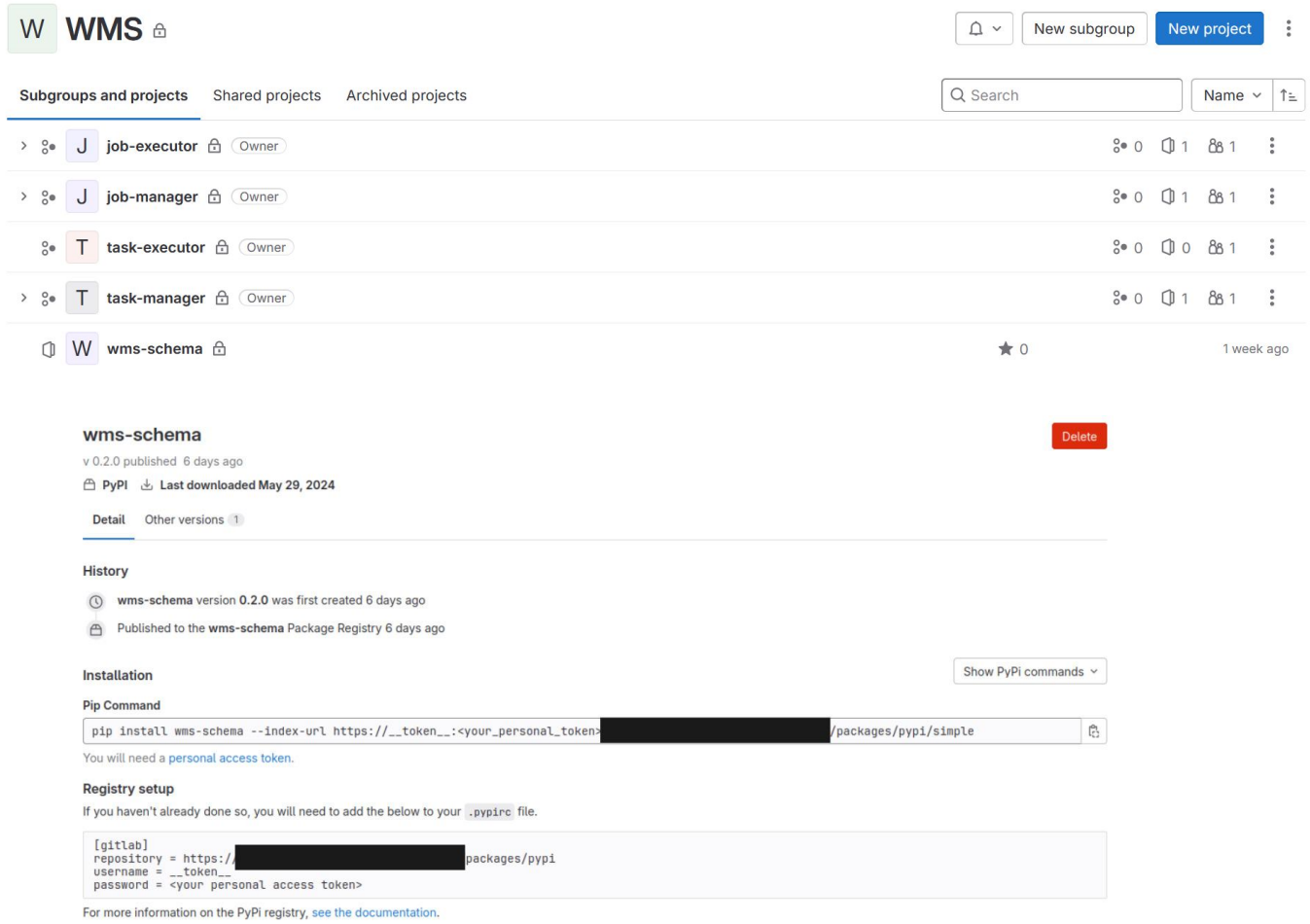
- SPD DAQs, after dividing sensor signals into time blocks, send data to the SPD Online Filter input buffer as files of a consistent size.
- The workflow management system creates and deletes intermediate and final data sets
- The **workload management system** “populates” the data sets with information about the resulting files
- At each stage of data processing, pilots will read and write files to storage and create secondary data



Modularization: deploying and using own packages

Following tools are used:

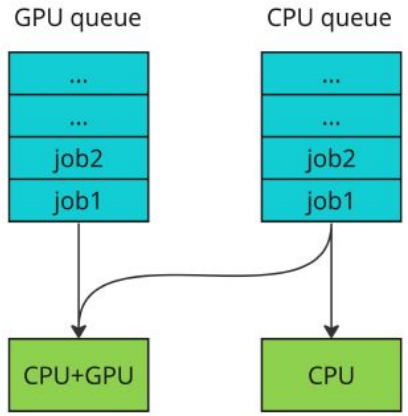
- Poetry
  - Particularly good at handling complex dependency trees and ensuring that the different modules can integrate with each other without version conflicts
- Python packages
  - Separate GitLab repositories for each package
  - Poetry for packaging and dependency management
- Gitlab
  - *Access Tokens* used as kind of credentials for scripts and other tools
  - CI/CD for automate testing and building

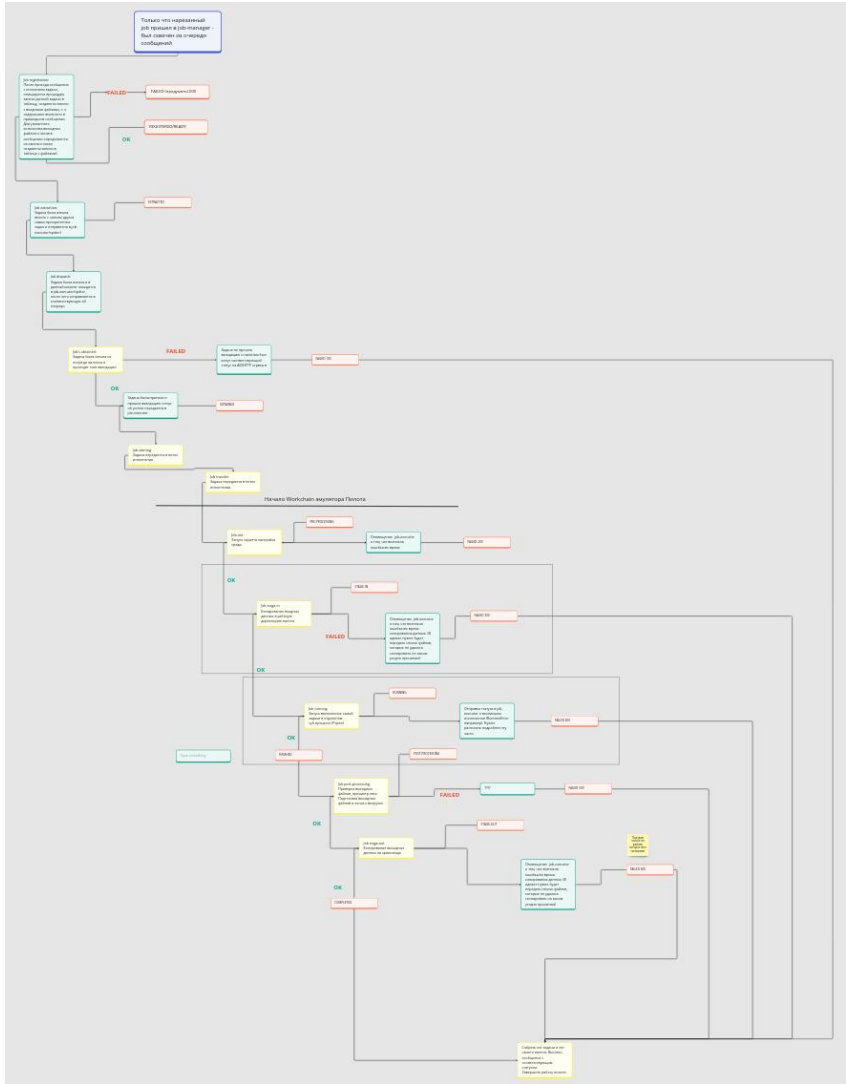


wms-schema is a package that contains a scheme for task and job data that is used in almost every other service

Interaction with the Pilot Agent

- Pilot has a series of prepossessing stages before running a job itself:
  a. start logging
  b. read configuration
  c. getting a job from message queue
  d. validation
- After those steps the Pilot launches another thread where it does
  a. environment setup script
  b. copying files locally from the input storage
  c. starts execution of a job itself in a separate sub-process
  d. analysis of the result of a job
  e. copying output data and logs to storage
  f. sends regular messages to **WMS**
  g. cleaning up the local environment
- Pilot sends status-update message at any point of internal changes
- **WMS** may terminate the job if the corresponding task is cancelled or if an error occurs.

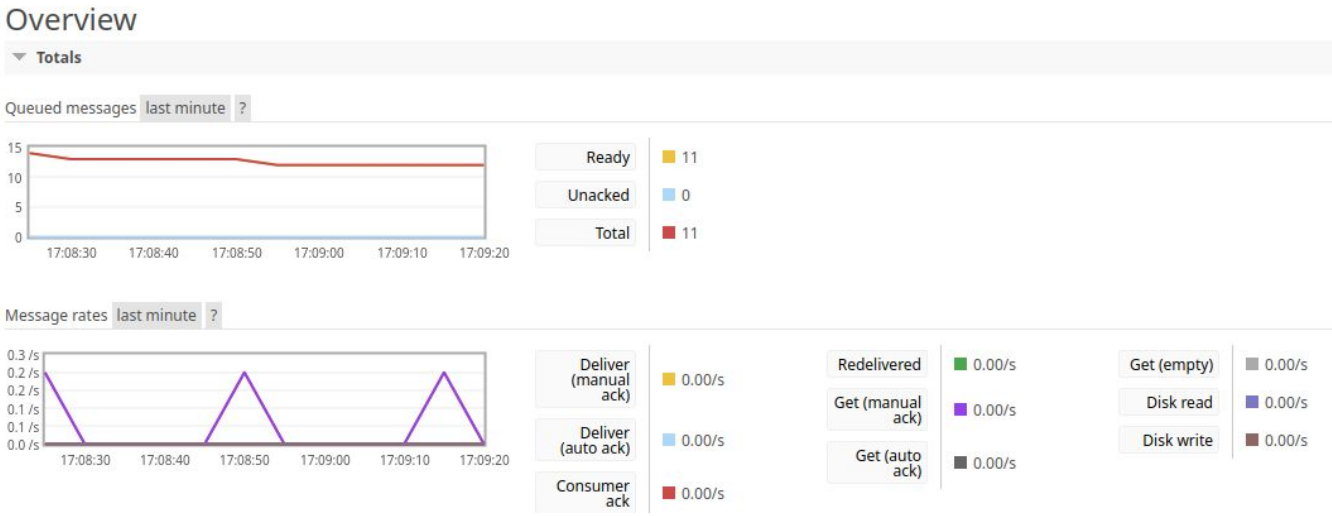




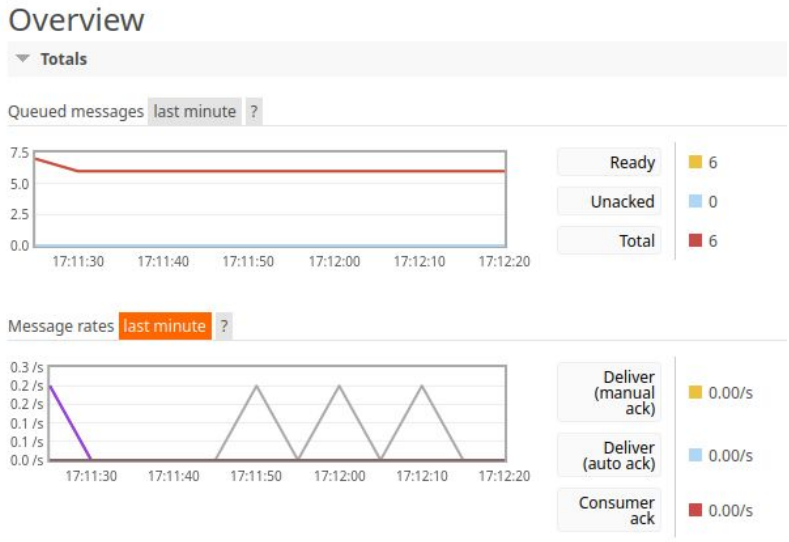
Система управления нагрузкой - Пилот

MESHCHERYAKOV LABORATORY of INFORMATION TECHNOLOGIES

- ✓ Описана подробная модель состояния задач;
- ✓ Введены коды ошибок;
- ✓ Pilot прошел все этапы выполнения задачи;
- ✓ Демон реализован и запущен;
- ✓ **Больше нет эмулятора пилота!**
- ✖ Требуется большой цикл тестирования и рефакторинга;
- ✖ Отладка выполнения всего задания (всех задач, связанных с заданием);



UNIX Daemon's running Pilot

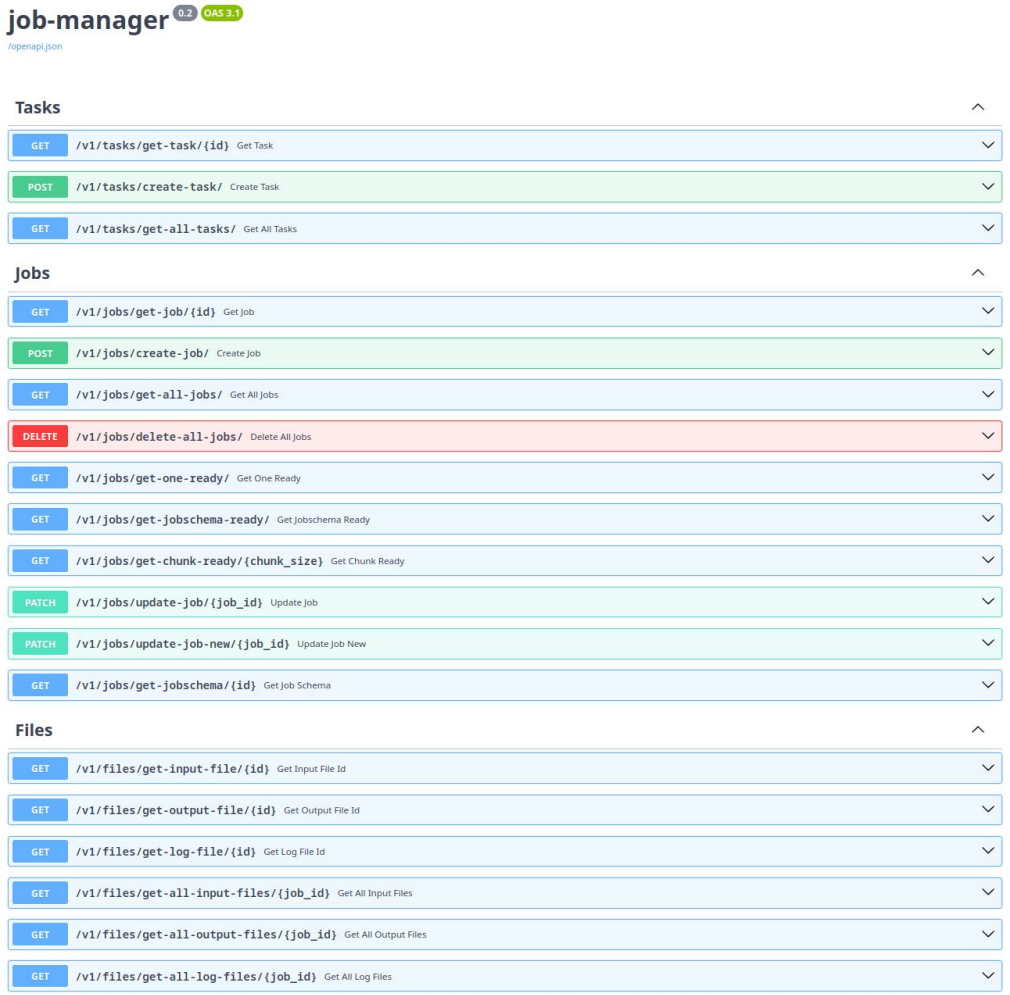


Aftermath of executing the entire queue

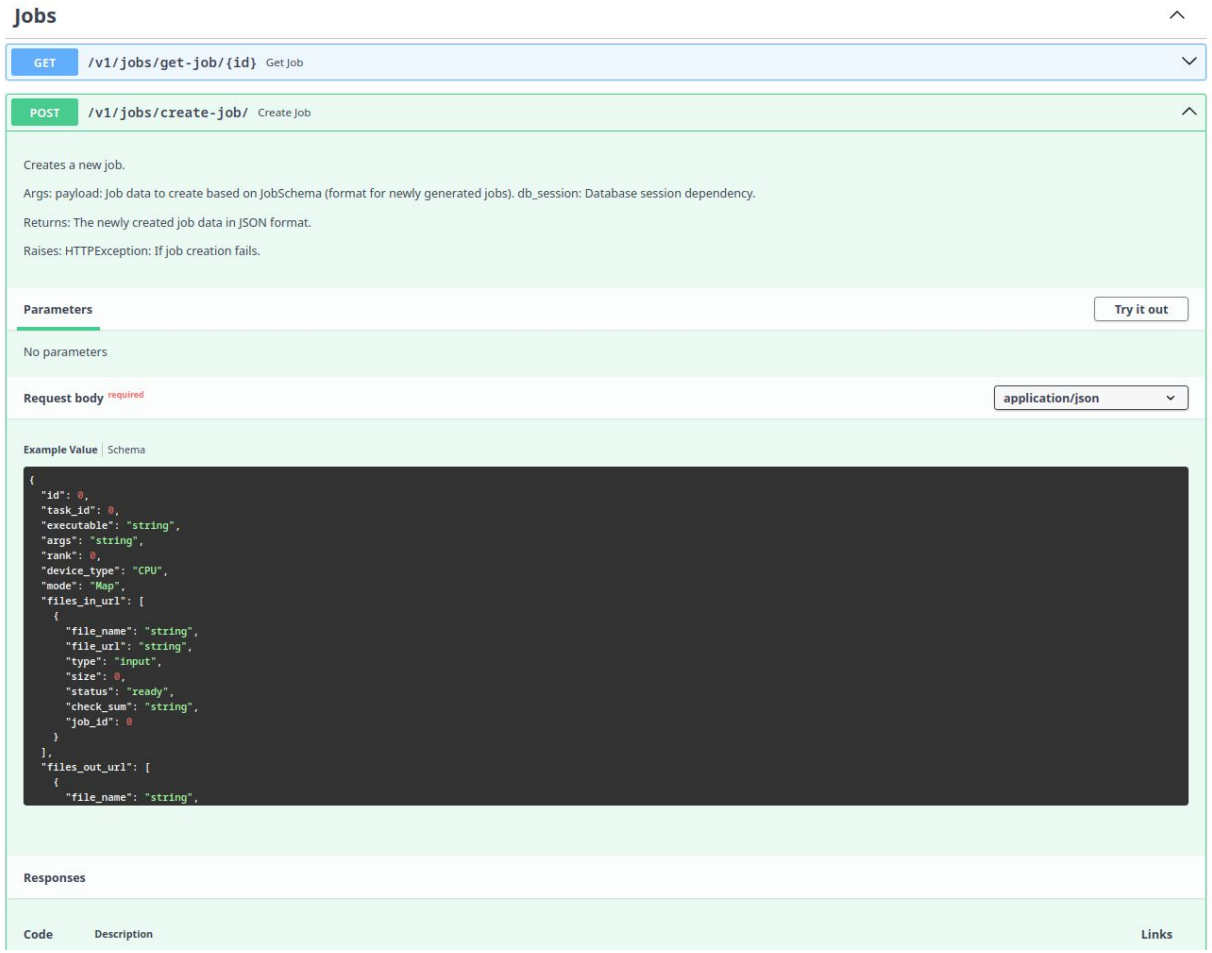
What it all looks like

Prototyping Job-Manager (API)

- The chosen framework for building the service is FastAPI + Uvicorn asynchronous framework
- A basic set of CRUD operations on data in the form of REST API is developed.
- API description autogeneration according to OpenAPI 3.0 specification is implemented (available in Swagger UI at <server address>/docs)



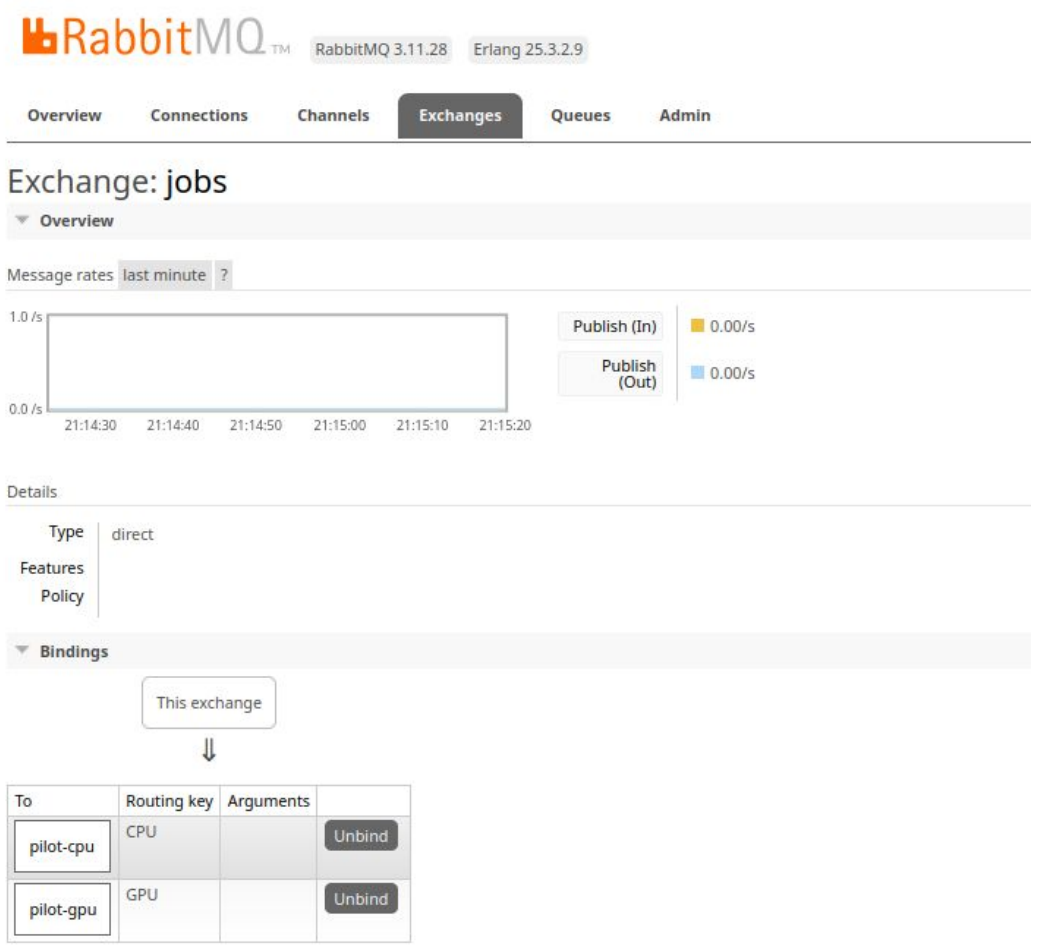
Swagger UI with job-manager service API description



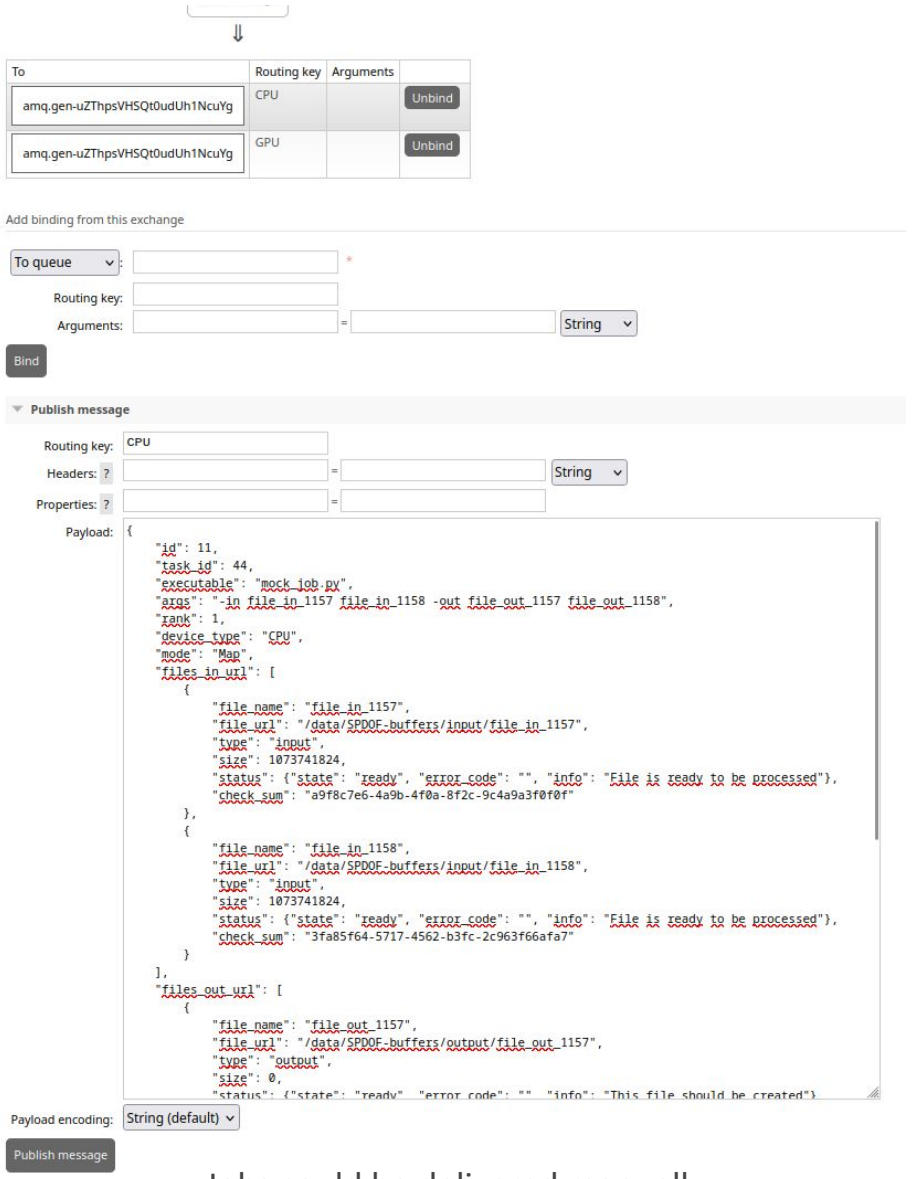
Example of a service call to post a new job

Prototyping Job-Executor - Pilot (RabbitMQ queues)

- RabbitMQ is selected as the message broker
- Queues are defined using the declarative notation of the aio-pika tool
- At the start of the application their unfolding is performed



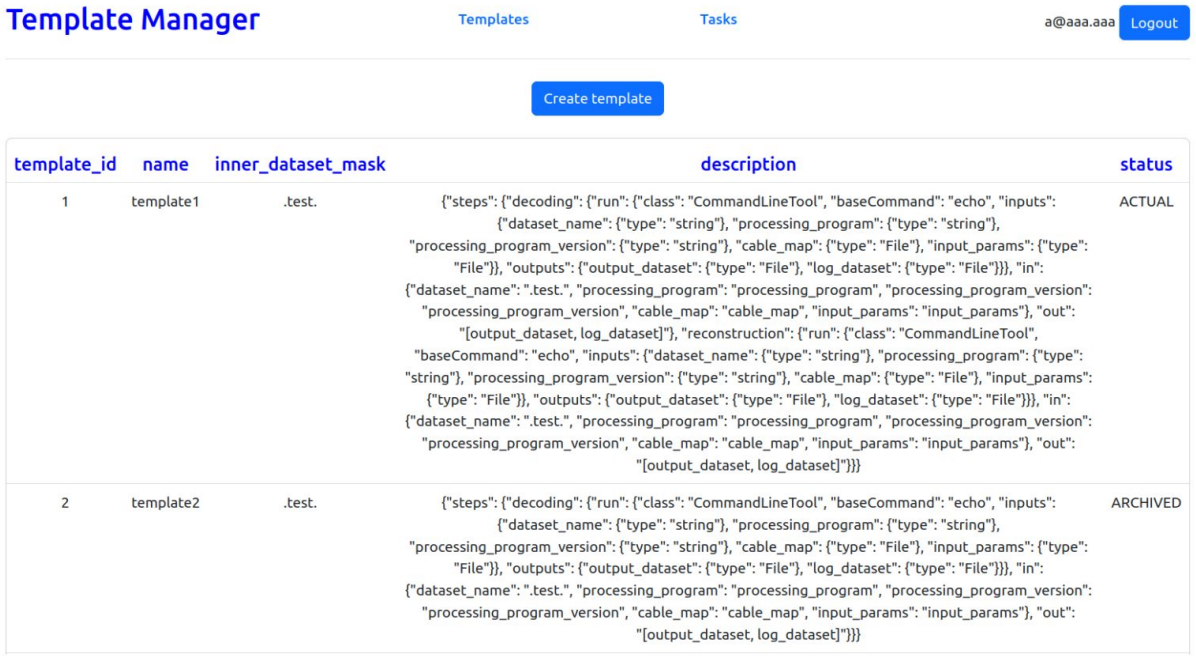
Configured RabbitMQ queues



Jobs could be delivered manually

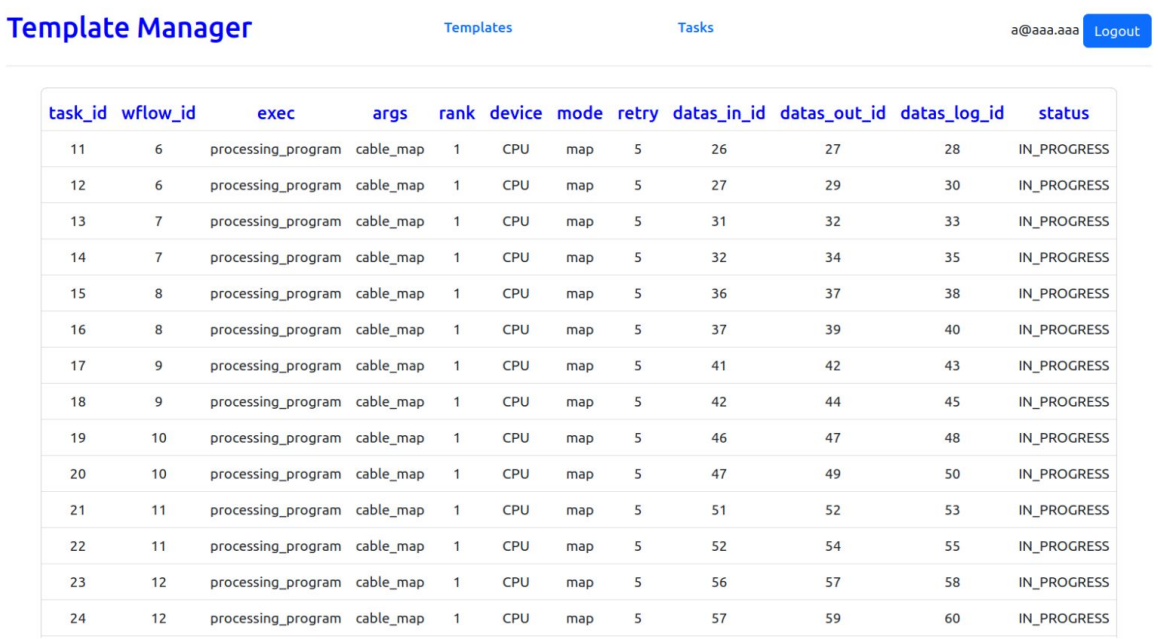
Examples of Templates and Tasks

- Registration and authorization
- Template and task output
- CWL template creation by user
- Preliminary validation and writing of CWL templates to the database



template_id	name	inner_dataset_mask	description	status
1	template1	.test.	{"steps": {"decoding": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}, "reconstruction": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}}}	ACTUAL
2	template2	.test.	{"steps": {"decoding": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}}}	ARCHIVED

Created template



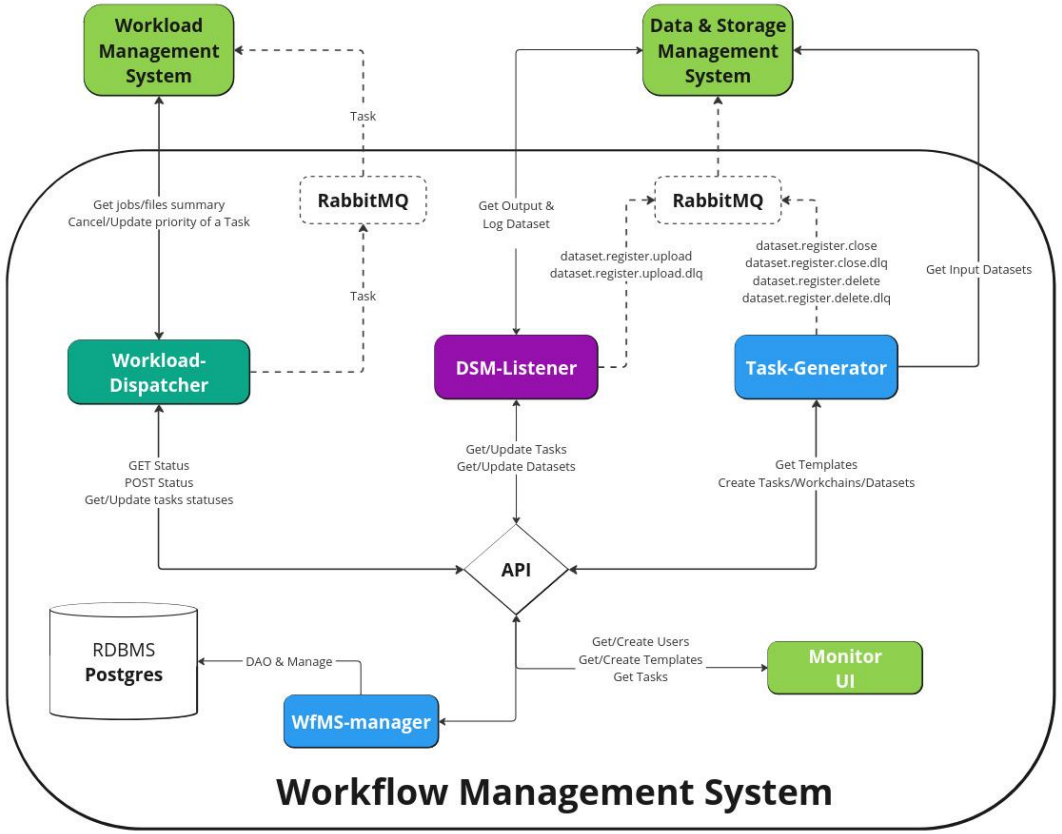
task_id	wflow_id	exec	args	rank	device	mode	retry	datas_in_id	datas_out_id	datas_log_id	status
11	6	processing_program	cable_map	1	CPU	map	5	26	27	28	IN_PROGRESS
12	6	processing_program	cable_map	1	CPU	map	5	27	29	30	IN_PROGRESS
13	7	processing_program	cable_map	1	CPU	map	5	31	32	33	IN_PROGRESS
14	7	processing_program	cable_map	1	CPU	map	5	32	34	35	IN_PROGRESS
15	8	processing_program	cable_map	1	CPU	map	5	36	37	38	IN_PROGRESS
16	8	processing_program	cable_map	1	CPU	map	5	37	39	40	IN_PROGRESS
17	9	processing_program	cable_map	1	CPU	map	5	41	42	43	IN_PROGRESS
18	9	processing_program	cable_map	1	CPU	map	5	42	44	45	IN_PROGRESS
19	10	processing_program	cable_map	1	CPU	map	5	46	47	48	IN_PROGRESS
20	10	processing_program	cable_map	1	CPU	map	5	47	49	50	IN_PROGRESS
21	11	processing_program	cable_map	1	CPU	map	5	51	52	53	IN_PROGRESS
22	11	processing_program	cable_map	1	CPU	map	5	52	54	55	IN_PROGRESS
23	12	processing_program	cable_map	1	CPU	map	5	56	57	58	IN_PROGRESS
24	12	processing_program	cable_map	1	CPU	map	5	57	59	60	IN_PROGRESS

WfMS task description

Interaction with Workflow Management System

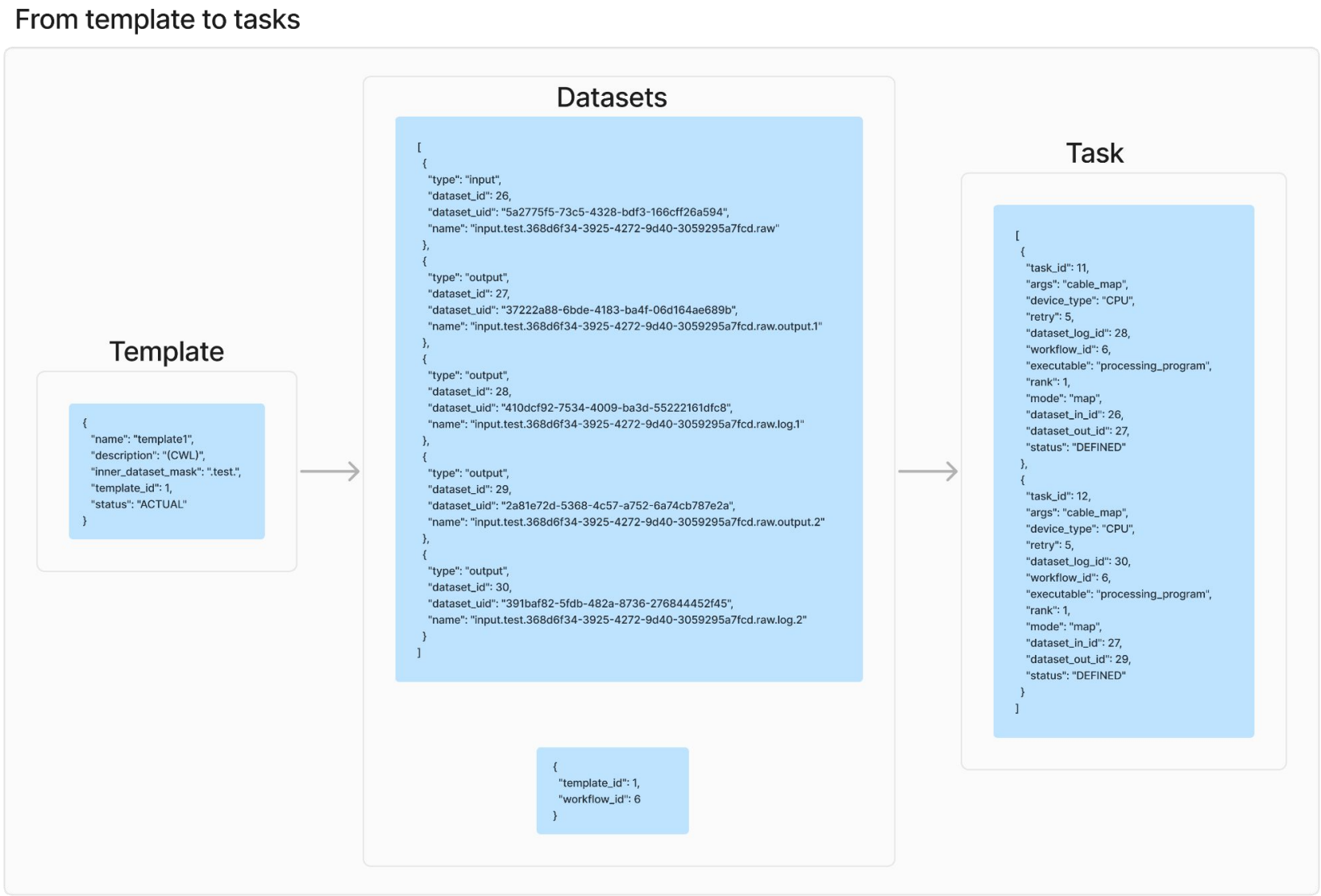
The following interaction scenarios have been identified with the Workflow Management System

- Registration of a task for processing: **WfMS** passes the task description into the message queue;
- Summary of current intermediate properties of jobs/files in the system: aggregated information about the status of each job/file for further decision making;
- Task cancellation: based on the decision made on the **WfMS** (*too many errors occurring*) on operator side;
- Change priority of a task: is used to accelerate the rate at which the corresponding dataset is being processed;



Task generation service

1. Getting registered datasets from **DMS** from RabbitMQ;
2. Matching datasets by name mask to the desired template;
3. Registration of input dataset in the system;
4. Creating a workflow from a template;
5. Creating output dataset and log dataset in the system;
6. Creating a task;



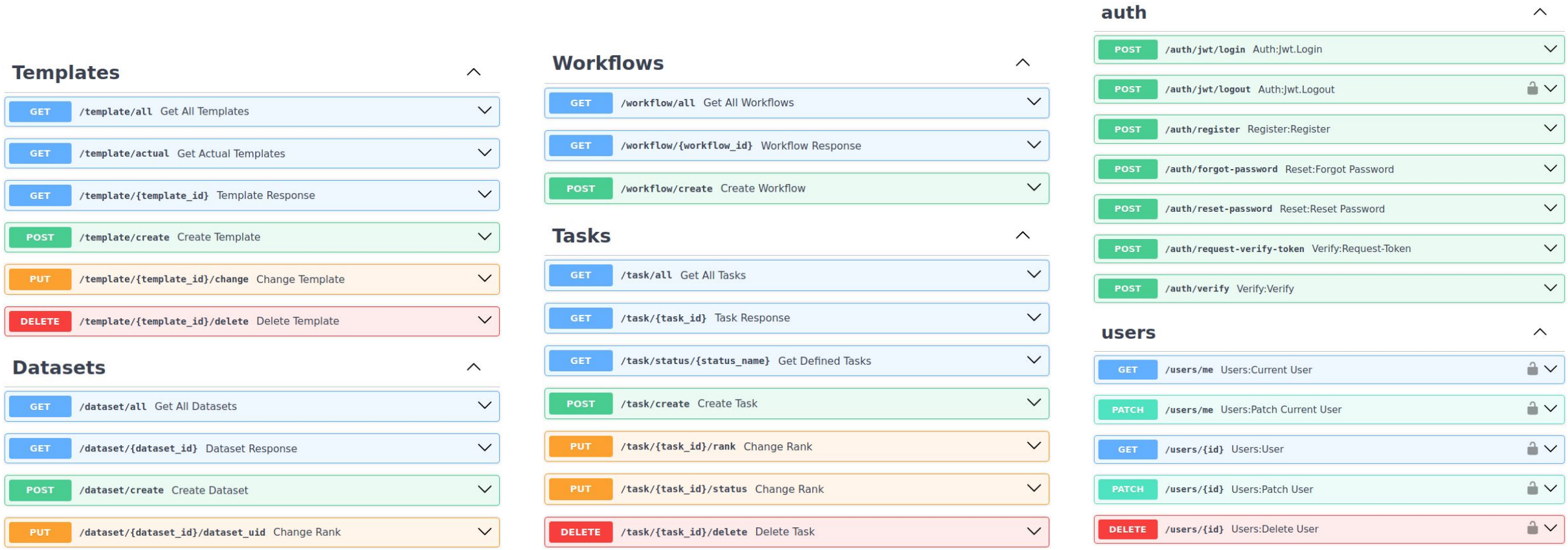
Task management service

- Iterate on tasks in “DEFINED” status;
- Querring DMS about the status of the input dataset (“CLOSED”);
- Creating output datasets and log datasets in DMS;
- Sending the task to RabbitMQ for further processing in WMS;
- Change task status to "IN_PROGRESS".

Sending tasks for processing

Data access service

- Data access service is implemented and provides all necessary endpoints at this stage;
- Test coverage is required;

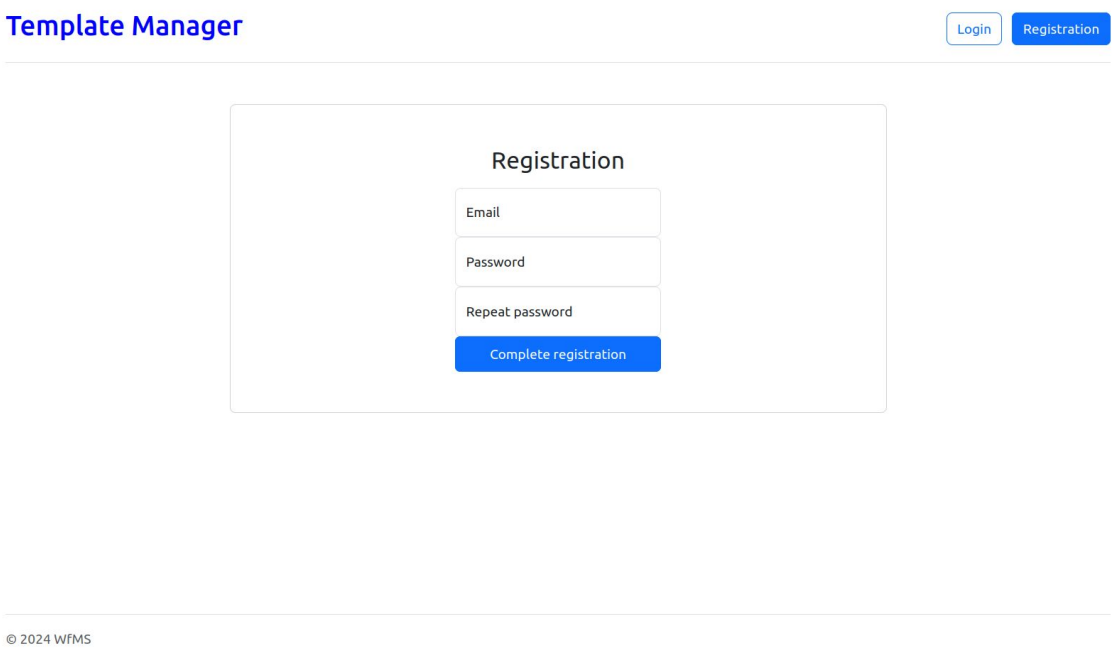


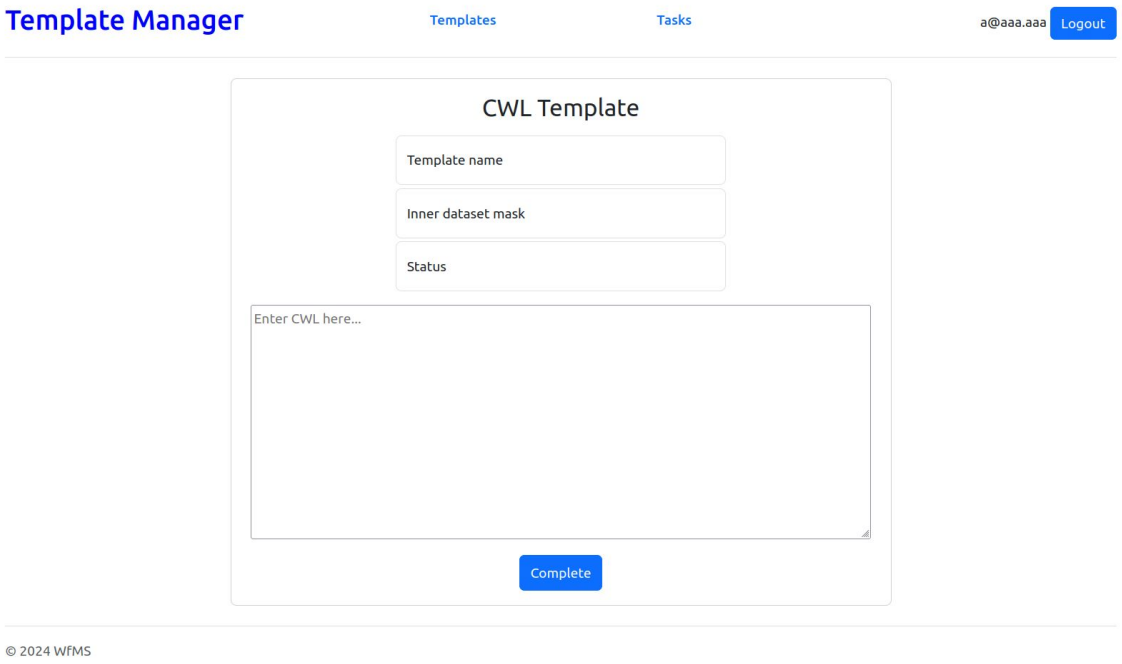
Description of the current implemented API's

Service for interaction with user

- Registration and authorization of users with different rights;
- CWL template/tasks output;
- Creation of CWL templates by superuser;
- CWL template status changes by superuser;
- Store template in the database;

- FastAPI Users
- JWT-token





Examples of Templates and Tasks

- Viewing templates and tasks is available to all users who have completed the authorization process;
- Template creation is only available to superusers;

Template Manager Templates Tasks a@aaa.aaa Logout

Create template

template_id	name	inner_dataset_mask	description	status
1	template1	.test.	{"steps": {"decoding": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}, "reconstruction": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}}}	ACTUAL
2	template2	.test.	{"steps": {"decoding": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}}}	ARCHIVED

Created template

Template Manager Templates Tasks a@aaa.aaa Logout

task_id	wflow_id	exec	args	rank	device	mode	retry	datas_in_id	datas_out_id	datas_log_id	status
11	6	processing_program	cable_map	1	CPU	map	5	26	27	28	IN_PROGRESS
12	6	processing_program	cable_map	1	CPU	map	5	27	29	30	IN_PROGRESS
13	7	processing_program	cable_map	1	CPU	map	5	31	32	33	IN_PROGRESS
14	7	processing_program	cable_map	1	CPU	map	5	32	34	35	IN_PROGRESS
15	8	processing_program	cable_map	1	CPU	map	5	36	37	38	IN_PROGRESS
16	8	processing_program	cable_map	1	CPU	map	5	37	39	40	IN_PROGRESS
17	9	processing_program	cable_map	1	CPU	map	5	41	42	43	IN_PROGRESS
18	9	processing_program	cable_map	1	CPU	map	5	42	44	45	IN_PROGRESS
19	10	processing_program	cable_map	1	CPU	map	5	46	47	48	IN_PROGRESS
20	10	processing_program	cable_map	1	CPU	map	5	47	49	50	IN_PROGRESS
21	11	processing_program	cable_map	1	CPU	map	5	51	52	53	IN_PROGRESS
22	11	processing_program	cable_map	1	CPU	map	5	52	54	55	IN_PROGRESS
23	12	processing_program	cable_map	1	CPU	map	5	56	57	58	IN_PROGRESS
24	12	processing_program	cable_map	1	CPU	map	5	57	59	60	IN_PROGRESS

WfMS task description

RabbitMQ configured queues

Exchange: dsm.register

Overview

Bindings

This exchange

⇓

To	Routing key	Arguments	
dsm.register.dataset.close	dataset.close		Unbind
dsm.register.dataset.delete	dataset.delete		Unbind
dsm.register.dataset.input	dataset.input		Unbind
dsm.register.dataset.upload	dataset.upload		Unbind
dsm.register.file.input	file.input		Unbind
dsm.register.file.process	file.process		Unbind
dsm.register.file.process.reply	file.process.reply		Unbind

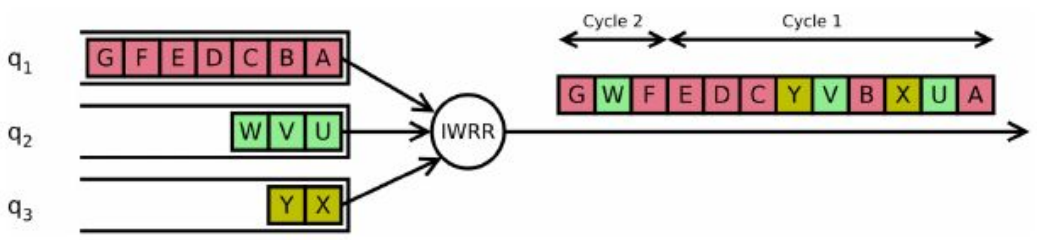
Exchange	Routing Key	Appointment
dsm.register (direct)	file.input	Receiving information about incoming files to the input buffer
	file.process	Receiving information about new files, received during processing
	dataset.close	Accepting a request to close a dataset
	dataset.upload	Accepting an application to upload files in a dataset to an external storage
	dataset.delete	Accepting a request to delete files in a dataset on the internal storage

R&D

- Jobs scheduling algorithm
- Partitioning of a task
  - Imagine a multitasking operating system.
  - Each dataset represents a process, and each record within a dataset is like a thread within that process.
  - The algorithm acts as the operating system's scheduler, allocating processing time to threads based on their priority.
- Chunk size and rank/priority of a job as a basic control unit:

$$rank_{i+1} = \alpha \times x_i + \beta \times y_i + \gamma \times rank_i$$

$$x_i - aging, y_i - retries$$



Interleaved Weighted round-robin

Algorithm 1 Task Scheduling Algorithm

Variables:

global_queue – global queue with tasks
dataset – array of datasets
N – number of datasets
rank_max – maximum task priority
heap – binary heap storing maximum task priorities
rank – array with task priorities

Algorithm:

```
1: initilize_datasets(dataset)
2: build_heap(rank)
3: while true do
4:    rank_max = heap.top()
5:    for r = 1 to rank_max do
6:       for i = 1 to N do
7:          if not dataset[i].chunk.empty() and rank[i] ≥ r then
8:             await dataset[i].chunk.cur_item
9:             update(dataset[i].chunk – ¿ cur_item)
10:         else if dataset[i].chunk.empty() then
11:            if dataset[i].chunk.cur_item then
12:               dataset[i] = global_queue.head()
13:            end if
14:            update(rank[i])
15:            update(heap)
16:         end if
17:      end for
18:   end for
19: end while
```

Proposed task-partitioning algorithm