St Petersburg University

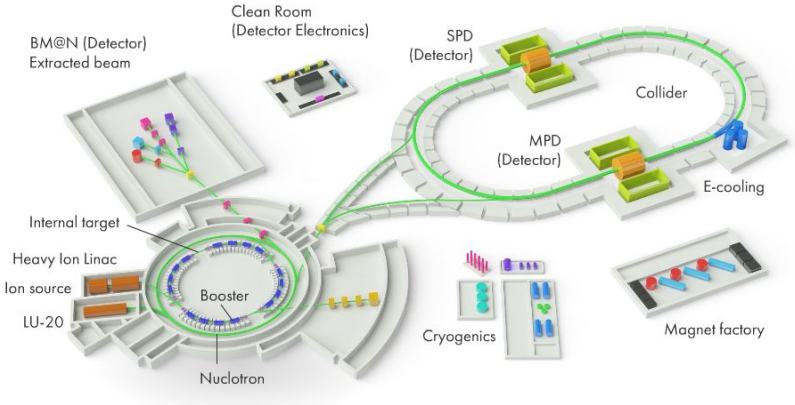
SPD NICA

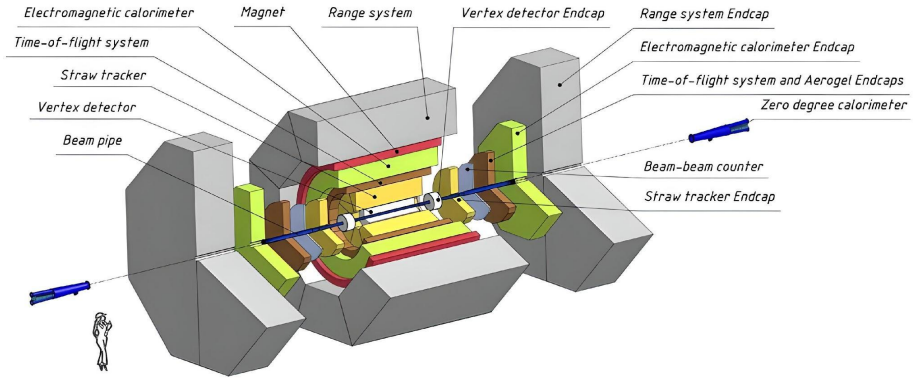
SPD On-line Filter: Workload management system and Pilot Agent

Greben N.V., Romanychev L.R., Oleynik D.A., Degtyarev A.B.
Saint Petersburg State University

10th International Conference
«Distributed Computing and GRID Technologies in Science and Education» (GRID-2023)
06 july 2023

SPD experiment: brief reminder





- 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

- Number of registration channels in SPD ~ 500000
- ~ 3 MHz event rate (at max luminosity) = pileups
  - ~ 20 GB/s (or 200PB/year) “raw” data
- Selection of physics signal requires momentum and vertex reconstruction
  - no simple trigger is possible

SPD OnLine Filter - High-throughput Computing system

- HTC is defined as a type of computing that simultaneously executes numerous simple and computationally independent jobs to perform a data processing task.
- Since each slice of data can be processed simultaneously, it can be applied to data aggregated by SPD DAQ.
- Data processing should be multistaged for providing the efficient usage of computing resources
  - One stage of processing → Task
  - Processing of data unit (file) → Job

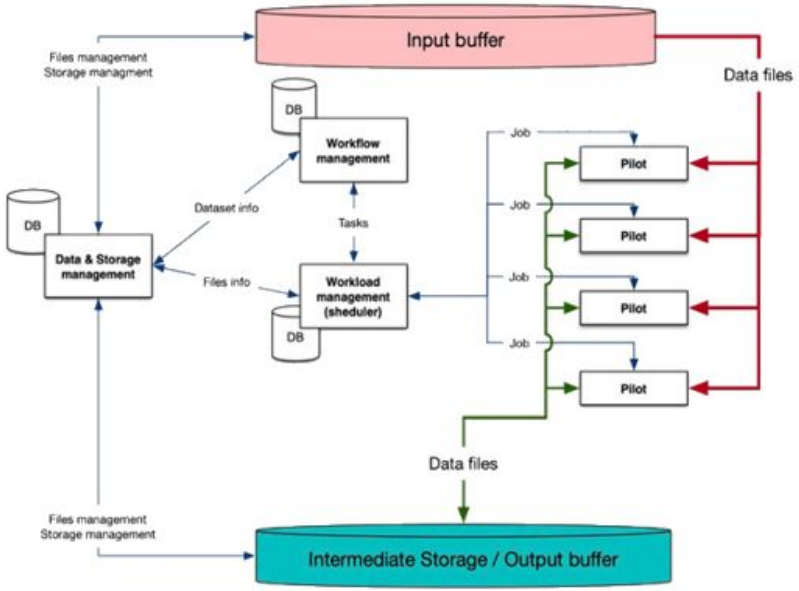
SPD Online Filter - Middleware

The following are the main services responsible for delivering all functionality

- **Data management system**
  - data catalog and lifetime support
- **Workflow Management System:**
  - defining and execution of processing chains by generating of required number of computational tasks
- **Workload management system:**
  - Generation of required number of jobs to perform task;
  - Dispatch jobs to working nodes through pilots;
  - Control of jobs executions;
  - Control of pilots (identifying of "dead" pilots)
  - Efficient resource management

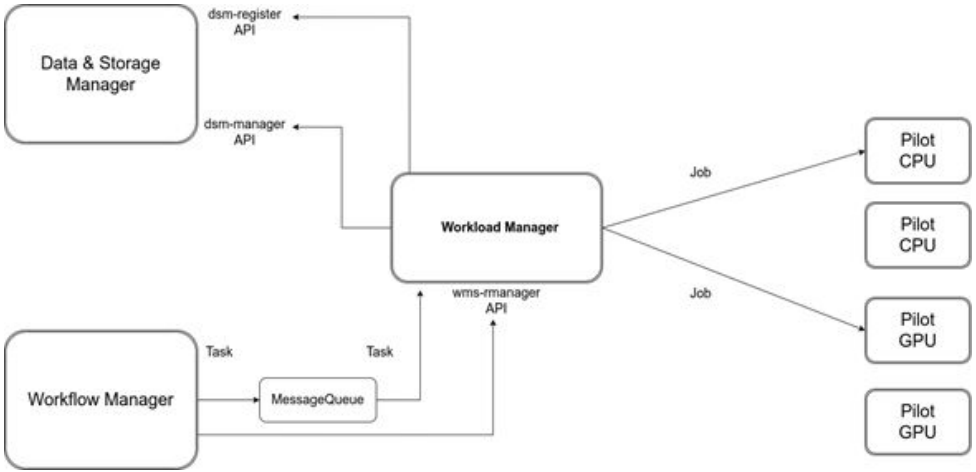
Key features

- ❖ Event unscrambling: transform data into event's oriented format:
  - ➢ Partial preliminary reconstruction;
- ❖ Filter 'boring' events and leave only 'hot';
- ❖ Settle output data, merge events into files and files in datasets for future processing.
- ❖ Prepare data for online data quality monitoring etc.



Workload Management System

- Server part: managing of tasks throughput, controlled tasks slicing to jobs, distribution of jobs across resources, managing of job execution
- Agent application (Pilot): payload execution environment, collecting and providing of job monitoring information



WFMS interactions

- ➔ Accepting of data processing tasks from top level management of the tasks: canceling, changing of priority
- ➔ Task compilation progress report
- ➔ List of produced files in dataset

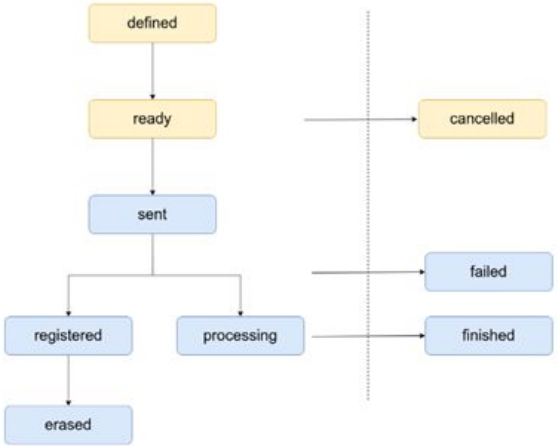
DSM interactions

- ➔ Dataset structure gathering
- ➔ File/logs registration in the associated dataset
- ➔ Dataset closure

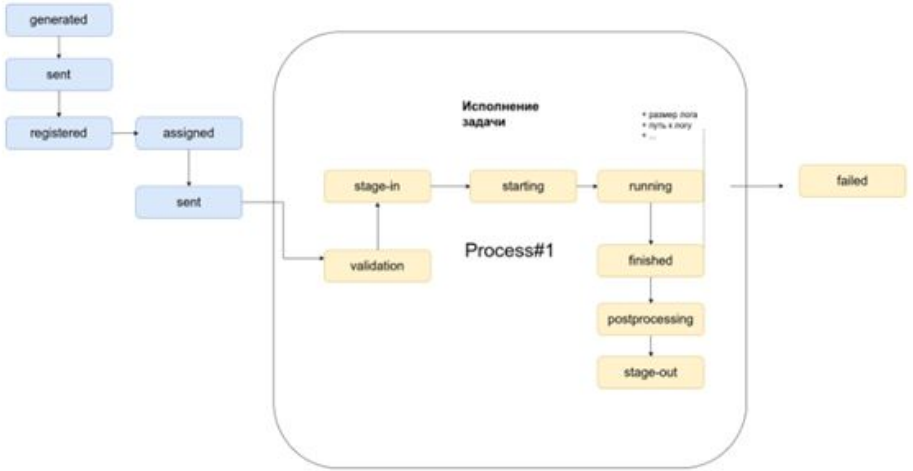
Task and jobs

- Task - unit of workload that is responsible for processing a block of homogeneous data
- Task completion criteria is processing the whole block of data

- Job (payload) is a unit of work which process a unit of data



Task state DAG



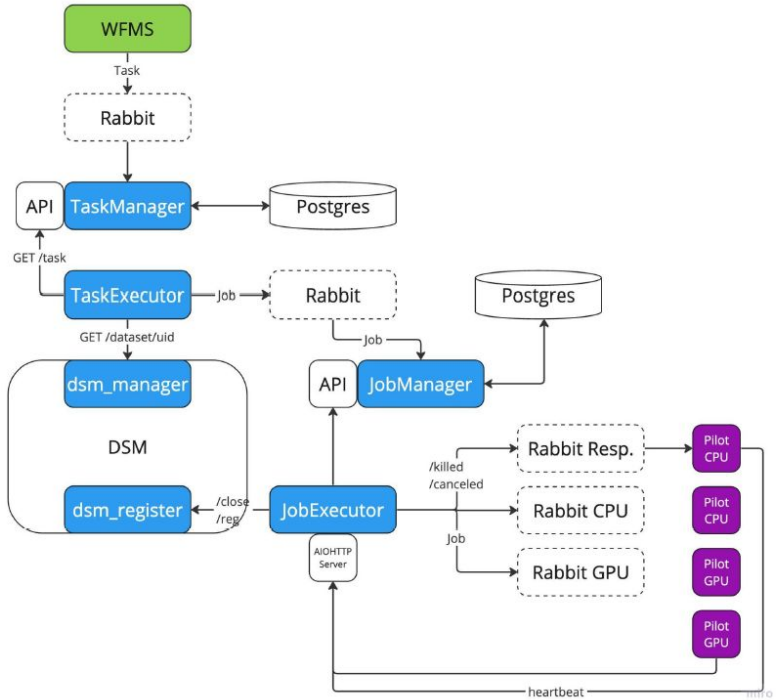
Job state DAG

WMS requirements

Task registration: formalized task description, including job options and required metadata registration.

Jobs defenition: 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.



Workload management system architecture

Architecture and functionality of WMS

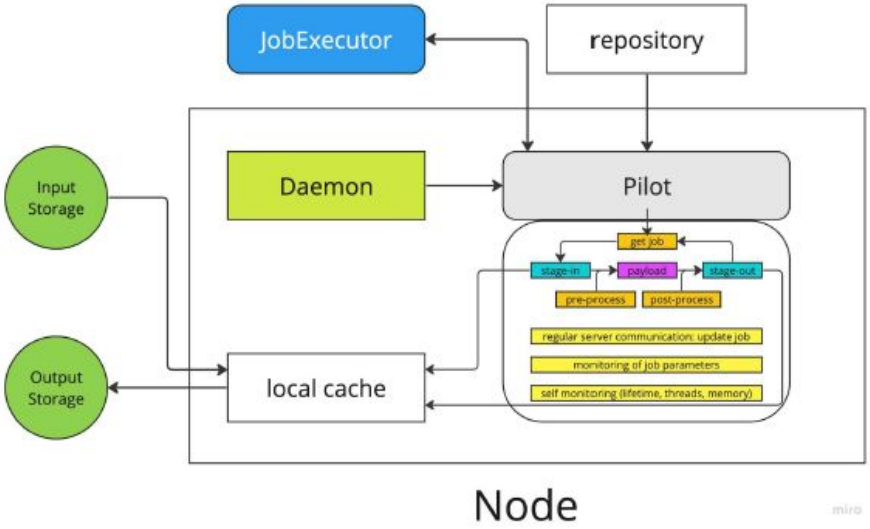
- **Microservice architecture**
  - **task-manager –** responsible for interaction with ***workflow management system,*** providing functionality for task registration, cancellation, progress tracking, and final report generation
  - **task-executor –** responsible for: jobs generation by the dataset content and task type, management of the task queue.
  - **job-manager –** responsible for: receiving jobs from the broker, jobs registering, transfering ready jobs to the job-executor, provides internal API's for jobs and files metadata.
  - **job-executor –** dispatches jobs to pilots and oversees their completion, data registration.

Method	API	Input	Output
WMFS requests			
Information gathering about the status of jobs	GET /task/<id>/summary/jobs	ID	{ *total :" running :" canceled :" killed :" failed :"* }
Information gathering about the output files	GET /task/<id>/summary/files	ID	{ *total :" failed :"* }
Task's priority changing	PUT /task/<id>/rank	Task ID + rank	Task info
Task cancelling	PUT /task/<id>/canceled	Task ID + status	Task info

task-manager API

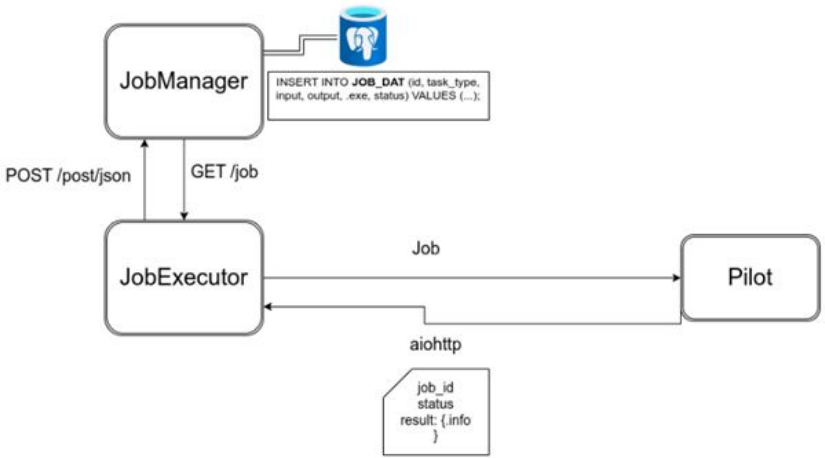
Architecture and functionality of Pilot Agent

- Standalone set of applications:
  - Daemon - control continuous (one by one) launching of pilot application on working node
  - Pilot application - communicate with **jobExecutor** to get a job and update job state, perform payload execution



WSM-Pilot communication

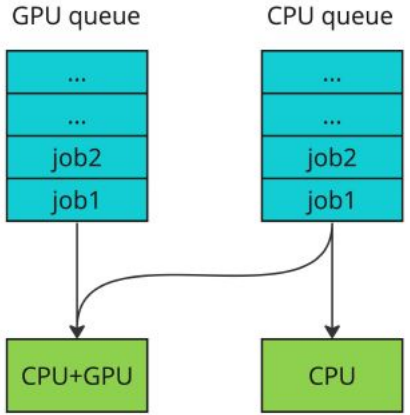
- Two channels of communication:
  - REST (aiohttp)
  - Message queue (RabbitMQ)



Two types of architectures:

- Multi-CPU
- Multi-CPU + GPU

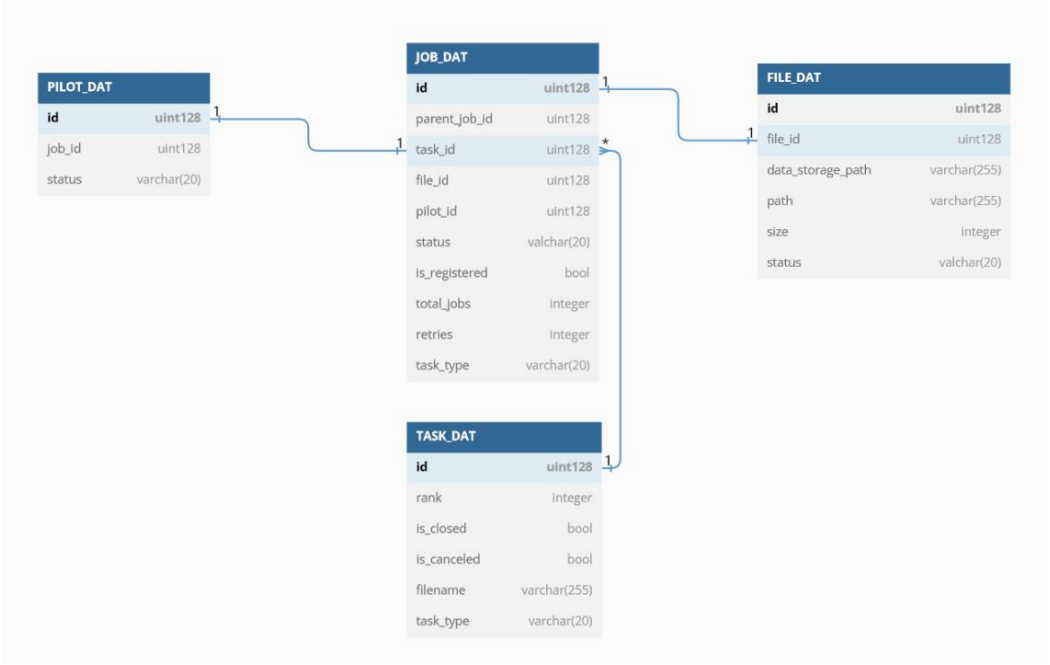
Pilot identify type of resource and pickup job from proper queue.
In case there is no GPU oriented jobs, pilot will take the job from "CPU" queue.



Database design

Tables:

- **JOB_DAT** - catalogue with current jobs waiting to be processed in the system (job queue)
- **TASK_DAT** - task queue
- **PILOT_DAT** - pilots catalogue
- **FILE_DAT** - files catalogue (for registered jobs)



Database scheme

Task partitioning

- The **task-executor** microservice is responsible for generating jobs for a set of files associated with dataset which should be processed by task.
- Jobs generation process should be performed asynchronously, taking into account the possibility of simultaneous processing of several tasks.
- Within each task (dataset), jobs are generated in chunks.
- The algorithm must guarantee the processing of each dataset in a proportion that will depend on the priorities of the tasks.

Asynchronous job generation

Tech stack

Main

- Python 3.11
- Docker + Docker Compose
- PostgreSQL

Database

- psycopg
- asyncpg
- alembic
- SQLAlchemy

Frameworks

- FastAPI
- aio-pika (RabbitMQ)
- aiohttp

Next steps

WMS

Short-term

- ➔ Complete the prototyping for the rest of the microservices: task-executor, job-executor
- ➔ Integration testing
- ➔ DLQ (Death letter queues) implementation

Mid/long-term

- ➔ Modeling the workload on the system and measuring its throughput at different load levels
- ➔ Profiling: CPU's, RAM, IO operations
- ➔ The RED Method (Rate, Errors and Duration):
  - ◆ requests rate, number of failed requests, requests processing time

Pilot

Short-term

- ➔ Complete integration of WMS and pilot application

Mid/long-term

- ➔ Analysis of system performance according to known methodologies
  - ◆ USE Method (Utilization, Saturation and Errors)

Conclusion

As a result, the system design has been carried out and the main components of the workload management system have been described.

- The architecture of the entire **WMS** has been developed, taking into account the features and business logic of such systems as **WFMS**, **DMS** and agent application.
- REST API interfaces with the **WFMS** and **DMS** been described.
- Task partitioning algorithm has been proposed.
- **task-manager** prototype has been developed.
- A coordinated interface is provided between the agent application and the WMS.
- A tool template is obtained that can be used to run simple tasks on a cluster.

Next, it is planned to debug integration with other systems and move to the stage of extended functional testing of the computing system.