SPD Online Filter Middleware

Status Update

Nikita Greben

Meshcheryakov Laboratory of Information Technologies

XI SPD Collaboration Meeting

May 20, 2026 Tomsk, Russia

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.

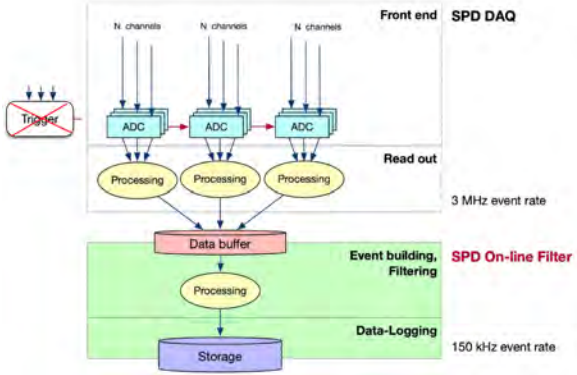
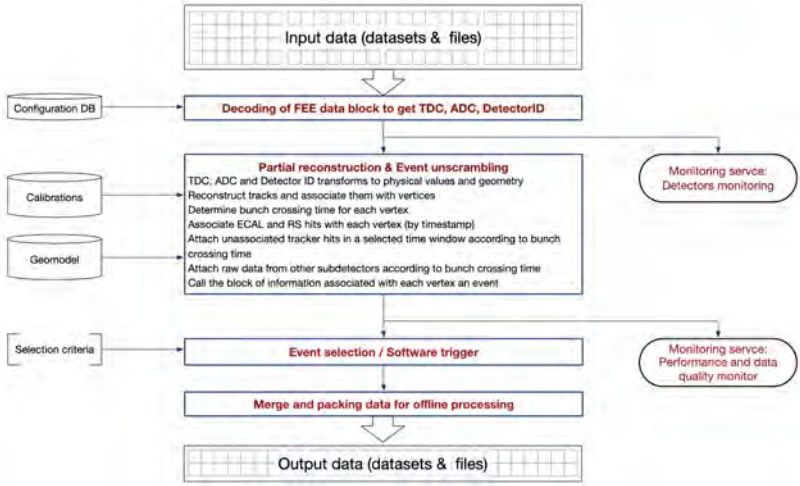
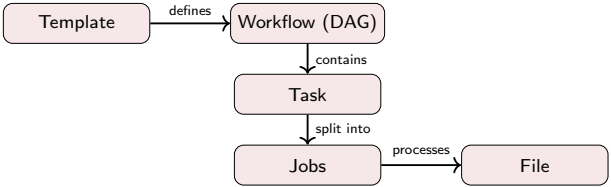


Figure 1: Triggerless dataflow in SPD

Data processing workflow in SPD Online Filter



Core concepts



- **Template**: a type of entire data processing cycle: executable, parameters, input/output dataset schemas, resource requirements. Instantiates a workflow (DAG). One template can be used to create multiple workflows, one of the reasons why had to move from **CWL**.
- **Workflow (DAG)**: a directed acyclic graph of tasks whose edges are data dependencies: the output dataset of one task is the input dataset of the next. The **WfMS** directly responsible for managing the workflow.
- **Task**: a template bound to a specific input dataset. The **WfMS** admits and tracks tasks statyses.
- **Job**: one execution unit, produced by splitting a task across its input files. The **WMS** dispatches to a pilot.
- **File**: the atomic unit of data: DAQ output, intermediate results, final results. Stored by the **DSM**. Files are grouped into *datasets*.

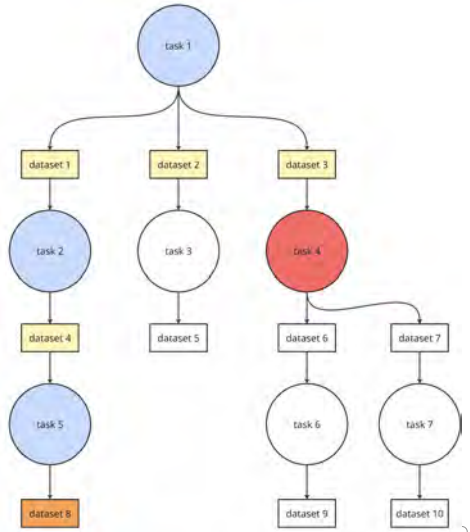
The main goal of SPD Online Filter Middleware

Goal

Keep the **input buffer of raw DAQ data as small as possible** at all times.

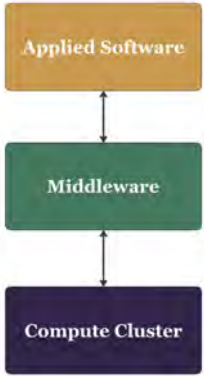
To do so the middleware must execute an arbitrary number of concurrent workflows in the most efficient way. This is a major schedulign problem.

At runtime, many workflows of different templates evolve in parallel across the cluster, tasks become ready when their input datasets are produced, and the middleware must always decide *which task to admit, which job to dispatch*.

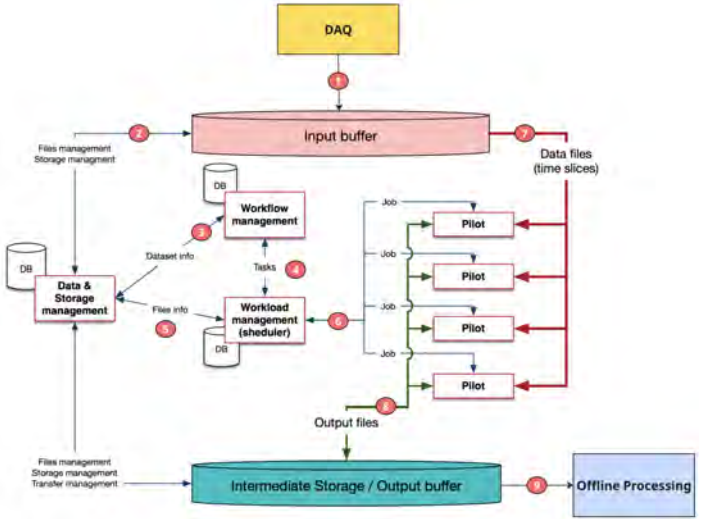


Reminder

- **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



The cluster and the deployment problem

Target cluster

256-GPU / 1 TB RAM / 120 TB NFS provisioned on oVirt as virtual machines. Each pilot agent is one VM running the worker process.

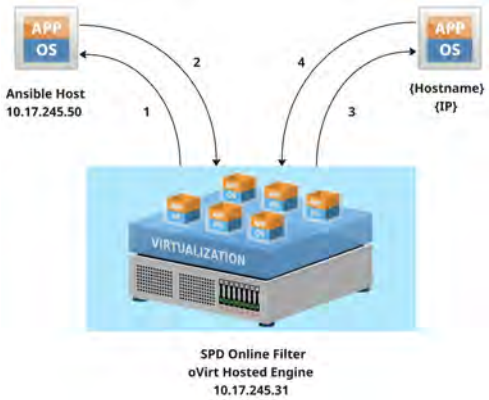
Deploying one pilot-agent :

$$\text{Created} \rightarrow \text{NFS-mounted} \rightarrow \text{Daemonized}$$

We need to bring N pilots up, track their state, scale up and down, and tear down repeatedly, idempotently, across the cluster.

Pure Ansible	This is an good *configuration* management tool, but no *state* management. It does not know which VMs exist, what their resource are, or whether a previous run completed partially.
Our tool: `pilot-ctl`	A small CLI that plays the Terraform (state management) role for out environment. It owns the state, generates Ansible inputs dynamically.

Middleware deployment on oVirt via Ansible (IaC)



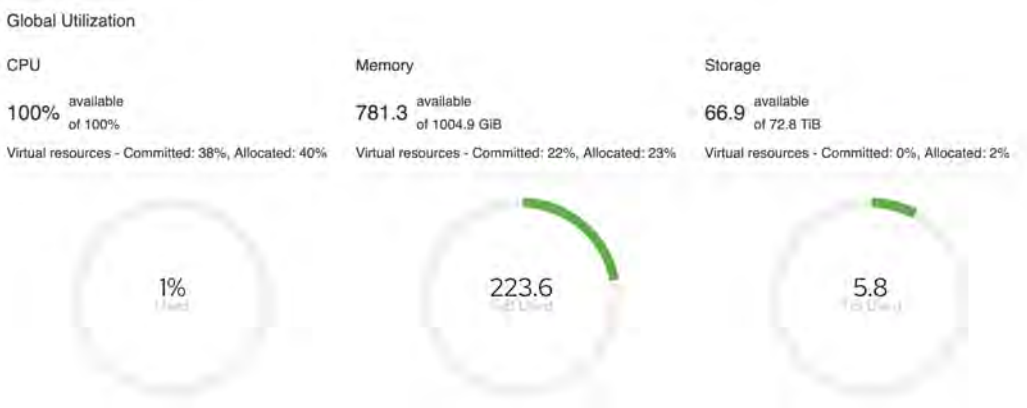
Automated deployment pipeline via Ansible

- **VM provisioning via oVirt API.** Ansible clones VMs from a pre-existing template; Cloud-Init (ConfigDrive) injects static IPs, DNS, and SSH keys on first boot.
- **OS configuration over SSH.** Installs OS-level dependencies and persistently mounts the required NFS directories in `/etc/fstab`.
- **Pilot software deployment.** Extracts the Pilot software from NFS, dynamically assigns a unique pilot ID, and registers a user-level daemon.
- **Result.** A single master playbook (main.yml) handled the entire lifecycle, from an empty cluster to fully functional pilot daemons: `ansible-playbook -i hosts.ini main.yml`

oVirt monitor: $\approx$ 20 active pilot-agents

	Name	Com	Host	IP Addresses	FQDN	Cluster	Data C	Memory	CPU
	pilot-agent-20		sof-srv2.jinr.ru	10.17.245.96 fe80::546f:e0ff:fe38:66	pilot-agent-20	SOF	JINR	19%	3%
	pilot-agent-21		sof-srv4.jinr.ru	10.17.245.184 fe80::546f:e0ff:fe38:67	pilot-agent-21	SOF	JINR	20%	3%
	pilot-agent-22		sof-srv2.jinr.ru	10.17.245.135 fe80::546f:e0ff:fe38:68	pilot-agent-22	SOF	JINR	21%	4%
	pilot-agent-23		sof-srv3.jinr.ru	10.17.245.188 fe80::546f:e0ff:fe38:69	pilot-agent-23	SOF	JINR	19%	3%
	pilot-agent-24		sof-srv4.jinr.ru	10.17.245.131 fe80::546f:e0ff:fe38:6a	pilot-agent-24	SOF	JINR	21%	4%
	pilot-agent-25		sof-srv2.jinr.ru	10.17.245.207 fe80::546f:e0ff:fe38:6b	pilot-agent-25	SOF	JINR	19%	3%
	pilot-agent-26		sof-srv4.jinr.ru	10.17.245.146 fe80::546f:e0ff:fe38:6c	pilot-agent-26	SOF	JINR	20%	3%
	pilot-agent-27		sof-srv3.jinr.ru	10.17.245.72 fe80::546f:e0ff:fe38:6d	pilot-agent-27	SOF	JINR	20%	4%
	pilot-agent-28		sof-srv2.jinr.ru	10.17.245.113 fe80::546f:e0ff:fe38:6e	pilot-agent-28	SOF	JINR	21%	3%
	pilot-agent-29		sof-srv3.jinr.ru	10.17.245.171 fe80::546f:e0ff:fe38:6f	pilot-agent-29	SOF	JINR	20%	5%
	pilot-agent-30		sof-srv4.jinr.ru	10.17.245.203 fe80::546f:e0ff:fe38:70	pilot-agent-30	SOF	JINR	19%	4%
	pilot-agent-31		sof-srv2.jinr.ru	10.17.245.166 fe80::546f:e0ff:fe38:71	pilot-agent-31	SOF	JINR	19%	3%
	pilot-agent-32		sof-srv4.jinr.ru	10.17.245.85 fe80::546f:e0ff:fe38:72	pilot-agent-32	SOF	JINR	20%	3%
	pilot-agent-33		sof-srv3.jinr.ru	10.17.245.83 fe80::546f:e0ff:fe38:73	pilot-agent-33	SOF	JINR	20%	4%
	pilot-agent-34		sof-srv2.jinr.ru	10.17.245.140 fe80::546f:e0ff:fe38:74	pilot-agent-34	SOF	JINR	19%	3%
	pilot-agent-35		sof-srv3.jinr.ru	10.17.245.211 fe80::546f:e0ff:fe38:75	pilot-agent-35	SOF	JINR	20%	3%

oVirt usage

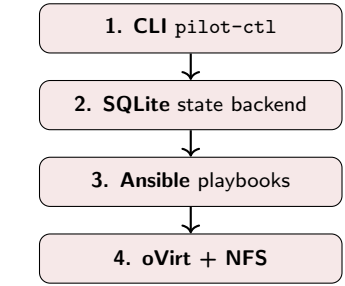

Global Utilization
CPU
100% available of 100%
Virtual resources - Committed: 38%, Allocated: 40%
1%
Memory
781.3 available of 1004.9 GiB
Virtual resources - Committed: 22%, Allocated: 23%
223.6
Storage
66.9 available of 72.8 TiB
Virtual resources - Committed: 0%, Allocated: 2%
5.8

Encountered issues during deployment

- No default Cloud-Init datasources: oVirt does not have a native, built-in instance metadata service (like the HTTP-based AWS or Azure IMDS) - no way to get VM metadata via oVirt API.
- Cloud-Init payload failure due to template having an installation ISO left in the virtual CD-ROM drive, blocking oVirt from injecting the network configuration.
- NetworkManager (RHEL 9+) looks at the legacy format network configuration file generated by oVirt.

These encountered issues and others led us to recondider the approach and move from static IP's to **DHCP**, and give each VM a **FQDN (Fully Qualified Domain Name)**.

pilot-ctl
Our internal developer tool



1. pilot-ctl CLI. Installed on the Ansible control node, requires ansible dependencies.

2. State (SQLite). VM table holds `id`, `name`, `cpu`, `ram`, `status`, `created_at`. Survives crashes, can make a retry on failure.

3. Orchestration (Ansible). Dynamic IaC: `pilot-ctl` generates `vms_definition.yml` from SQLite on every launch, then executes `ansible-playbook`.

Usage

```
$ pilot-ctl scale --count 20 --cpu 2 --ram 8GiB
$ pilot-ctl list
$ pilot-ctl destroy --all
```

pilot-ctl example

```
nick@localhost.localdomain ~/pilot-ctl/pilot-control-cli git:(main)  2 files changed, 97 insertions(+), 10 deletions(-) (11m 25.33s)
cargo run -- scale --count 18 --cpu 2 --ram 4GiB
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/pilot-ctl scale --count 18 --cpu 2 --ram 4GiB`
Scaling: count=18 cpu=2 ram=4GiB
Wrote ansible/vms_definition.yml (21 pilots).
Vault password:
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
```

pilot-ctl result: 18 pilot-agents created and launched

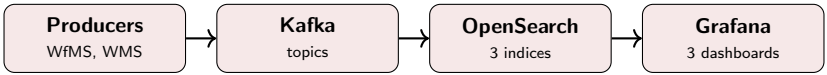
```
PLAY RECAP *********************************************************************************************************************
localhost                  : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-1              : ok=16   changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-2              : ok=16   changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-20             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-21             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-22             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-23             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-24             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-25             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-26             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-27             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-28             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-29             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-30             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-31             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-32             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-33             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-34             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-35             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-36             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-37             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
pilot-agent-38             : ok=16   changed=11   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

Promoted 19 pilot(s) to Running.
```

11 minutes starting from VM creation to running state on all 18 pilot-agents.

Observability: Kafka → OpenSearch → Grafana

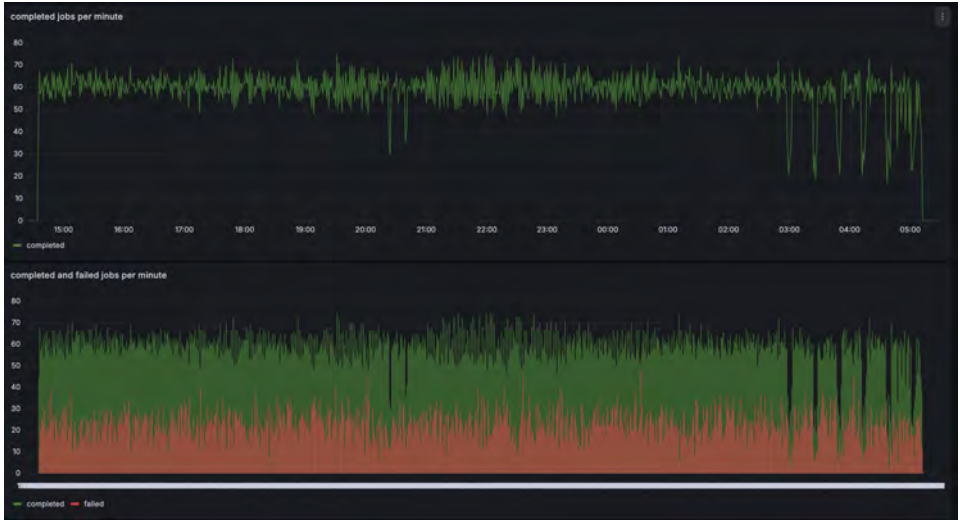
Three telemetry streams flow from the middleware microservices into OpenSearch via Kafka, and shows in Grafana.



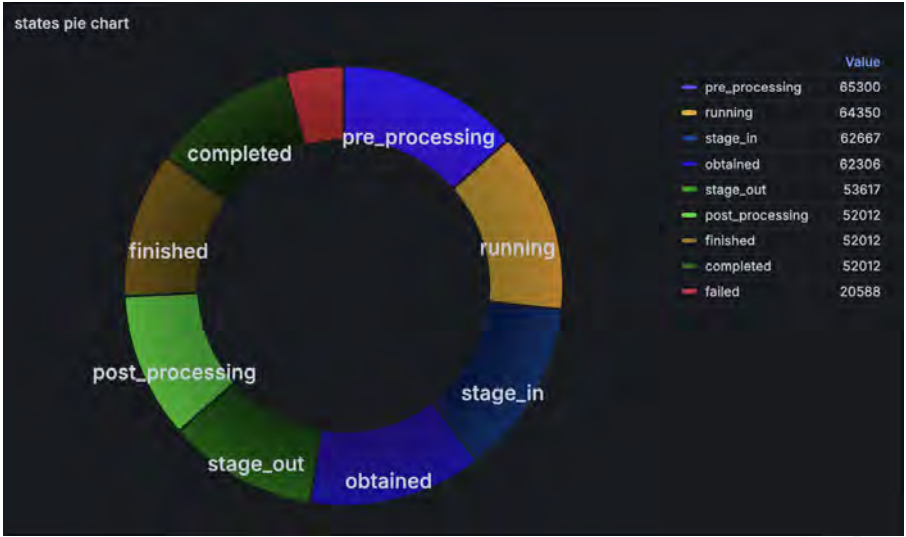
What gets collected:

Stream	Time field	Captured fields
Job lifecycle events	`datetime`	`job_id`, `task_id`, `state`, `error_code`, `exit_code`, `pilot_id` on every state transition
Pilot heartbeats	`heartbeat_ts`	`pilot_id`, `cpu_load`, memory, disk IO, network – at fixed interval
Task & dataset	`datetime`	`task_id`, `name`, `datasets`, ...

Jobs dashboards example: jobs per minute



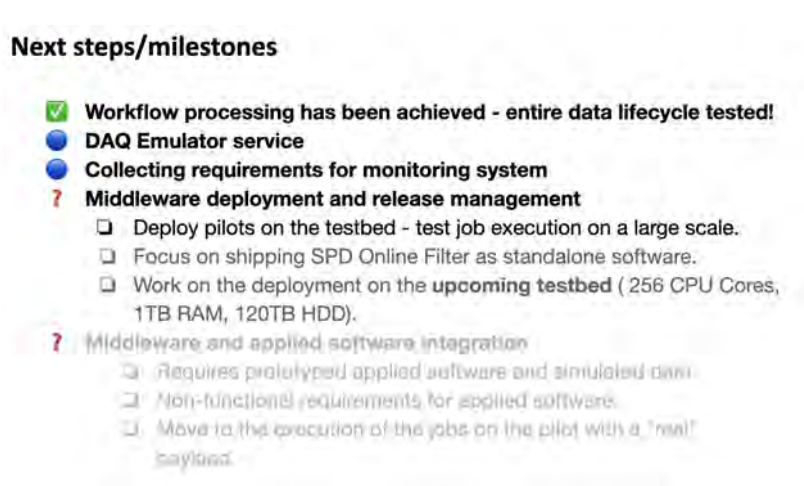
Jobs dashboards example: states pie chart



Pilots dashboards example

Tasks dashboards example

Plan from the previous collaboration meeting



Next steps/milestones

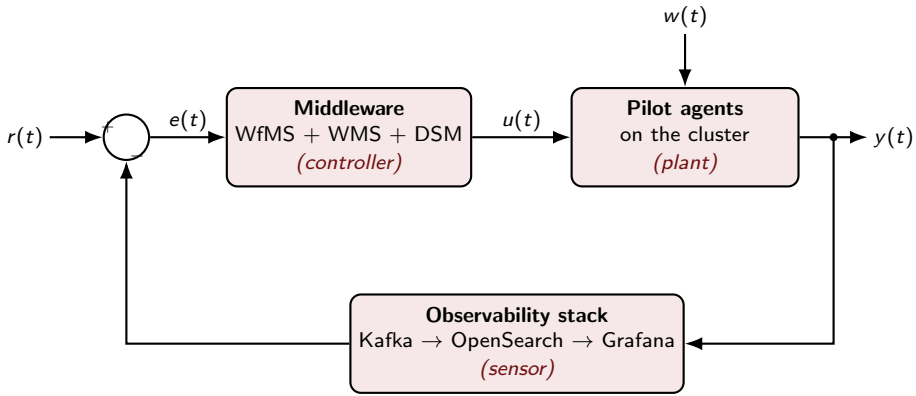
- ✅ **Workflow processing has been achieved - entire data lifecycle tested!**
- 🔵 **DAQ Emulator service**
- 🔵 **Collecting requirements for monitoring system**
- ? **Middleware deployment and release management**
  - ❑ Deploy pilots on the testbed - test job execution on a large scale.
  - ❑ Focus on shipping SPD Online Filter as standalone software.
  - ❑ Work on the deployment on the **upcoming testbed** (256 CPU Cores, 1TB RAM, 120TB HDD).
- ? 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

Summary

- **Deployment**
  - Achieved deployment of the entire SPD Online Filter Middleware on the cluster;
  - Reproducible deployment pipeline for *pilot-agents* (workers) via Ansiblea and own developed tool;
- **Observability**
  - Setted up pipeline for Kafka $\rightarrow$ OpenSearch $\rightarrow$ Grafana;
  - Three live dashboards for tasks, jobs and pilots.
- **Next plans**
  - Perform complex loading tests to see how the middleware behaves under different situations;
  - Further debugging and optimization.
  - Middleware and applied software integration.

A control-theoretic view

The middleware as a closed loop controller – eurika moment



About diagram. $r(t)$ is workflow goals (*e.g.* throughput target, storage usage); $u(t)$ the action: tasks, jobs, and ranks dispatched; $y(t)$ telemetry observed (heartbeats, state transitions, queue depth); $w(t)$ the disturbance: pilot failures, network etc. The controller acts on the error $e(t) = r(t) - y(t)$.

Observability *is* state feedback

Each dashboard panel is basically partly component of the system's output $y(t)$. The middleware is not just a set of microservices , it is a **closed-loop dynamical system**.

Symbol	Meaning	Measured today
$L(t)$	Cluster load (fraction busy)	`avg(cpu_load)`: pilot health
$\mu(t)$	Aggregate processing rate	`obtained`→`completed` latency: job lifecycles
y_i	Retry / failure count per job	`state:failed` per minute + `error_code` field: job Lifecycles
$P(t)$	Active pilot count	Distinct value count over heartbeat: pilot health
DAG	Task / dataset lineage	. . . task flow
$Q(t)$	Ready-queue depth	**Not yet**: RabbitMQ queue depth panel (roadmap)
$S(t)$	NFS storage free capacity	**Not yet**: NFS free GB info (roadmap)

Note

Five of seven state-space components are already measured; the remaining two (Q, S) are on the way. Once recived, the middleware should be a *fully observed* dynamical system, then th formalism of the next four slides applies directly.

Formal statement: state equation

The middleware evolves as a continuous in time system:

$$\dot{x}(t) \;=\; f\big(x(t),\, u(t),\, w(t)\big), \qquad y(t) \;=\; h\big(x(t)\big).$$

State $x(t)$

- $S(t)$ — free storage capacity (NFS)
- $Q(t)$ — ready queue depth (RabbitMQ)
- $\mu(t)$ — aggregate pilot processing rate

Control $u(t)$

- $\lambda_{\text{admit}}(t)$ — admission rate *(long-term)*
- $\mathbf{r}(t)$ — rank vector *(short-term)*
- $\sigma(t)$ — suspend/resume *(medium-term)*

Disturbance $w(t)$

pilot failures, network and storage latency, etc.

Output $y(t)$

Heartbeats, state transitions, queue telemetry, basicaly the Grafana dashboards from the previous slides.

Fluid approximation. With $N \gg 1$ jobs, we can treat this discrete problem as a continuous flow. The rank dynamics on the next slide is the discrete time implementation of this continuous controller.

Rank update

Per job k, rank evolves between scheduling rounds i as:

$$r_{i+1}^{(k)} = \underbrace{\gamma_k\, r_i^{(k)}}_{\text{persistence}} + \underbrace{\alpha \ln\!\left(x_i^{(k)} + 1\right)}_{\text{aging}} - \underbrace{\beta\, 2^{y_i^{(k)}}}_{\text{failure penalty}} + \underbrace{\delta\,(1 - L)}_{\text{load relief}}.$$

Stacking over the N ready jobs gives a vector recurrence:

$$\mathbf{r}_{i+1} = \Gamma\, \mathbf{r}_i + \alpha\, \ln(\mathbf{x}_i + \mathbf{1}) - \beta\, 2^{\mathbf{y}_i} + \delta\,(1 - L)\,\mathbf{1}, \qquad \Gamma = \operatorname{diag}(\gamma_1, \ldots, \gamma_N).$$

This is a **linear discrete-time system with bounded nonlinear part** exactly the form $\mathbf{r}_{i+1} = \Gamma\, \mathbf{r}_i + \mathbf{f}_i$ with $\|\mathbf{f}_i\|$ bounded. The standard tools of discrete linear systems apply: eigenvalue stability (next slide), reachability, sensitivity to parameter drift (not yet studied).

Note

The formula is a *working hypothesis* for the rank-driven scheduler. Four parameter strategies $(\alpha, \beta, \gamma, \delta)$ are not fixed and arguibly will need an adjustmen, the study is underway.

Stability

$\Gamma = \mathrm{diag}(\gamma_1, \ldots, \gamma_N)$ is diagonal, so its eigenvalues *are* the γ_k themselves. Discrete-time stability of the rank dynamics

$$\mathbf{r}_{i+1} = \Gamma\, \mathbf{r}_i + (\text{bounded forcing}) \qquad \text{requires} \qquad \big|\lambda_k(\Gamma)\big| < 1 \iff |\gamma_k| < 1 \ \ \forall k.$$

Intuition. γ_k is the persistence of yesterday's rank for job k. With $\gamma_k < 1$ the past decays; with $\gamma_k \geq 1$ it compounds and ranks march to infinity.

Regime	γ_k	**Behavior**
Divergent	≥ 1	$\mathbf{r}_i$ grows unboundedly: ordering meaningless, scheduler broken
Good spot	0.7–0.9	History aware and responsive to aging and load
Bad	> 0.95	Stable but adapts too slowly, meaning that history dominates fresh signals

Middleware = an OS for the cluster

The middleware orchestrates compute, storage, and communication across the cluster which is basicaly exactly the job of an operating system.

Operating system	**SPD Online Filter**
Kernel	Middleware: WfMS + WMS + DSM
Long-term scheduler (task admission)	WfMS admission control *(heuristic prototype)*
Medium-term scheduler (swapper)	**[Under development]**
Short-term scheduler (CPU scheduler)	WMS: IWRR + rank based dispatch
File system	Data & Storage Management
Interprocess communication	RabbitMQ / Kafka
System calls	REST API between microservices

The OS hierarchy

The OS textbook three level scheduler hierarchy not justa metaphor for our middleware, it can offer a more unified approach to the scheduling, end perhaps, simplify architecture.
The control vector $u(t) = (\lambda_{\text{admit}}, \mathbf{r}, \sigma)$.

OS level	**Control**	**Today**	**What the model demands**
Long-term (admission)	$\lambda_{\text{admit}}(t)$	WfMS prototype: heuristic input-buffer boundary; handles branched DAG templates	Storage-pressure aware admission
Medium-term (swapper)	$\sigma(t)$	**Developing**	`SUSPEND` / `RESUME` / `CANCEL` on tasks required for $\sigma(t)$ to exist as an object
Short-term (CPU scheduler)	$\mathbf{r}(t)$	IWRR + rank-proportional fairness	Stable iff $\lvert\gamma_k\rvert < 1$ for all k

Summary once again

Engineering

Working middleware on the 256-GPU / 1 TB RAM / 120 TB NFS cluster. Deployment using Infractructure as Code via Ansible and own `pilot-ctl` tool. First monitoring setup with Kafka → OpenSearch → Grafana. Three live dashboards.

Theory

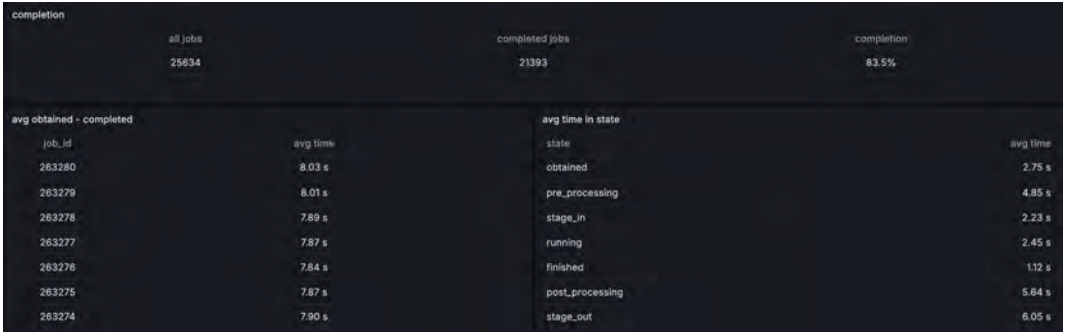
The middleware is a closed lloop dynamical system in state space form. The rank update is a linear discrete time system, stable iff $|\lambda(\Gamma)| < 1$. The OS three level scheduler hierarchy *is* the decomposition of control vector $u(t)$.

Next

Research the system from the control-theoretic perspective to develop a more unified, rigorous and concrete scheduler.

Backup

Jobs dashboards example



completion

all jobs	completed jobs	completion
25634	21393	83.5%

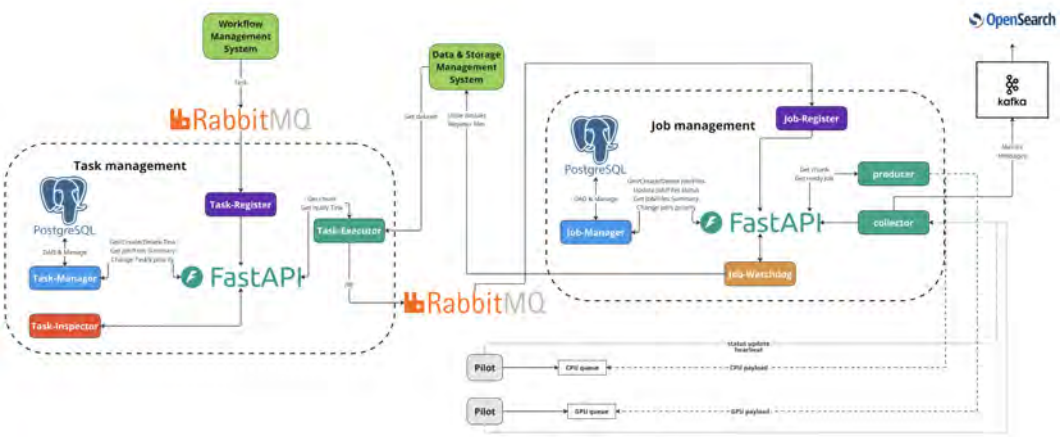
avg obtained - completed

job_id	avg time
263280	8.03 s
263279	8.01 s
263278	7.89 s
263277	7.87 s
263276	7.84 s
263275	7.87 s
263274	7.90 s

avg time in state

state	avg time
obtained	2.75 s
pre_processing	4.85 s
stage_in	2.23 s
running	2.45 s
finished	1.12 s
post_processing	5.64 s
stage_out	6.05 s

WMS Architecture



SPD Online Filter Middleware

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.