Scalable Orchestration and Dynamic Modeling of Distributed Workloads in SPD Online Filter

Nikita Greben

MLIT JINR

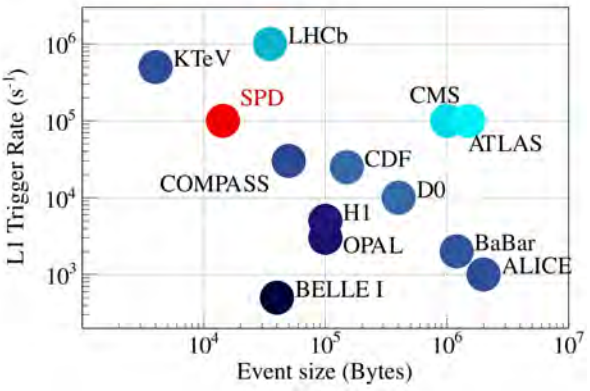
Tashkent, Uzbekistan, 22 October 2026





SPD at the NICA collider

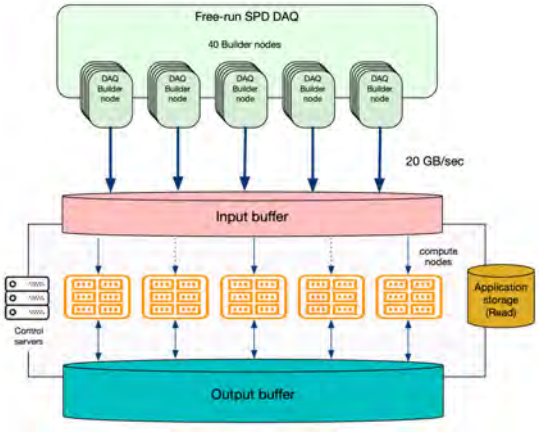
- Polarized protons and deuterons: the spin structure of the nucleon
- Bunches cross every 76.3 ns (13 MHz); ∼3 MHz of collisions
- 20 GB/s, or 200 PB a year; worth keeping: ∼10 PB
- Selection needs tracks and vertices: **no simple trigger**



The SPD Online Filter

The first stage after the DAQ: 200 PB → ∼10 PB a year

- No trigger. The DAQ writes **time slices**, not events
- Physics software rebuilds events, keeps ∼1 in 20
- Hardware: compute nodes between two disk buffers
- Middleware, **our part**: what runs, where and when



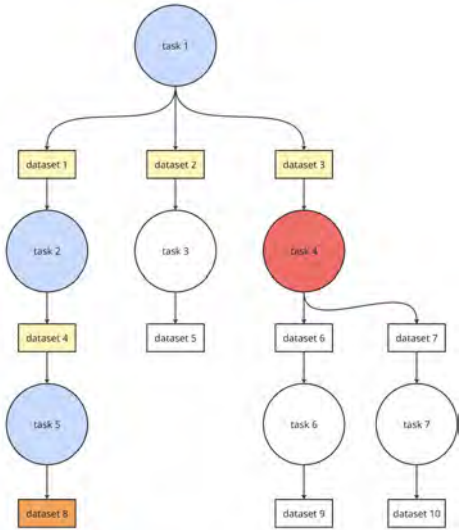
The main goal of SPD Online Filter Middleware

Goal

Keep the **disk buffers of raw DAQ data as small as possible** at all times.

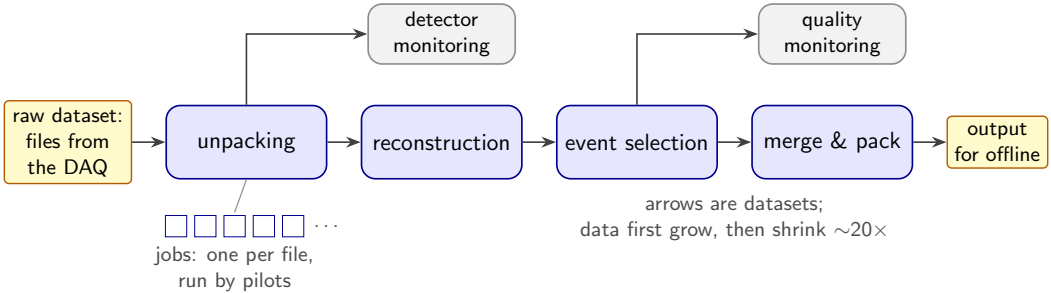
To do so the middleware must execute an arbitrary number of concurrent workflows in the most efficient way. This is a major scheduling problem.

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*.



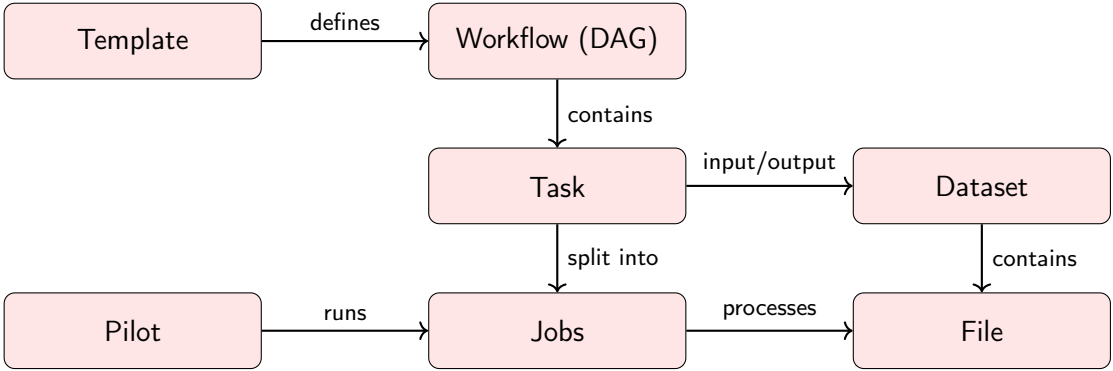
What the middleware runs: workflows

A raw dataset becomes a tree of steps, each run as many small jobs

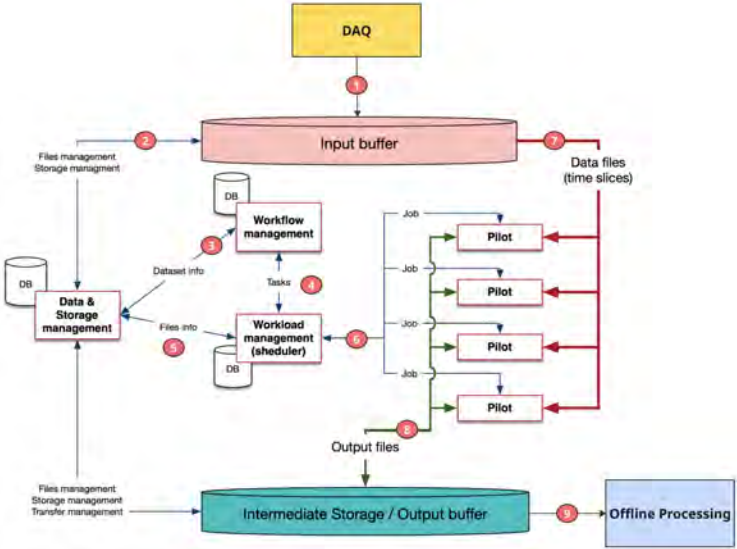


- **Pilot**: a program on a worker VM that pulls the next job

Core concepts

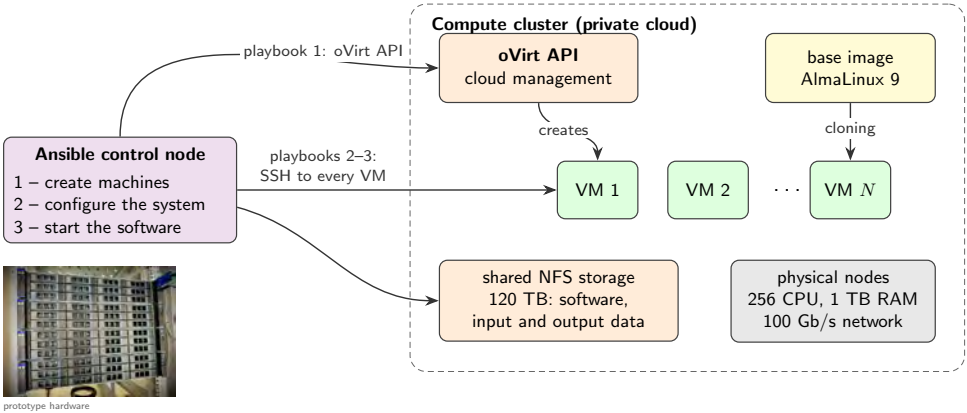


High-level architecture



The prototype cluster

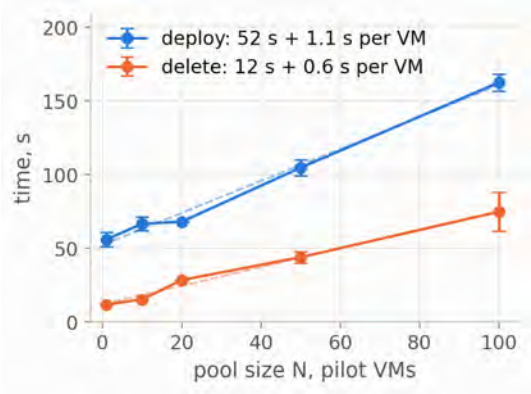
An oVirt-based private cloud



`pilot-ctl` (ours): one command builds or removes a pool of pilots.

It works

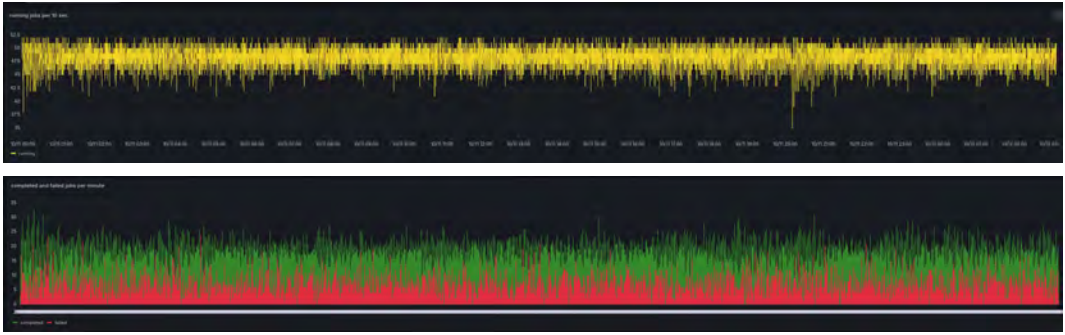
100 pilots in under 3 minutes



- It takes minutes from the empty cluster to the first job running.
- Mock jobs for now, SPD has no data yet

A test run on 52 pilots

All pilots busy around the clock

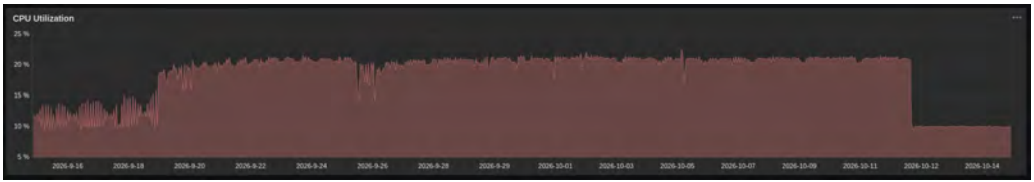


- Monitoring: Kafka → OpenSearch → Grafana
- 37 089 job runs, 30 889 completed on the real middleware and cluster

Three weeks of continuous test load

The test load filled 35 TB of the shared storage in 24 days

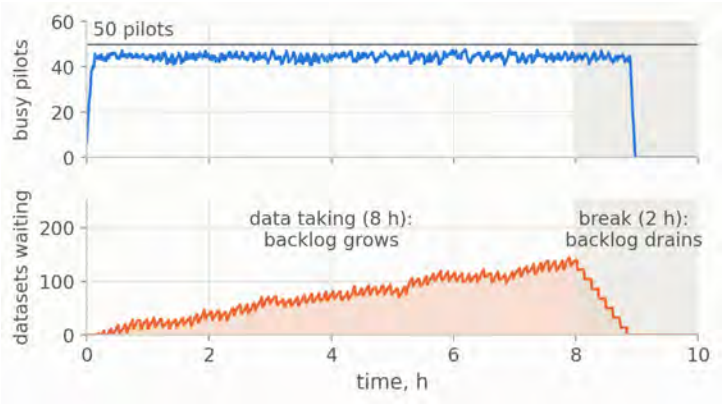




- Disk used: 1.5 → 35.4 TB from 18 September to 12 October, about 1.4 TB a day
- NFS storage CPU about 20 % while the test load runs, 10 % without

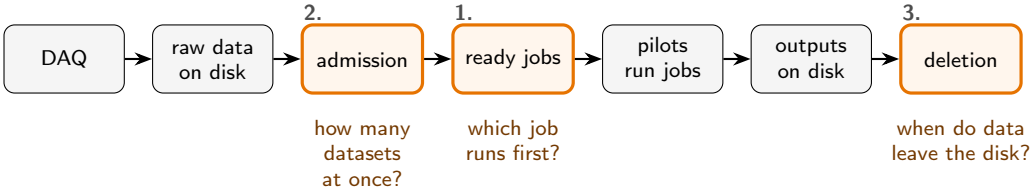
When the DAQ outruns the cluster

Data wait on disk, the scheduler decides who waits and what sits there



- Assumed cycle: 8 h of data taking, then a 2 h break
- Here data come 10 % faster than 50 pilots can process

Three questions



- We tested each one in a model of the new design

How we tested

What arrives

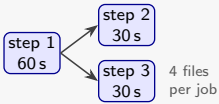


a **dataset**: 10 files × 512 MB

random (Poisson) arrival times, **8 h** of data taking, then a **2 h** break

1.0–1.1 × what the pilots can process: 155–170 datasets an hour

What runs: the new design



mock jobs of the acceptance test's template: fixed run time and output size

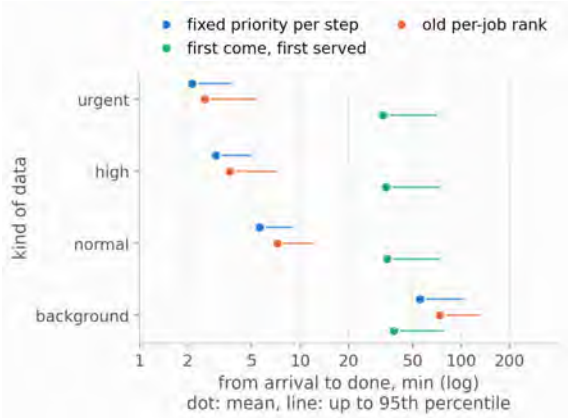
50 pilots; copying in and out: 1 s per 512 MB

What we measure

- time from arrival to done
- idle pilot time
- data on disk: still needed, or released and waiting for deletion

- Every policy gets the same arrivals and run times
- 20 simulated hours per run, 10–30 runs per point, ≈ 2 600 runs

1. Which job runs first?

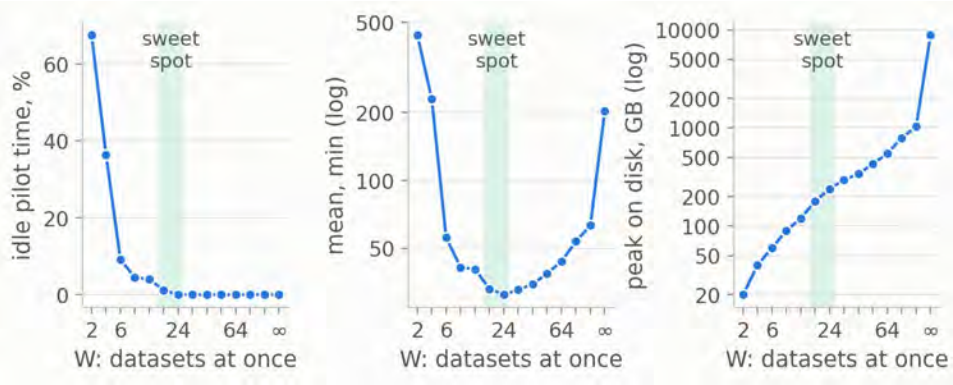


- Urgent to normal: minutes
- First come, first served: everyone takes ∼35 min
- Per-job priority based on age and failures: never better

Model · four classes, datasets of 30 files · data 10 % faster than processing · 30 runs per point

2. How many datasets at once?

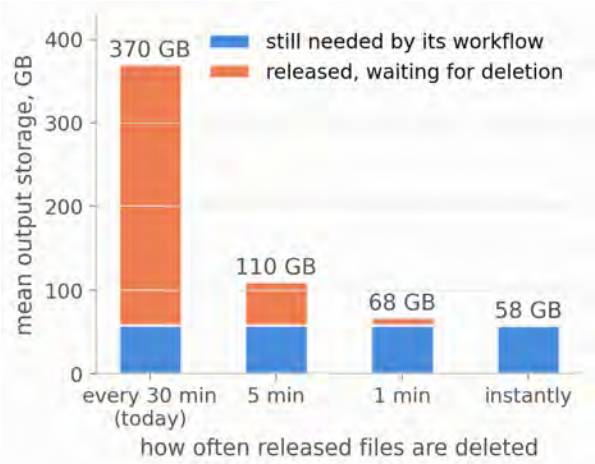
Sweet spot: 18-24 datasets in processing for 50 pilots



- No cap, new first steps overtake old last steps, old workflows stall

Model · test template · data 10 % faster than processing · deletion every minute · 10 runs per point

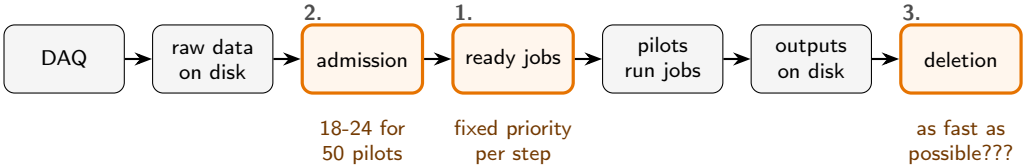
3. When do the data leave the disk?



- Deleted every 30 min, means 84 % is already released
- Every minute: 370 → 68 GB
- But if we delete faster, we might increase the NFS load.

Model · test template · data as fast as processing · 18 datasets at once · 20 runs per point

Conclusions



- The number of simultaneous workflows made an impact on the new design of the WfMS.
- The priority handling goes to the level of Tasks, not Jobs.
- There's a real trade-off between how fast we delete the data (increased NFS load) and how much released data we can keep in the system (increased disk usage).

Summary

- Achieved deployment of the entire SPD Online Filter Middleware on the cluster.
- Infrastructure as code and observability turned the middleware into a system we can rebuild, measure and model.
- The new design: static task weights for order, admission window W for the buffer.
- Next is to close the loop on the buffer level, and check the model on the stand, perform complex loading tests to see how the middleware behaves under different situations, and integrate the middleware with the applied software.

Thank you!

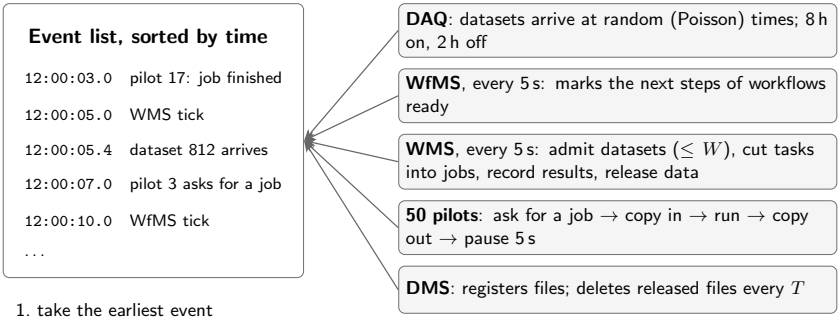
Backup

The model setup

	Test template	**Four classes**
Dataset	10 files × 512 MB	30 files of 3 / 1 / 0.3 / 0.05 GB: background, normal, high, urgent (40 / 35 / 20 / 5 %)
Workflow	test template of the acceptance test: step 1, 60 s (output = input) → step 2, 30 s (half the size), and step 3, 30 s, 4 files per job (half the size)	one step: 120 ± 30 / 60 ± 15 / 30 ± 8 / 15 ± 5 s; output 5 % of input
Failures	none	20 / 8 / 4 / 2 % per launch
Priorities	step weights 8 / 5 / 4	1 / 3 / 7 / 15
Arrivals	Poisson during 8 h of data taking, 2 h break; load 0.8, 1.0, 1.1 (125, 155, 170 datasets/h)	same; load 1.0, 1.1, 1.2
Datasets at once	$W = 2 \ldots \infty$	$W = 6$
Deletion	every 30 min; also 5, 1, 0 min	every 30 min
Runs	20 h, 20 runs per point	20 h, 30 runs per point

50 CPU pilots; scheduler ticks every 5 s, 50 jobs per tick; copying in and out 1 s per 512 MB; 5 s pilot pause between jobs; 3 launches per job (first try + 2 retries), a retry stays at the head of its task; deletion is instant when it runs. Every policy gets the same random arrivals, run times and failures.

How the event-driven simulation runs



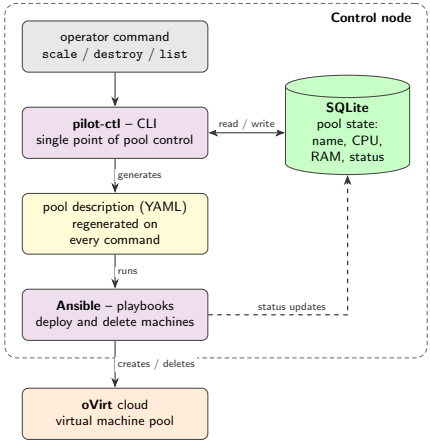
- One run: 20 simulated hours, about 640 000 events
- Every policy gets the same random arrivals, run times and failures

What the DAQ writes

- Time slices of 1 μs – 8.3 ms, packed into frames of up to 549 s and 16 GB
- A slice holds detector data blocks from several collisions
- The Online Filter rebuilds events from them



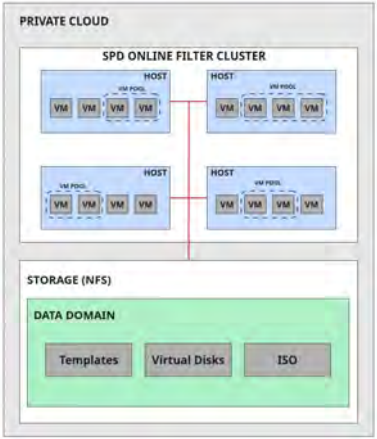
Managing the pilot pool with `pilot-ctl`



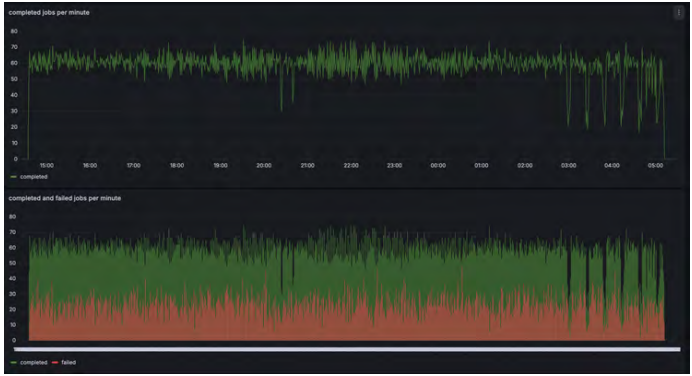
- Ansible alone keeps no state: it does not know which VMs exist
- `pilot-ctl` owns the pool state (SQLite) and regenerates the Ansible inputs on every command
- Measured: deploy 52 s per VM, delete 12 s per VM

The cluster layout

Hosts run VMs, pilot VMs form pools, NFS holds data, templates and disks

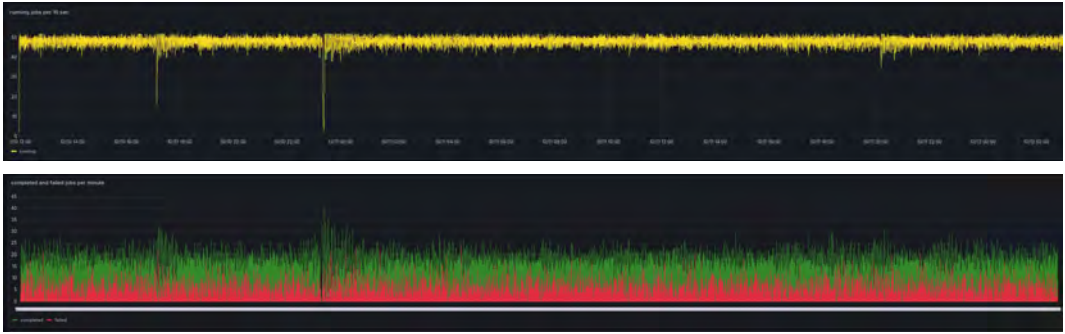


An earlier 15-hour test run



- 52 012 jobs completed, 20 588 attempts failed; each job gets 3 attempts

The 50-pilot run, wider window



Simple flow estimate

$$\frac{dQ}{dt} = \lambda(t) - \mu, \quad Q \geq 0, \qquad \lambda(t) = \begin{cases} \lambda, & \text{data taking, } 8\,\text{h} \\ 0, & \text{break, } 2\,\text{h} \end{cases}$$

$$Q_{n+1} = \max\bigl(0,\ Q_n + (\lambda(t_n) - \mu)\,\Delta t\bigr), \qquad Q_0 = 0, \quad \Delta t = 0.01\,\text{h}$$

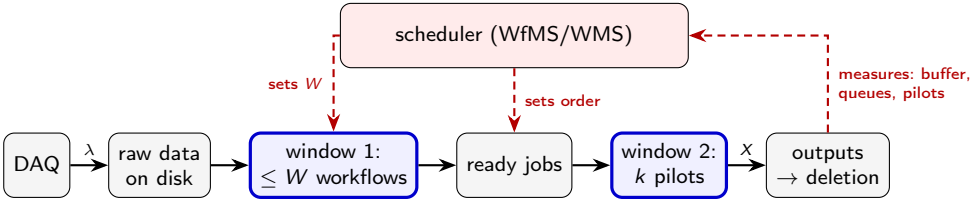
$$8\,\text{h} \cdot (\lambda - \mu) \ \leq\ 2\,\text{h} \cdot \mu \quad \Longleftrightarrow \quad \mu \geq 0.8\,\lambda$$

Q	data waiting on disk	minutes of data taking
λ	data rate of the detector	1 (the other rates relative to it)
μ	what the pilots process	0.7, 0.9, 1.0 (the three curves)
$\rho = \lambda/\mu$	load of the pilots while taking data	1.1 in our model, so $\mu \approx 0.91\,\lambda$

Model assumptions

- Normal operation only: no DMS/database outages, pilot losses, manual cancels.
- Job time and output size as in the stand's mock job; stage-in/out 1 s per 512 MB.
- Datasets of 10 files $\times$ 512 MB, Poisson arrivals while taking data.
- DMS deletion instant when it runs
- All CPU pilots

An open system with a control loop



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 a set of working nodes: currently multi-CPU, GPU accelerated nodes are planned for the next phase.

Middleware component

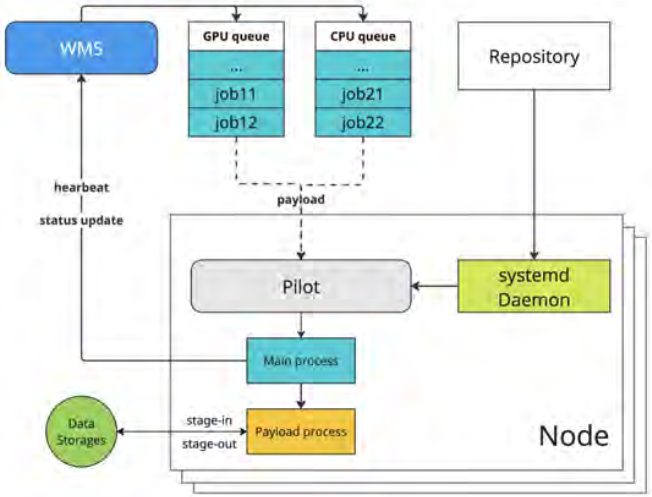
Software complex for reliable management of multistep data processing and efficient loading of the computing facility.

Applied software

Performs informational processing of data: unscrambling, reconstruction, etc.

Pilot architecture

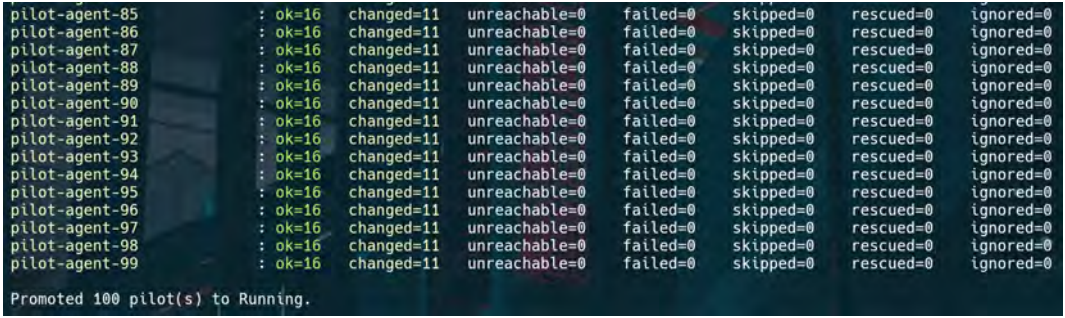
On each compute node a lightweight agent which runs the actual applied software.



Example: pilot-ctl scale 100



pilot-ctl result: 100 compute nodes created and launched

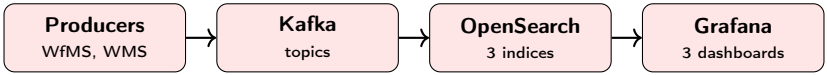


3 minutes starting from VM creation to running state on all 100 compute nodes.

oVirt monitor: ≈ 100 active compute nodes

		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%

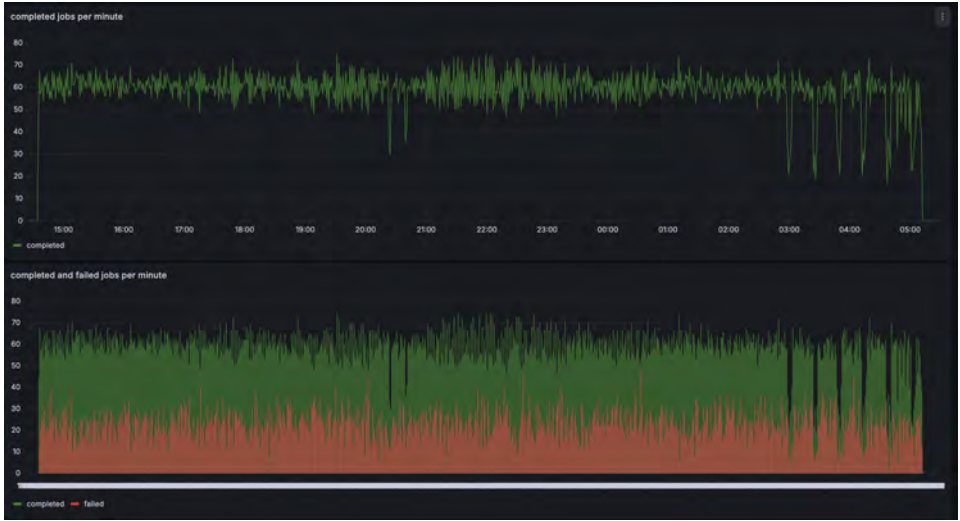
Observability: Kafka → OpenSearch → Grafana



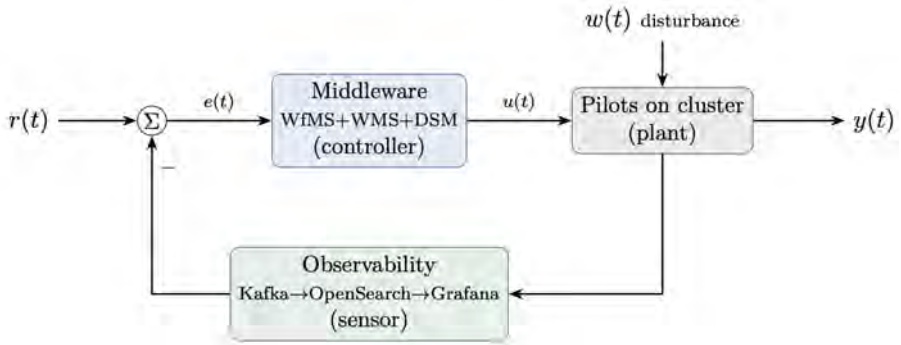
What gets collected:

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



The middleware as a closed loop controller

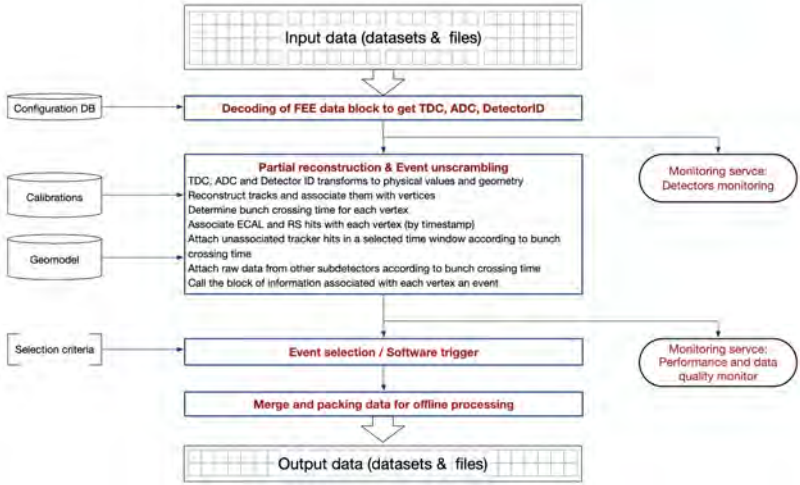


About diagram. $r(t)$ is workflow goals (*e.g.* throughput target, storage usage); $u(t)$ the action: tasks, jobs, and ranks dispatched; $y(t)$ telemetry observed (heartbeats, state transitions, queue depth); $w(t)$ the disturbance: pilot failures, network etc. The controller acts on the error $e(t)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

A control-theoretic view

- The middleware can be viewed as a closed loop dynamical system.
- The deployment pipeline for compute nodes offer an opportunity to study complex dynamics and scheduling policies with respect to the number of computte nodes.
- We can use control-theoretic approach to study the system when it scales with number of pilots and jobs, study stability and design a more unified and rigorous scheduler.

Data processing workflow in SPD Online Filter

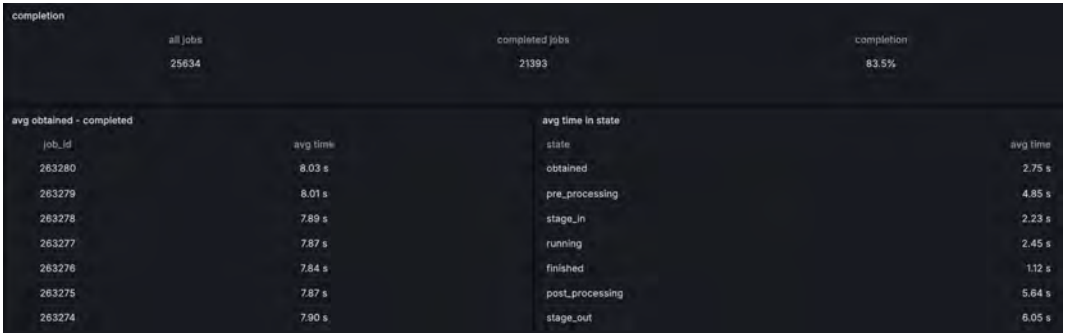


Core systems

- **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.



Jobs dashboards example



completion

all jobs	completed jobs	completion
25634	21393	83.5%

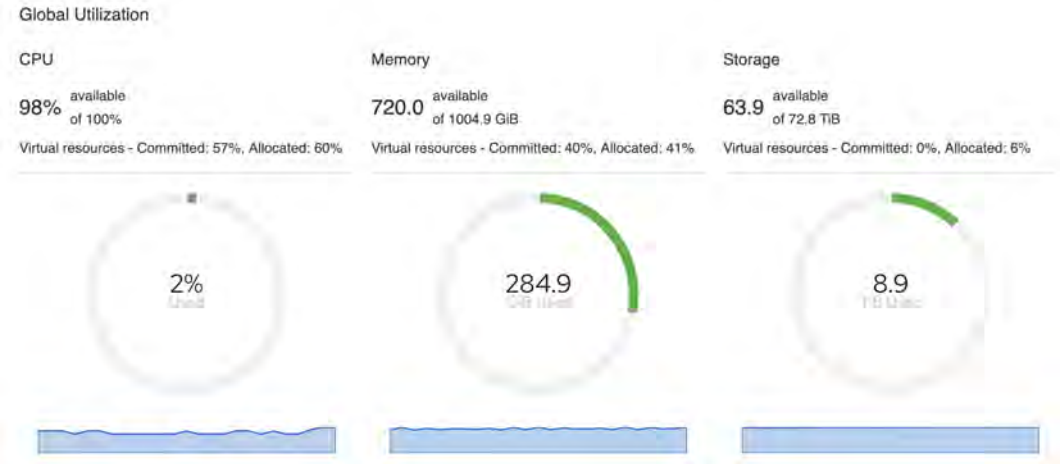
avg obtained - completed

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

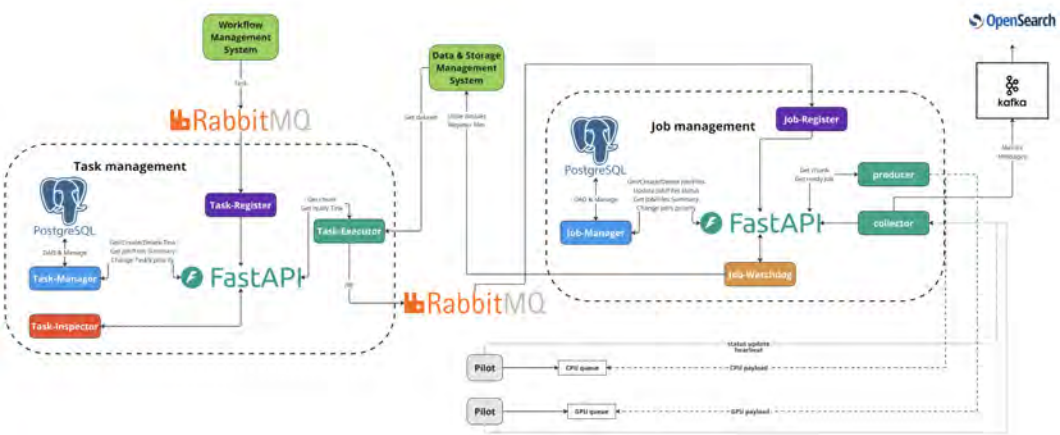
avg time in state

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

oVirt usage


Global Utilization
CPU
98% available
of 100%
Virtual resources - Committed: 57%, Allocated: 60%
2%
Memory
720.0 available
of 1004.9 GiB
Virtual resources - Committed: 40%, Allocated: 41%
284.9
Storage
63.9 available
of 72.8 TiB
Virtual resources - Committed: 0%, Allocated: 6%
8.9

WMS Architecture



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, reachability, sensitivity to parameter drift.

Note

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. Stability of the rank dynamics

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

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

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 $w(t)$

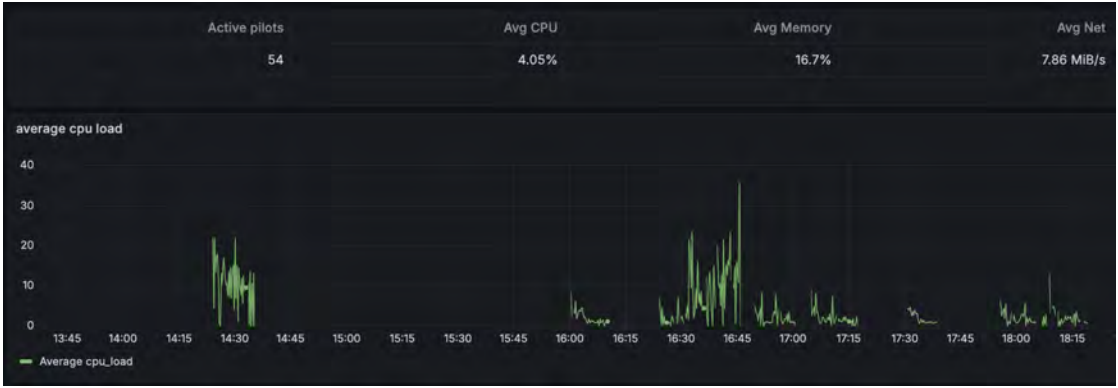
pilot failures, network and storage latency, etc.

Output $y(t)$

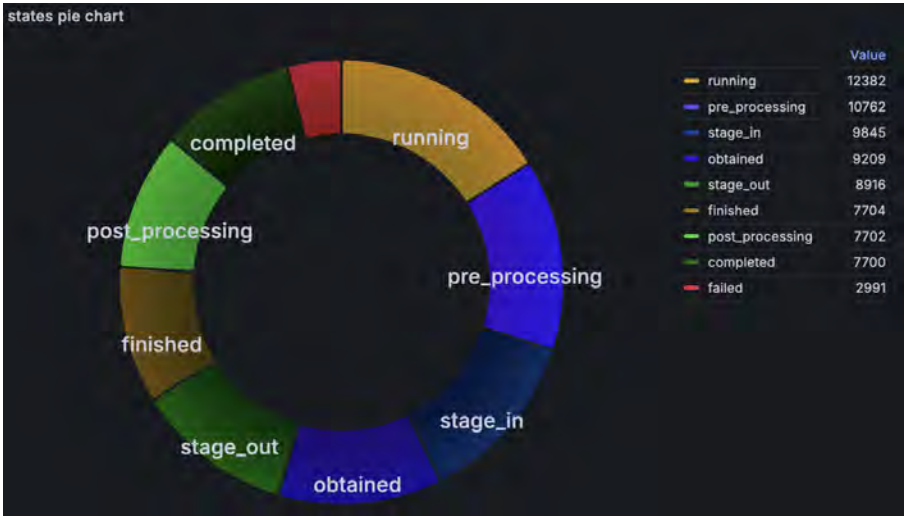
Heartbeats, state transitions, queue telemetry, basicaly the Grafana dashboards from the previous slides.

Fluid approximation. With $N \gg 1$ jobs, we'll try to treat this discrete problem as a continuous flow.

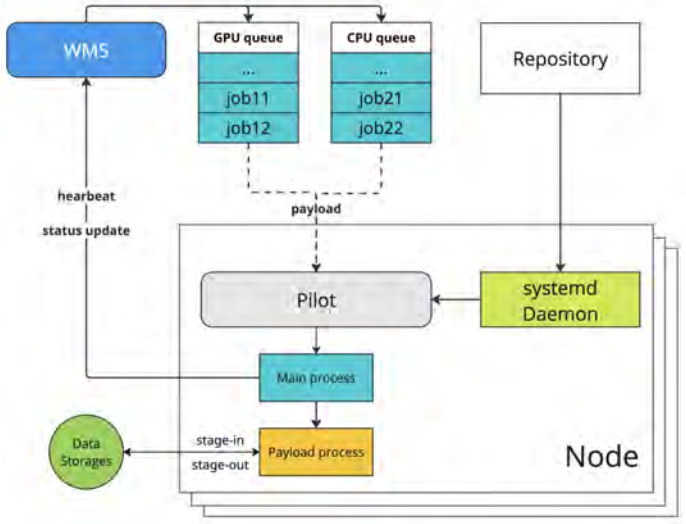
Pilots dashboards example



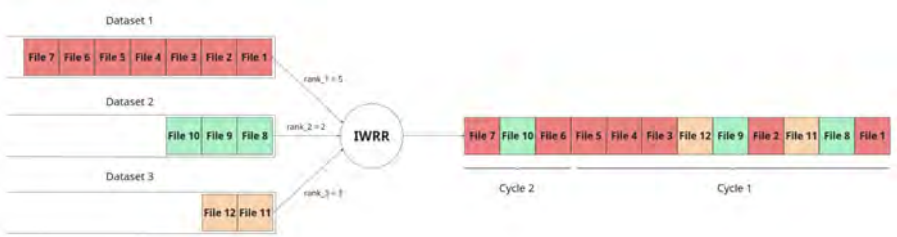
Jobs dashboards example: states pie chart



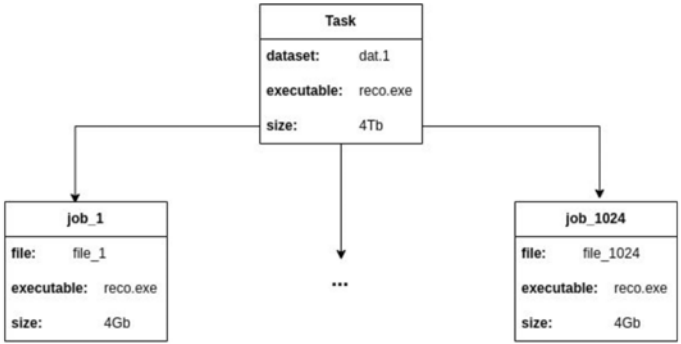
Pilot architecture



IWRR


Dataset 1
File 7
File 6
File 5
File 4
File 3
File 2
File 1
Dataset 2
File 10
File 9
File 8
Dataset 3
File 12
File 11
rank_1 = 5
rank_2 = 2
rank_3 = 1
IWRR
File 7
File 10
File 6
File 5
File 4
File 3
File 12
File 9
File 2
File 11
File 8
File 1
Cycle 2
Cycle 1

Task-jobs relationships



Data processing workflow

