SPD Online Filter Prototype

Current Status

Nikita Greben

MLIT JINR

XII SPD Collaboration Meeting, Dubna, 20 November 2026





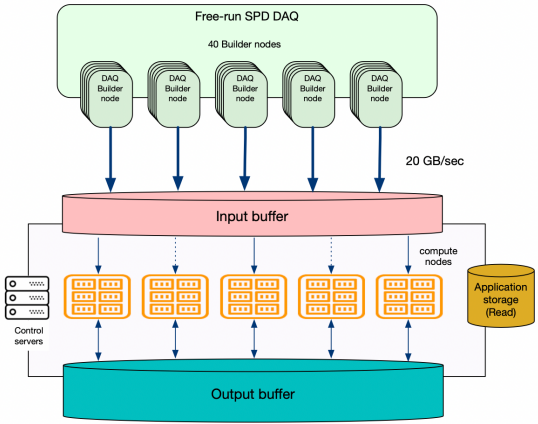
Outline

1. SPD Online Filter reminder
2. Progress on our cluster prototype
3. Comprehensive rework of the workload and workflow management systems
4. Monitoring system
5. Future plans

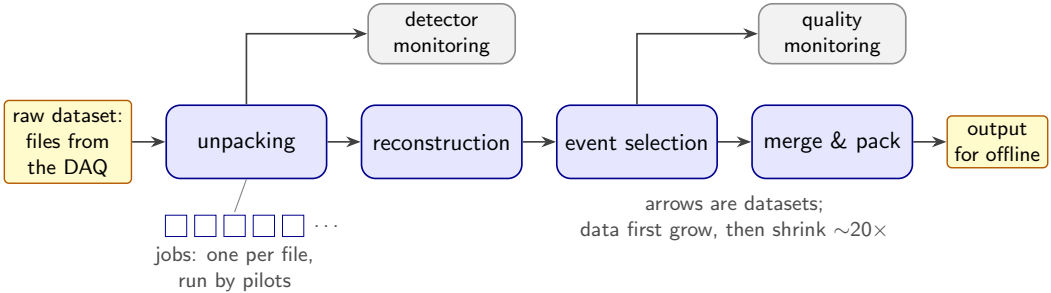
The SPD Online Filter

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

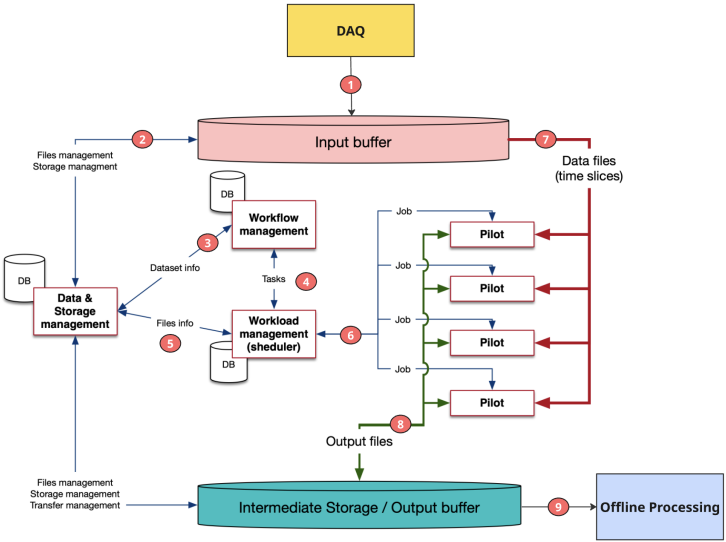


What the middleware runs: workflows



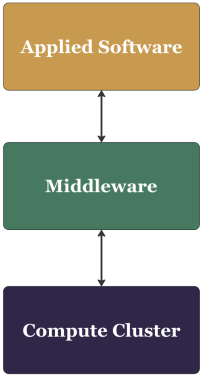
- **Pilot**: a program on a worker VM that pulls the next job

SPD Online Filter Middleware: Reminder

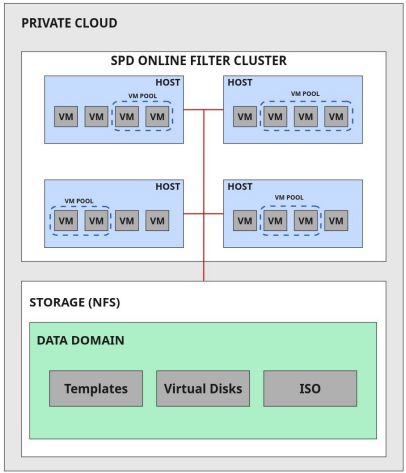


Core systems

- **Data Management System**
  - Data lifecycle support (data catalog, consistency check, cleanup, storage);
  - Details and current status in the next talk by **Polina Korshunova**.
- **Workflow Management System**
  - Define and execute data processing chains by generating the required number of computational tasks;
  - **We will be back to it later...**
- **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, etc.
  - **Also we'll be back to it...**



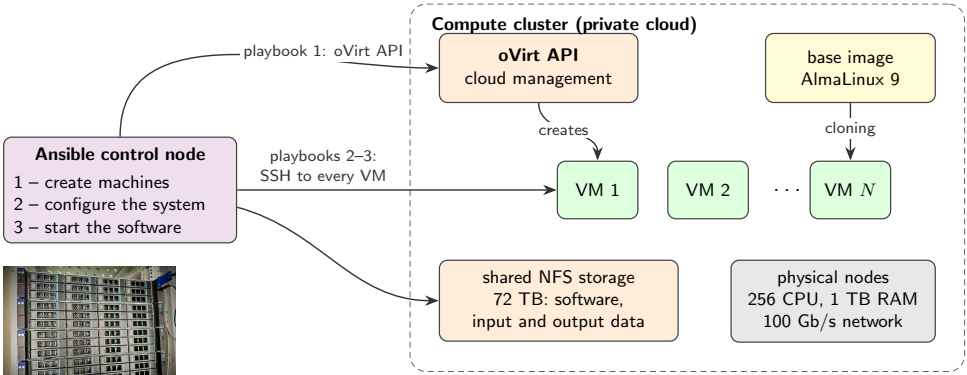
Prototype cluster layout



- **Four servers** with the same specs. One also hosts the shared storage.
- **Per server.** Two AMD EPYC 7513 CPUs, 64 physical cores (128 threads), 256 GB RAM and 2 TB NVMe.
- **Network.** Two 10 Gb/s links per server, and 80 Gb/s in total.
- **Shared storage.** Six 18 TB SAS drives in RAID 6 provide 72 TB of usable space.

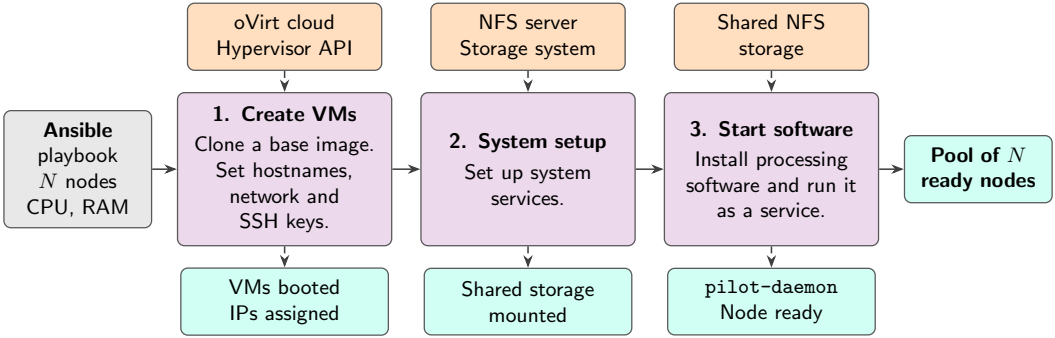
The prototype cluster

An oVirt-based private cloud (thanks to Andrey Kiryanov)



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

Provisioning a worker pool with Ansible

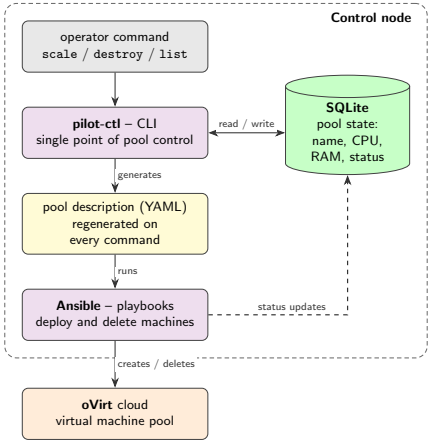


Encountered issues

- 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.
- Oudated documentation for ovirt.ovirt.ovirt_vm module in Ansible.

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)**.

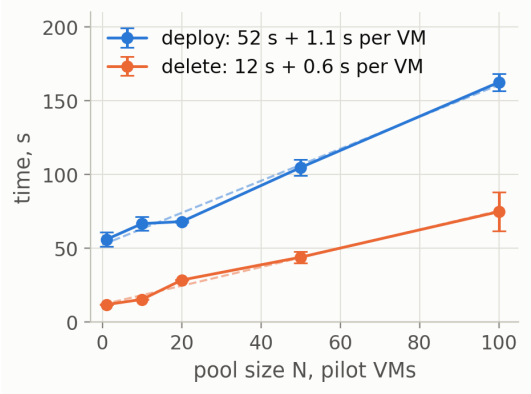
Managing the pilot pool with `pilot-ctl`



- 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

It works

100 pilots in under 3 minutes



- It takes minutes from the empty cluster to the first job running.
- Mock jobs for now: setup the resource usage, time of execution, variance for a size of an output files.

Example: pilot-ctl scale 100

```
nick@localhost.localdomain ~/pilot-ctl/pilot-control-cli git:(main) (2m 55.33s)
./pilot-ctl scale --count 100 --cpu 1 --ram 3GiB

Scaling: count=100 cpu=1 ram=3GiB
Wrote ansible/vms_definition.yml (100 pilots).
Vault password:
[WARNING]: provided hosts list is empty, only localhost is available. N

PLAY [Deploy Test VMs from Golden Image] ****************************

TASK [Authenticate to oVirt] ****************************************
ok: [localhost]
```

pilot-ctl result: 100 compute nodes created and launched

```
pilot-agent-85  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-86  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-87  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-88  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-89  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-90  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-91  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-92  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-93  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-94  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-95  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-96  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-97  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-98  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
pilot-agent-99  : ok=16  changed=11  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0

Promoted 100 pilot(s) to Running.
```

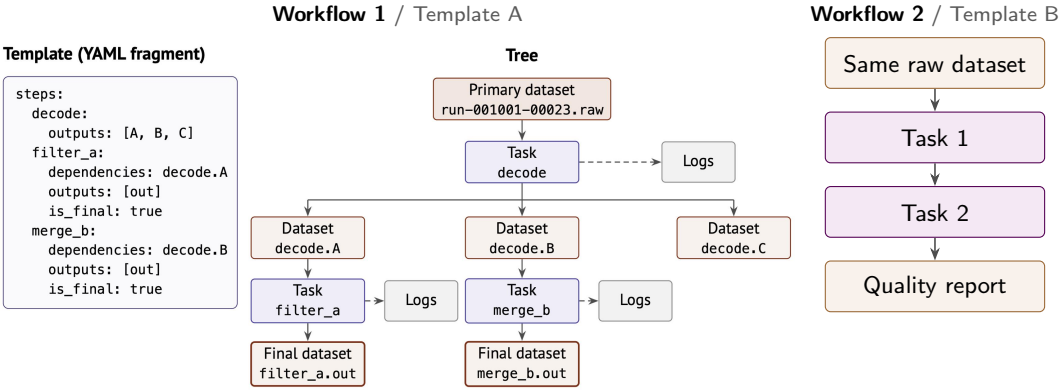
3 minutes starting from VM creation to running state on all 100 compute nodes.

oVirt monitor: ≈ 100 active compute nodes

		Name	Com	Host	IP Addresses	FQDN	Cluster	Data Cen	Memory	CPU
		AlmaLinux10.1-te		sof-srv1.jinr.ru	10.17.245.40 fe80::546f:e0ff:fe38:0	ansible	SOF	JINR	12%	0%
		daq-generator		sof-srv4.jinr.ru	10.17.245.45 fe80::546f:e0ff:fe38:4e	daq	SOF	JINR	14%	0%
		dms-dev		sof-srv1.jinr.ru	10.17.245.49 fe80::546f:e0ff:fe38:5a	dms-dev	SOF	JINR	12%	0%
		HostedEngine		sof-srv4.jinr.ru	10.17.245.31 fe80::216:3eff:fe52:c95b	sof-virt.jinr.ru	SOF	JINR	43%	2%
		pilot-agent-257		sof-srv1.jinr.ru	10.17.245.248 fe80::546f:e0ff:fe38:1c	pilot-agent-257	SOF	JINR	20%	26%
		pilot-agent-258		sof-srv1.jinr.ru	10.17.245.177 fe80::546f:e0ff:fe38:1d	pilot-agent-258	SOF	JINR	20%	19%
		pilot-agent-259		sof-srv1.jinr.ru	10.17.245.127 fe80::546f:e0ff:fe38:1e	pilot-agent-259	SOF	JINR	21%	22%
		pilot-agent-280		sof-srv3.jinr.ru	10.17.245.210 fe80::546f:e0ff:fe38:1f	pilot-agent-280	SOF	JINR	20%	29%
		pilot-agent-281		sof-srv3.jinr.ru	10.17.245.236 fe80::546f:e0ff:fe38:20	pilot-agent-281	SOF	JINR	20%	23%
		pilot-agent-282		sof-srv2.jinr.ru	10.17.245.168 fe80::546f:e0ff:fe38:21	pilot-agent-282	SOF	JINR	20%	19%
		pilot-agent-283		sof-srv3.jinr.ru	10.17.245.148 fe80::546f:e0ff:fe38:22	pilot-agent-283	SOF	JINR	20%	24%
		pilot-agent-284		sof-srv2.jinr.ru	10.17.245.78 fe80::546f:e0ff:fe38:23	pilot-agent-284	SOF	JINR	20%	18%
		pilot-agent-285		sof-srv3.jinr.ru	10.17.245.216 fe80::546f:e0ff:fe38:24	pilot-agent-285	SOF	JINR	20%	23%
		pilot-agent-286		sof-srv3.jinr.ru	10.17.245.235 fe80::546f:e0ff:fe38:25	pilot-agent-286	SOF	JINR	20%	27%
		pilot-agent-287		sof-srv3.jinr.ru	10.17.245.138 fe80::546f:e0ff:fe38:26	pilot-agent-287	SOF	JINR	20%	23%
		pilot-agent-288		sof-srv3.jinr.ru	10.17.245.81 fe80::546f:e0ff:fe38:27	pilot-agent-288	SOF	JINR	20%	19%
		pilot-agent-289		sof-srv3.jinr.ru	10.17.245.202 fe80::546f:e0ff:fe38:28	pilot-agent-289	SOF	JINR	20%	20%
		pilot-agent-290		sof-srv4.jinr.ru	10.17.245.163 fe80::546f:e0ff:fe38:29	pilot-agent-290	SOF	JINR	20%	16%
		pilot-agent-291		sof-srv4.jinr.ru	10.17.245.136 fe80::546f:e0ff:fe38:2a	pilot-agent-291	SOF	JINR	20%	25%

Back to middleware...

In short, the middleware creates one workflow per matching template

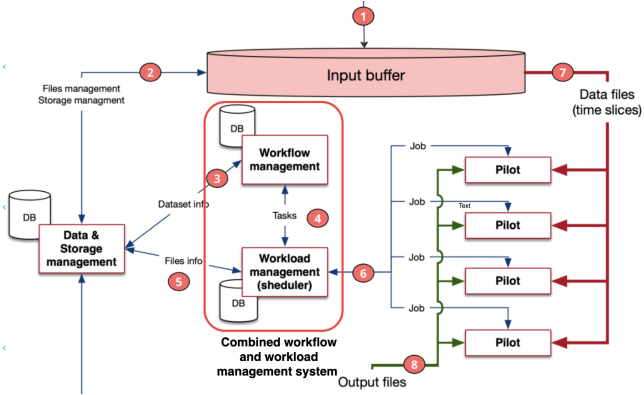


The middleware must execute an arbitrary number of concurrent workflows efficiently and take into account the buffer occupancy: what task to execute now and what not.

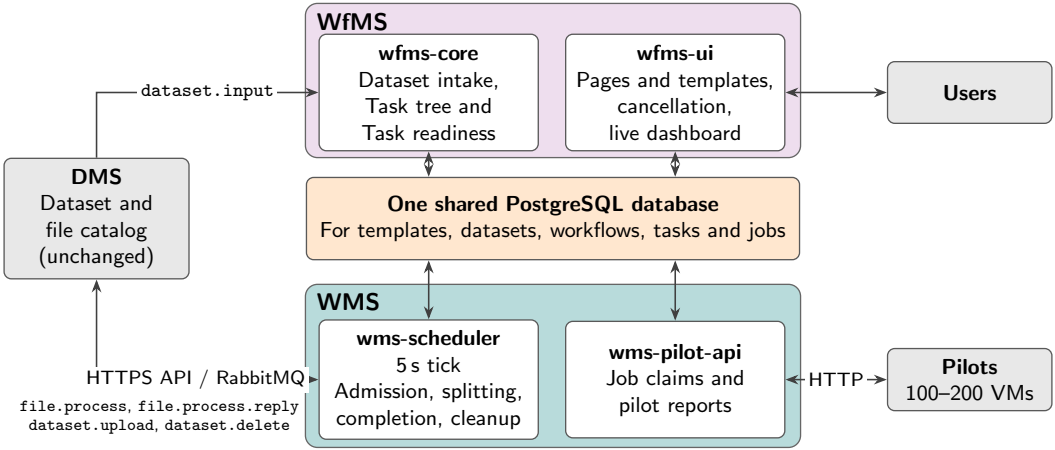
This is a major scheduling problem and should be performed in one place.

Moving all brains to just one place

- Long tests exposed issues with high asynchronisity and eventual consistency.
- Scheduling decisions are linked across workflows, tasks and jobs.
- Simulations showed that limiting workflow admission affects buffer use and waiting times.
- The new design coordinates admission and job dispatch within one scheduling architecture.
- Rework in progress with master's student **Boris Dvizov**.

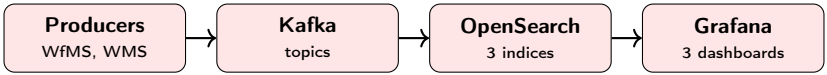


Target WfMS and WMS architecture (we need a new name)



Monitoring System

Ongoing work with a MC's student **Mikhail Medvedev**



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`, ...

Moving time series data to ClickHouse



- Around **100 million** jobs metadata records moved to ClickHouse.
- Some queries against the document-oriented OpenSearch database took too long to run, due to the nature of these analytical queries
- Initial results look promising6, time-series queries run pretty quickly in ClickHouse (OLAP), builds our dashboards in mere seconds.

Comparing a job lifecycle duration query

Run (Cmd+Enter) ✓ Logs Metrics 9 rows in result, 1.77 sec. 511.96 MB RAM Read 84.51 M rows, 1.44 GB (47.82 M/sec, 812.15 MB/sec)

№	n	metric	avg_sec	median_sec	jobs
1	1	obtained	1.1232904894311018	1.120597	1 484 877
2	2	pre_processing	1.1310573296524342	1.114754	1 484 876
3	3	stage_in	1.1291651000069576	1.1116925	847 961
4	4	running	119.23683701966316	119.0286775	847 963
5	5	finished	1.1123838473927017	1.111786	706 480
6	6	post_processing	1.1122877310274466	1.111739	706 481
7	7	stage_out	1.1123322402035976	1.111788	706 484
8	8	obtained → completed	125.92762391354209	125.68567949999999	696 239
9	9	obtained → failed	25.48083045203126	2.2379085	788 638

- Around 85 million job records.
- ClickHouse used about 500 MB of memory; OpenSearch used about 4 GB.
- The OpenSearch query did not finish after using all available VM memory.

A test run on 52 pilots

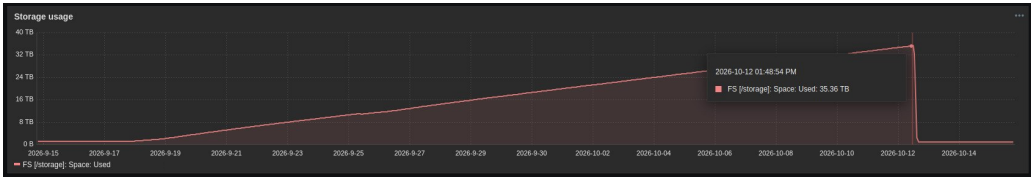
All pilots busy around the clock

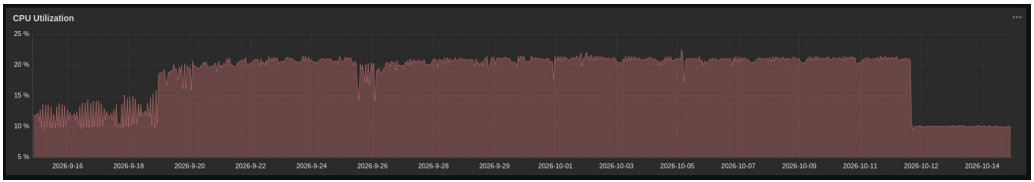


- Monitoring: Kafka → OpenSearch → Grafana
- 37 089 job runs, 30 889 completed on the real middleware and cluster

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

Summary

- Deployed the SPD Online Filter Middleware on the cluster.
- Infrastructure as code and observability make the middleware easier to rebuild, measure and model.
- Designed the combined workflow and workload management system and documented the technical design. Implementation is underway.
- Partially migrated to a dedicated PostgreSQL cluster supervised by a senior DBA.
- Ran simulations to study scheduling strategies. The updated strategies are awaiting implementation.

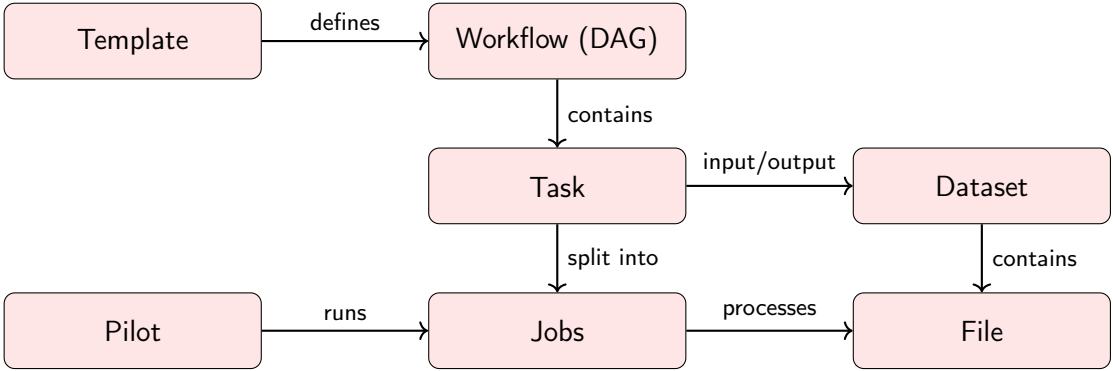
Next plans

- Continue the redesign and implementation of the workflow and workload management system.
- Enrich and further develop the monitoring system.
- Develop buffer aware feedback control and validate the model on the test stand.
- Run load tests to study middleware behavior under different conditions.

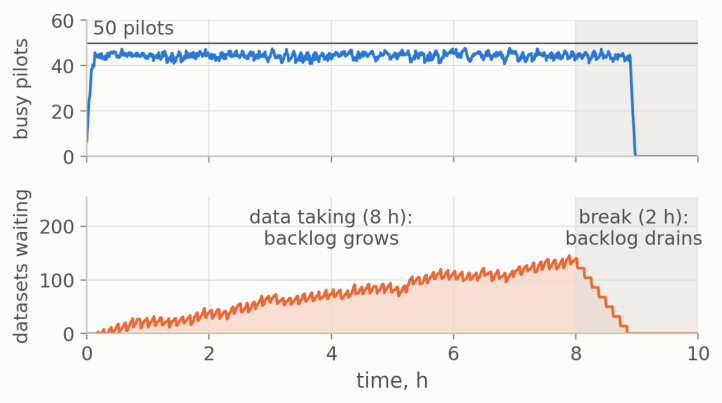
Thank you!

Backup

Core concepts

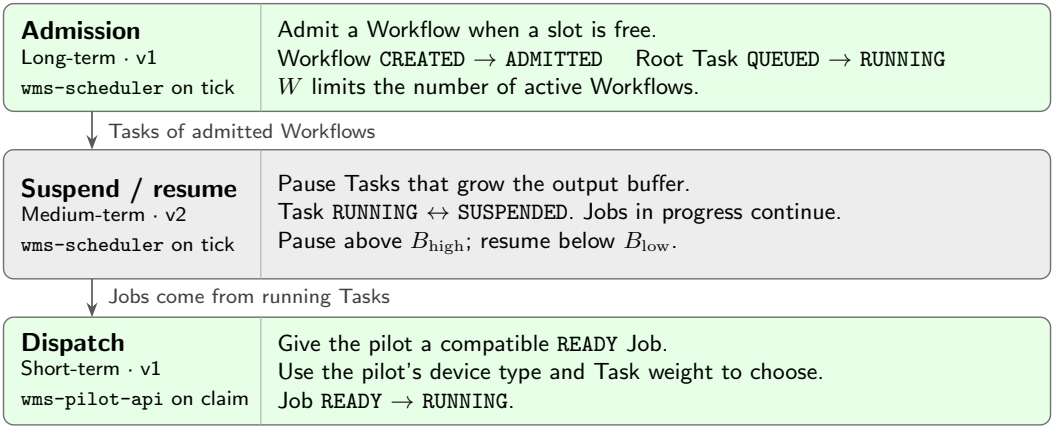


When the DAQ outruns the cluster



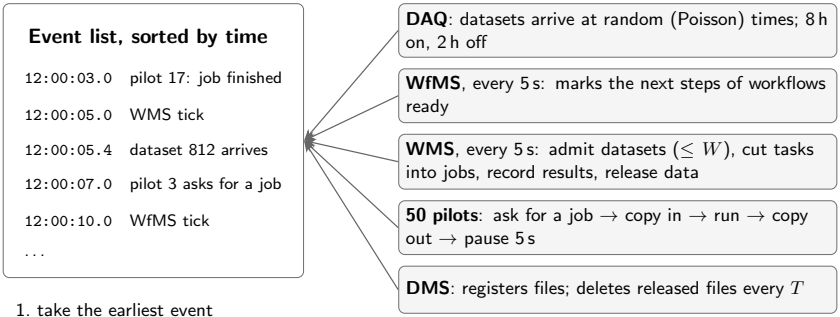
- Assumed cycle: 8 h of data taking, then a 2 h break
- Here data come 10 % faster than 50 pilots can process

Three levels of scheduling



Parameter values are still to be tuned (W, B_{high}, B_{low}).

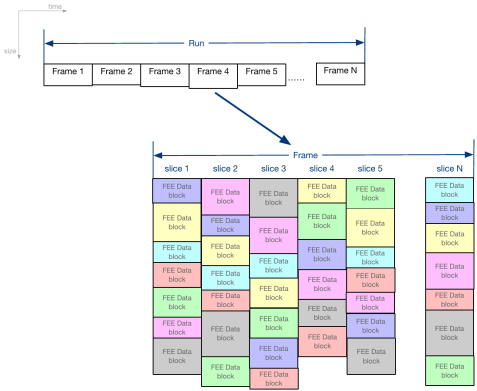
How the event-driven simulation runs



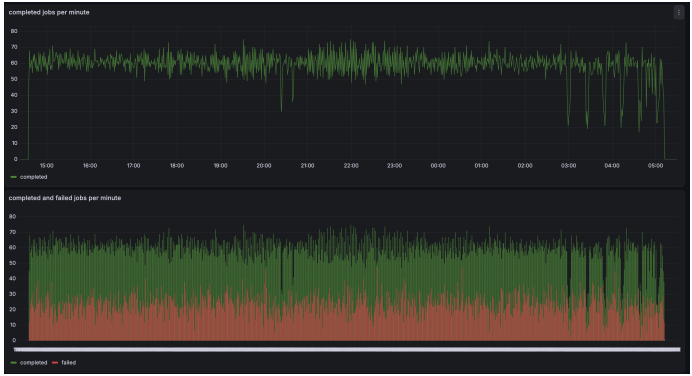
- One run: 20 simulated hours, about 640 000 events
- Every policy gets the same random arrivals, run times and failures

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



An earlier 15-hour test run



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

Simple flow estimate

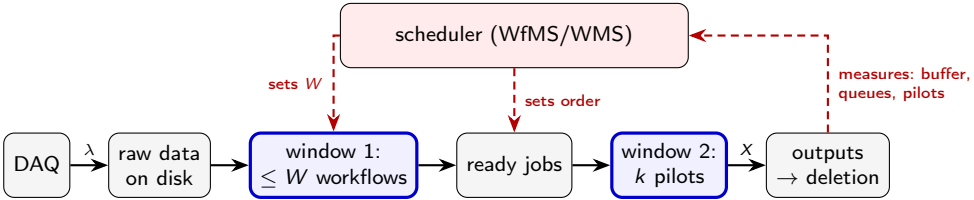
$$\frac{dQ}{dt} = \lambda(t) - \mu, \quad Q \geq 0, \qquad \lambda(t) = \begin{cases} \lambda, & \text{data taking, } 8\,\text{h} \\ 0, & \text{break, } 2\,\text{h} \end{cases}$$

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

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

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 $\mu \approx 0.91\,\lambda$

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

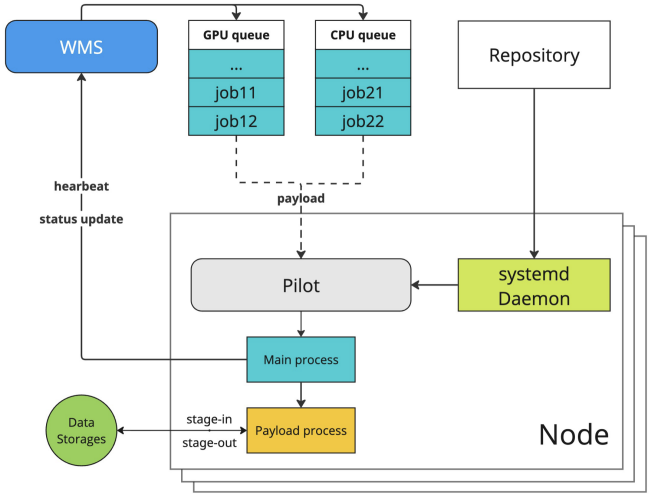
Software complex for reliable management of multistep data processing and efficient loading of the computing facility.

Applied software

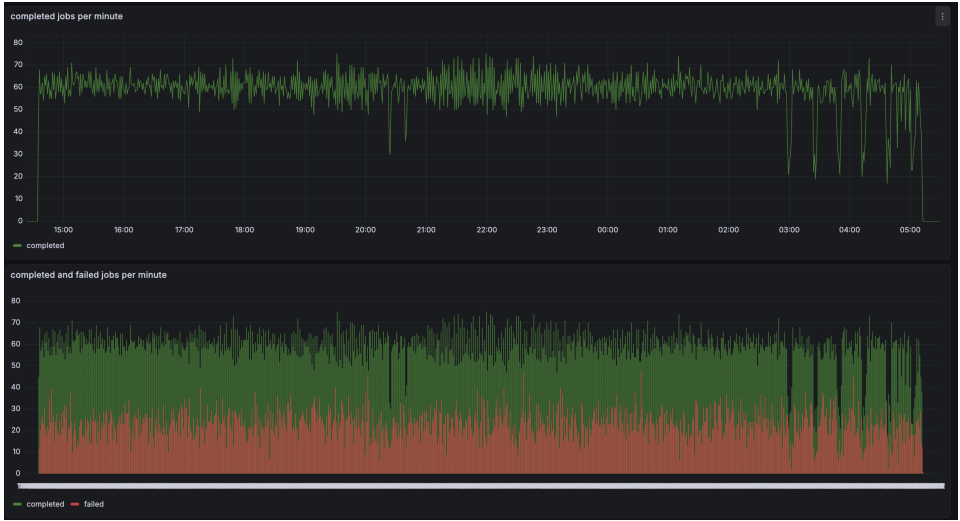
Performs informational processing of data: unscrambling, reconstruction, etc.

Pilot architecture

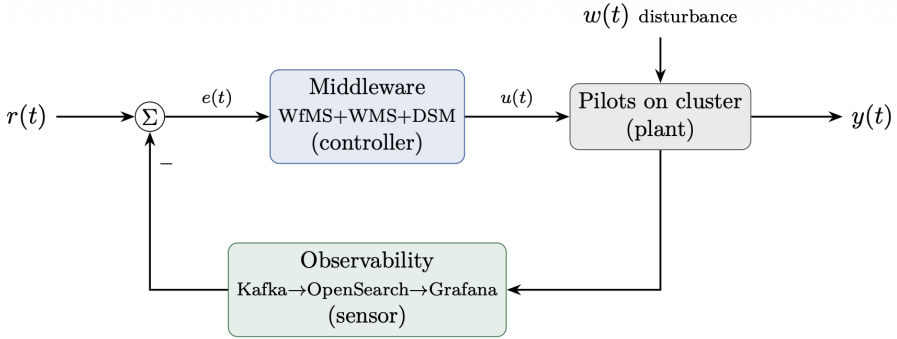
On each compute node a lightweight agent which runs the actual applied software.



Jobs dashboards example: jobs per minute

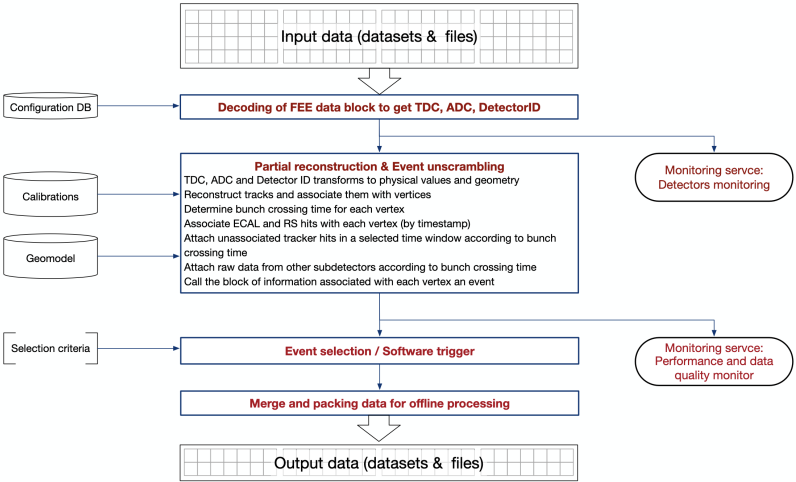


The middleware as a closed loop controller

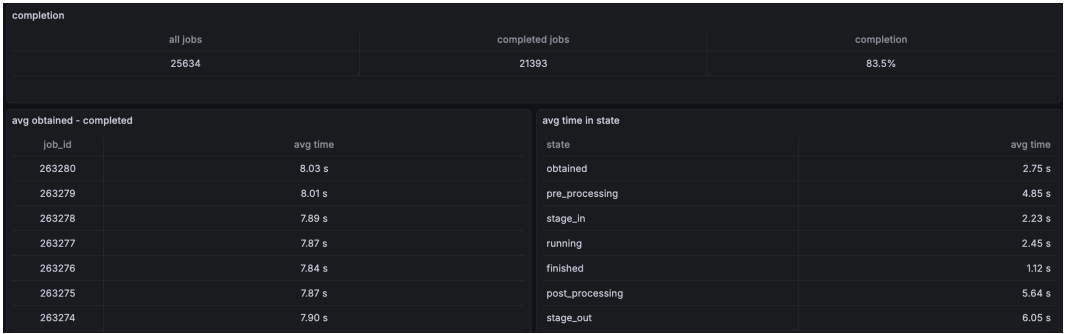


About diagram. $r(t)$ is workflow goals (*e.g.* throughput target, storage usage); $u(t)$ the action: tasks, jobs, and ranks dispatched; $y(t)$ telemetry observed (heartbeats, state transitions, queue depth); $w(t)$ the disturbance: pilot failures, network etc. The controller acts on the error $e(t)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

Data processing workflow in SPD Online Filter



Jobs dashboards example



completion

all jobs	completed jobs	completion
25634	21393	83.5%

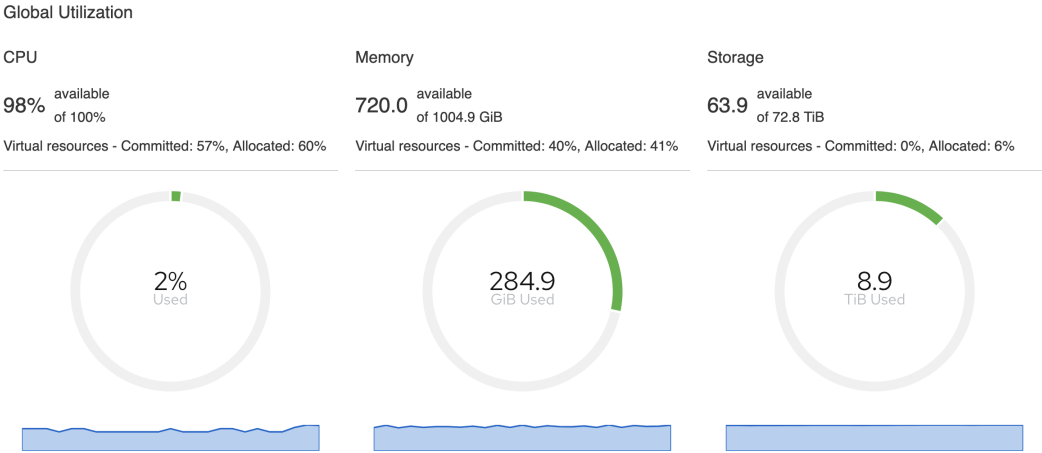
avg obtained - completed

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

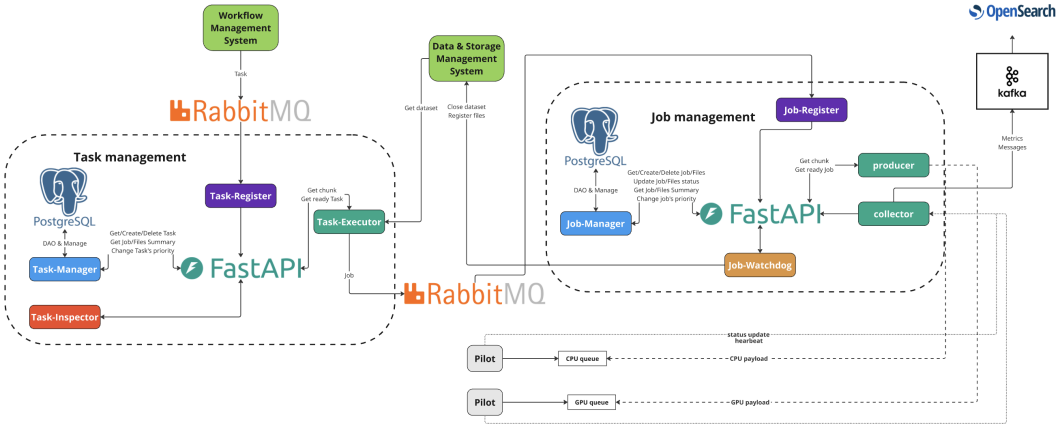
avg time in state

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% Used
Memory
720.0 available of 1004.9 GiB
Virtual resources - Committed: 40%, Allocated: 41%
284.9 GiB Used
Storage
63.9 available of 72.8 TiB
Virtual resources - Committed: 0%, Allocated: 6%
8.9 TiB Used

WMS Architecture



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 $w(t)$

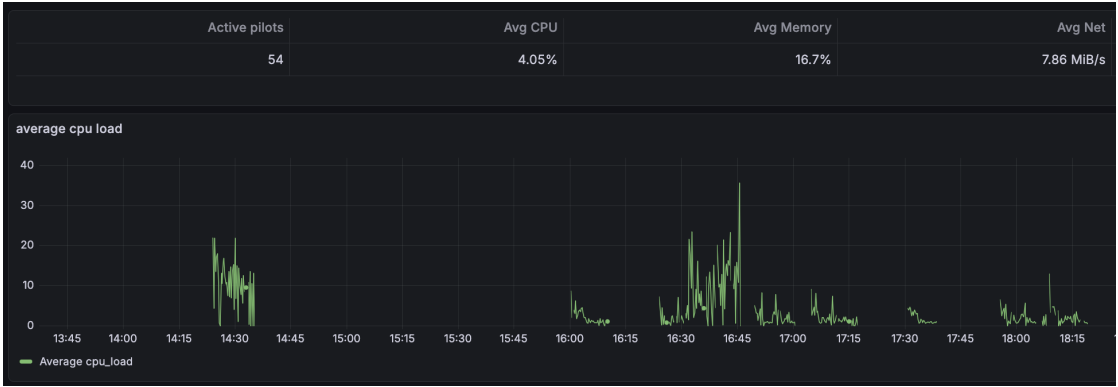
pilot failures, network and storage latency, etc.

Output $y(t)$

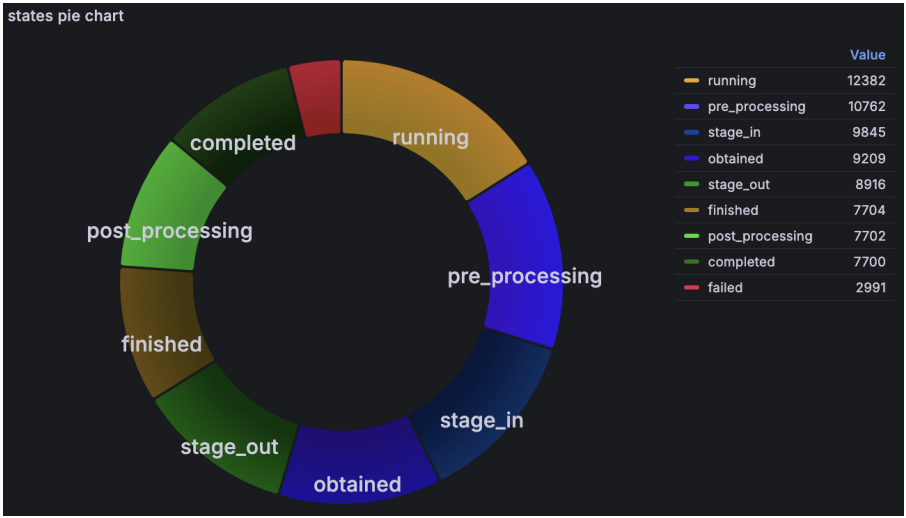
Heartbeats, state transitions, queue telemetry, basicaly the Grafana dashboards from the previous slides.

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

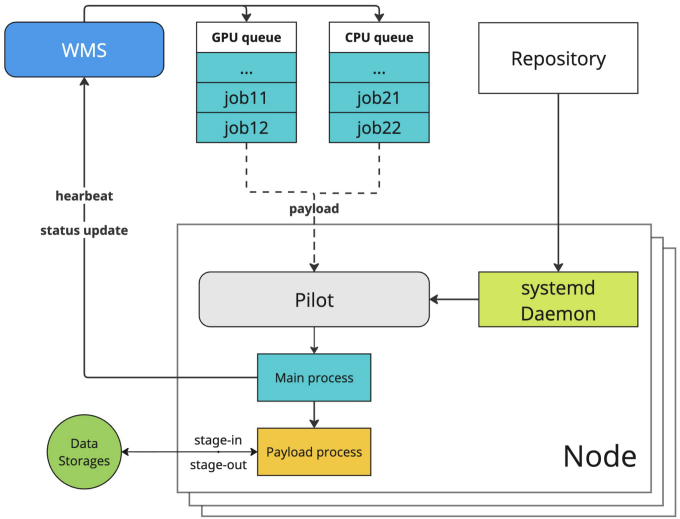
Pilots dashboards example



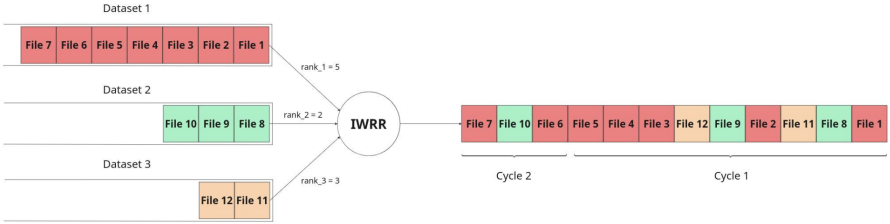
Jobs dashboards example: states pie chart



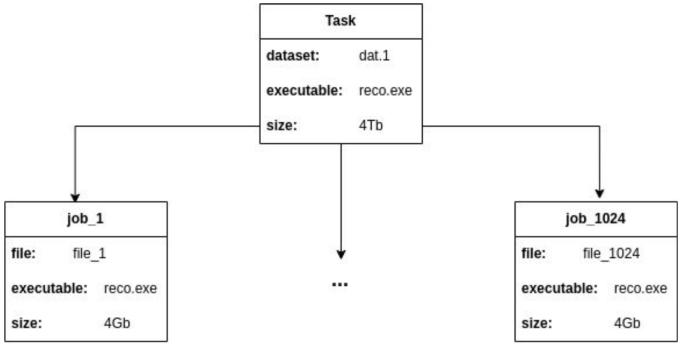
Pilot architecture



IWRR as a way to dispatch jobs



Task-jobs relationships



Data processing workflow

