Benchmarking massively-parallel Analysis Grand Challenge workflows using Snakemake and REANA

Marco Donadoni[1] Matthew Feickert[2] Alexander Held[2]
Andrii Povsten[3] Oksana Shadura[4] Tibor Simko[1]

[1]CERN [2]University of Wisconsin Madison (US)
[3]Princeton University (US) [4]University of Nebraska Lincoln (US)

27th Conference on Computing in High Energy and Nuclear Physics
October 21st-25th 2024, Krakow, Poland

IRIS-HEP

Institute for Research and Innovation in Software for High Energy Physics

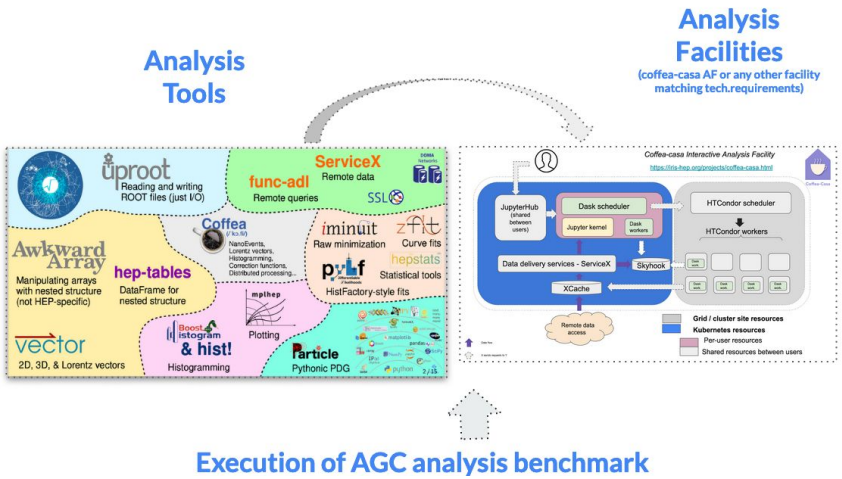
https://iris-hep.org/

Objective: Software R&D for HL-LHC

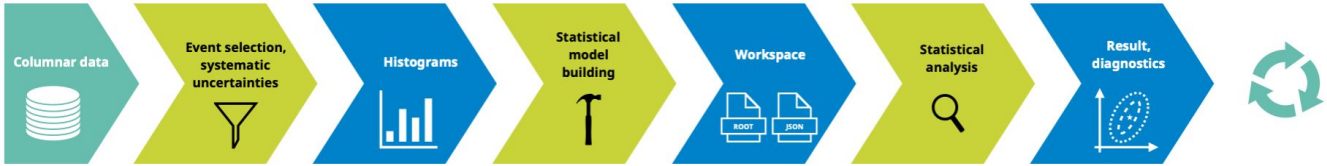
- Develop analysis tools
- Demonstrate execution at scale suitable for HL-LHC requirements
- Foster reproducibility and reuse

Analysis Grand Challenge

- Testbed for software library development
- Environment to prototype analysis workflows
- (Performance) testing of analysis facilities



Example: CMS ttbar analysis pipeline



AGC CMS ttbar cross-section measurements

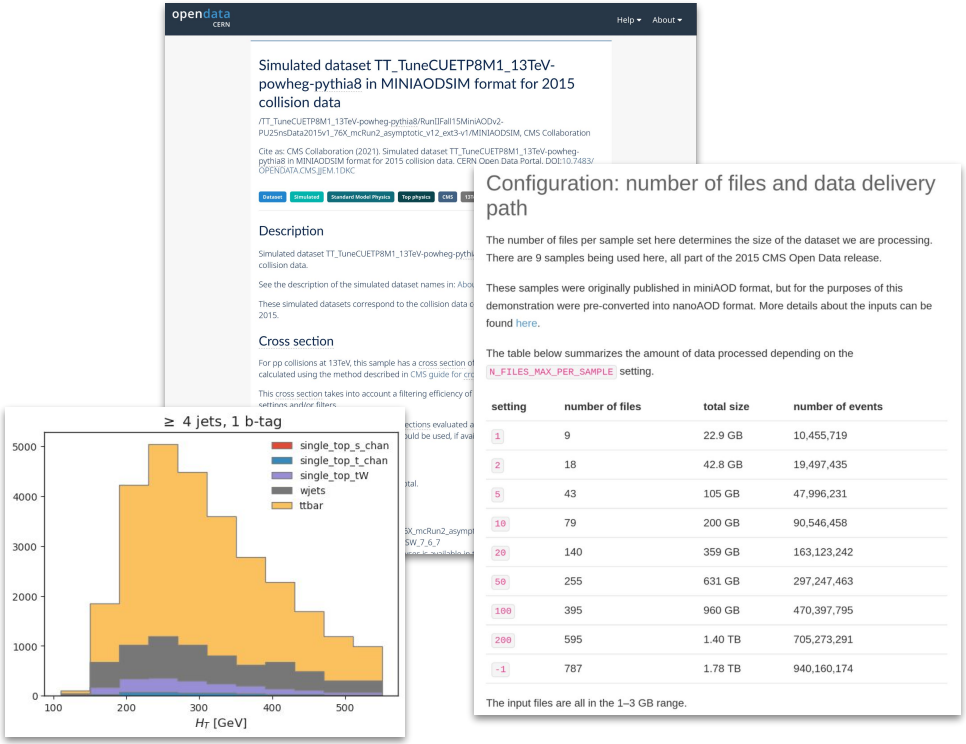
Chosen because of relevant analysis workflow aspects

Around 2TB of input NanoAOD from 2015 Run-2 CMS Open Data

Massively-parallel workflow, nearly 800 input files

To ease interactivity, one of the implementations is a Jupyter notebook using the coffea framework

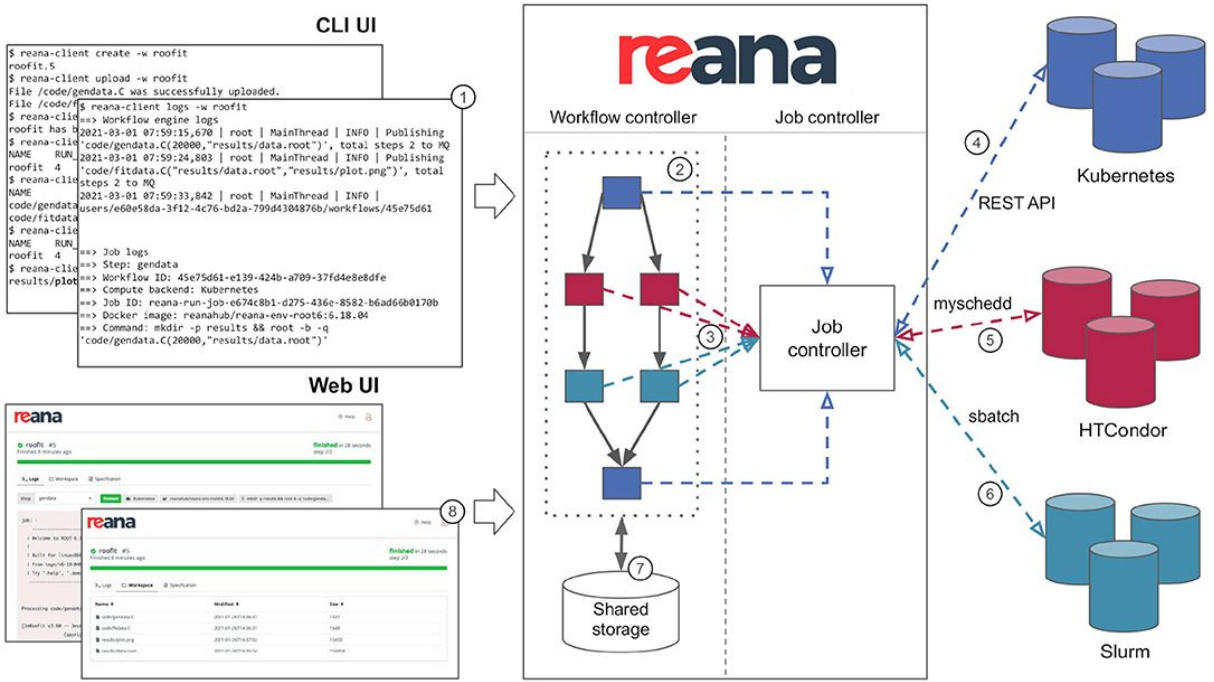
Can we scale-out on REANA?



https://github.com/iris-hep/analysis-grand-challenge

REANA

Running containerised analysis workflows on the cloud



Multiple **compute backends**:

- Kubernetes
- HTCondor
- Slurm

Multiple **workflow languages**:

- CWL
- Serial
- Snakemake
- Yadage

Multiple **means of use**:

- Command-line client
- Web UI

https://www.reana.io

Snakemake

Python-based workflow description language

Directed acyclic graph (DAG) of jobs is automatically constructed from provided rules

Docker container encapsulates all the needed dependencies

Notebooks are run parametrised via `papermill`

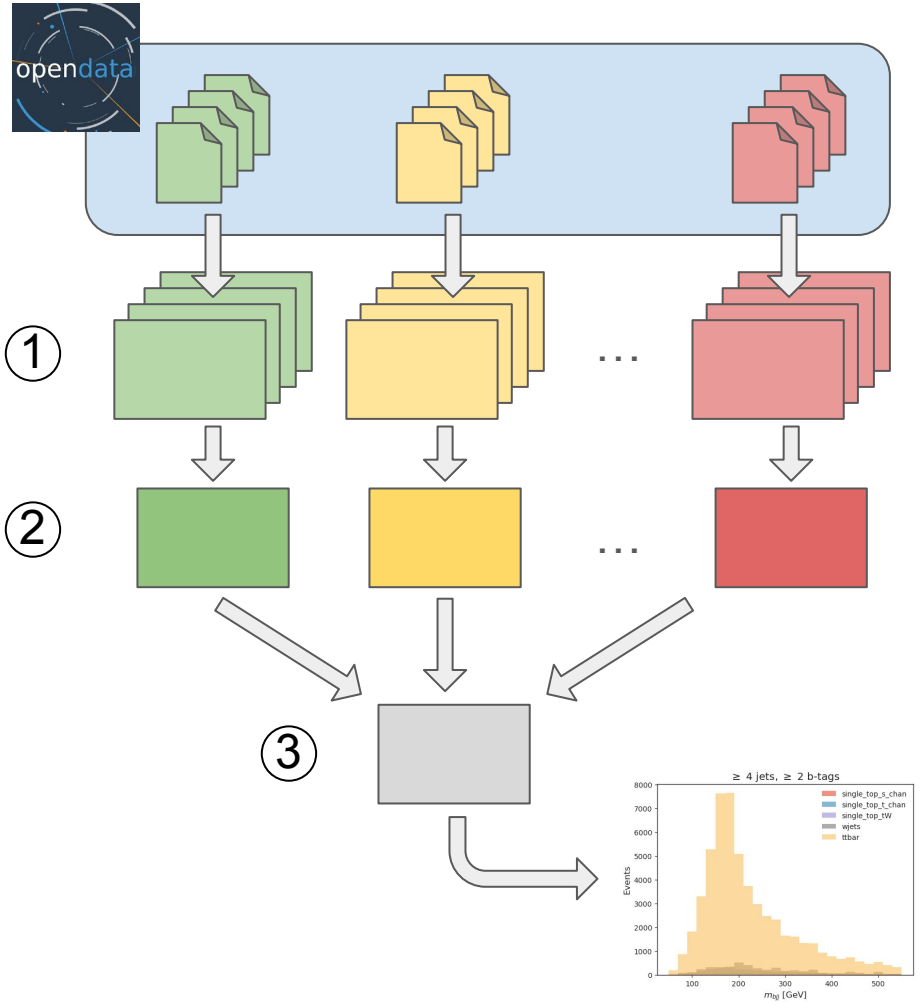
```
def get_file_paths(wildcards, max=N_FILES_MAX_PER_SAMPLE):
    "Return list of file paths for the given SAMPLE and CONDITION."
    filepaths = []
    ...

rule all:
    input:
        "histograms.root"

rule process_sample:
    container:
        "docker.io/reanahub/reana-demo-agc-cms-ttbar-coffea:1.0.0"
    resources:
        kubernetes_memory_limit ="1850Mi"
    input:
        get_file_paths,
        notebook="file_merging.ipynb"
    output:
        "everything_merged_{sample}__{condition}.root"
    params:
        sample_name = '{sample}__{condition}'
    shell:
        "/bin/bash -l && source fix-env.sh && "
        "papermill {input.notebook} "
        "merged_{params.sample_name}.ipynb "
        "-p sample_name {params.sample_name} -k python3"
```

https://snakemake.readthedocs.io

CMS ttbar analysis workflow

1. Processing one file at a time
   - one job per input file **(more than 700!)**
2. Merging all processed files of a given sample
   - one job per sample **(9)**
3. Merging all results from previous step
   - one single job **(1)**

All jobs are short-lasting (usually less than 1 minute each)



Snakemake scatter-gather paradigm

input files are not in input list; the notebook constructs remote URL from passed parameters (sample, condition)

```
rule process_sample_one_file_in_sample:
    container:
        "docker.io/reanahub/reana-demo-agc-cms-ttbar-coffea:1.0.0"
    resources:
        kubernetes_memory_limit="1850Mi"
    input:
        notebook="ttbar_analysis_reana.ipynb"
    output:
        "histograms/histograms_{sample}__{condition}__{index}.root"
    params:
        sample_name = "{sample}__{condition}"
    shell:
        "..."

rule process_sample:
    container:
        "docker.io/reanahub/reana-demo-agc-cms-ttbar-coffea:1.0.0"
    resources:
        kubernetes_memory_limit="1850Mi"
    input:
        get_file_paths,
        notebook="file_merging.ipynb"
    output:
        "everything_merged_{sample}__{condition}.root"
    params:
        sample_name = '{sample}__{condition}'
    shell:
        "..."
```

Python function that depends on wildcards (sample, condition)

automated multi-cascading scatter-gather thanks to declarative approach: each rule automatically creates as many jobs as needed, no need to manually create and dispatch every job

```
rule merging_histograms:
    container:

"docker.io/reanahub/reana-demo-agc-cms-ttbar-coffea:1.0.0"
    resources:
        kubernetes_memory_limit="1850Mi"
    input:
        "everything_merged_ttbar__nominal.root",
        "everything_merged_ttbar__ME_var.root",
        "everything_merged_ttbar__PS_var.root",
        "everything_merged_ttbar__scaleup.root",
        "everything_merged_ttbar__scaledown.root",
        "everything_merged_single_top_s_chan__nominal.root",
        "everything_merged_single_top_t_chan__nominal.root",
        "everything_merged_single_top_tW__nominal.root",
        "everything_merged_wjets__nominal.root",
        notebook="final_merging.ipynb"
    output:
        "histograms.root"
    shell:
        "..."

rule all:
    input:
        "histograms.root"
```

target rule

Setting up REANA test cluster

Kubernetes v1.30 cluster with 53 nodes

- one master node
- three REANA infrastructure nodes
  - web server, database, message broker
- one workflow orchestration node
  - Snakemake
- **48 job-running nodes**

Node flavour

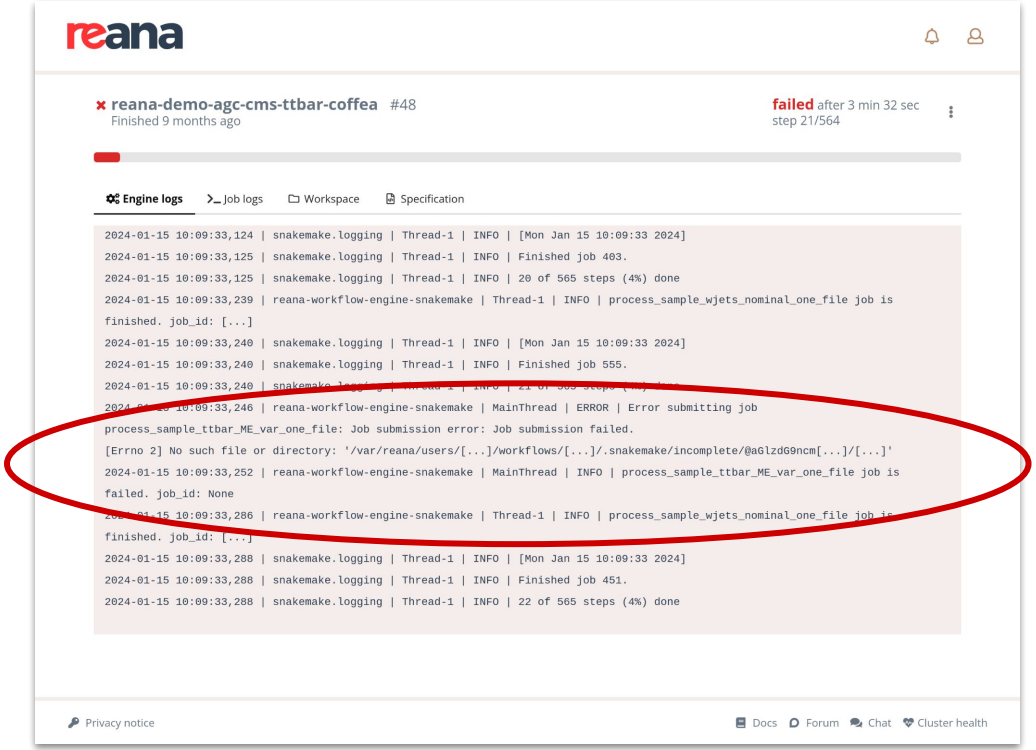
- 8 vCPUs
- 15 GB RAM

Episode 1: First run

Ready, set … Oops!

Workflow (intermittently) fails due to a bug in caching mechanism when handling deleted files, occurring when Snakemake manages many concurrent jobs

Lesson: highly concurrent workloads will uncover synchronization issues

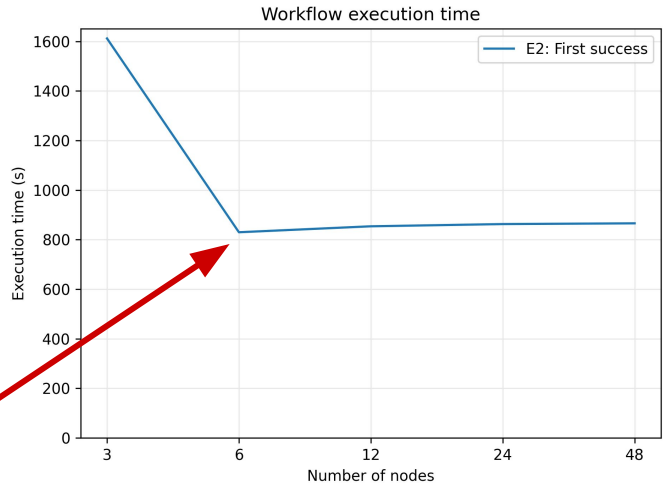


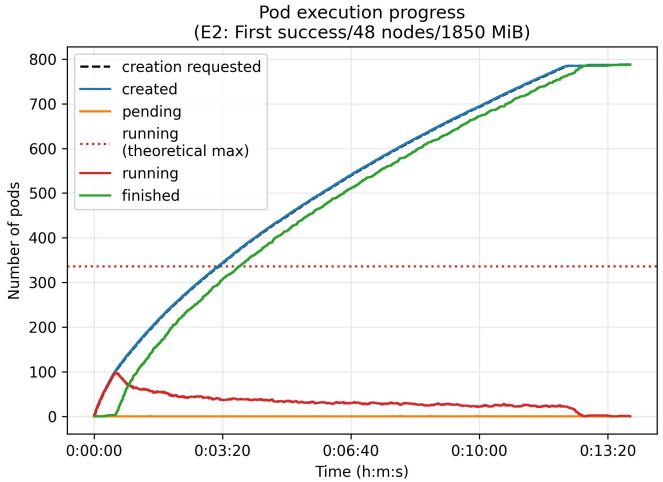
Episode 2: First success

After fixing the caching issue, the workflow execution succeeds on REANA 0.9.2

However:

- the workflow does not scale beyond six nodes
- bottleneck is the creation of jobs that gets slower and slower as more jobs are being created
- REANA 0.9.2 is not able to spawn jobs fast enough to fully utilise the cluster





Episode 3: Profiling job scheduling

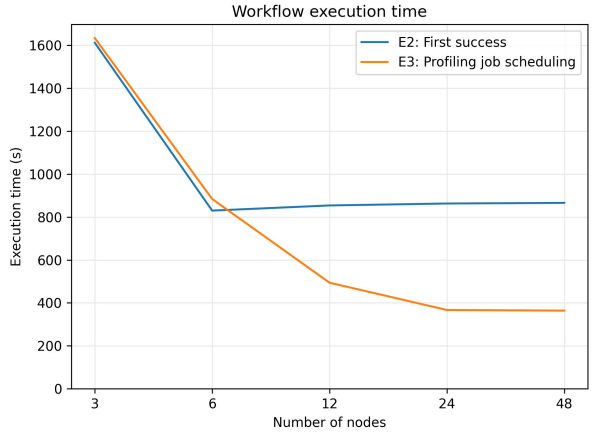
Profiling shows that mechanism to cache job results is slowing down the creation of jobs

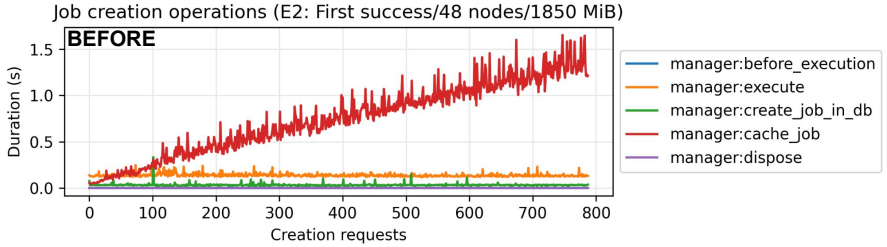
- slower the more jobs are running/files are present
- disabled by default but still affecting workflows
- not needed as Snakemake has its own caching system

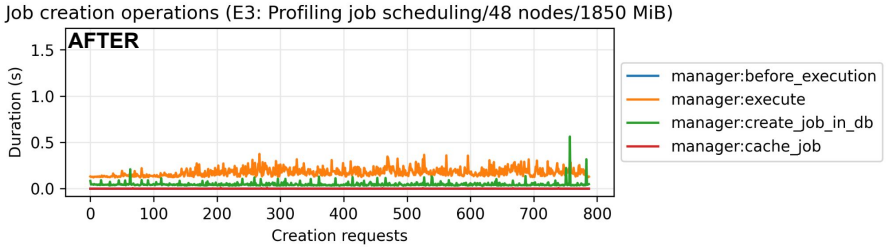
REANA caching system was fully disabled as part of release 0.9.3

Workflow execution now scales much better compared to previous REANA version (-58% execution time)

Lesson: avoid or reduce disk access along critical path







Episode 4: Improving Kerberos authentication

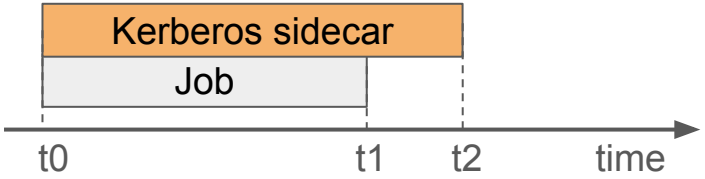
Kerberos is needed to avoid rate limits on EOSPUBLIC, where the input datasets are stored and read from

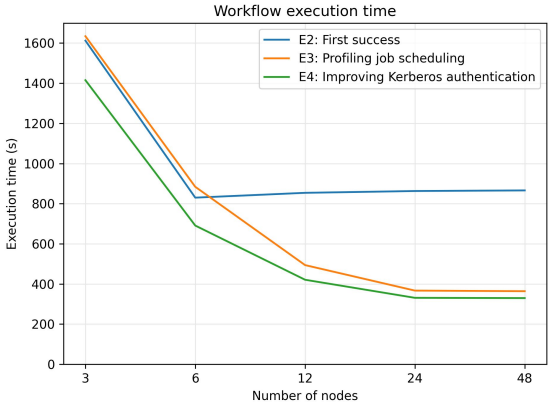
“Sidecar” container periodically renews Kerberos tickets

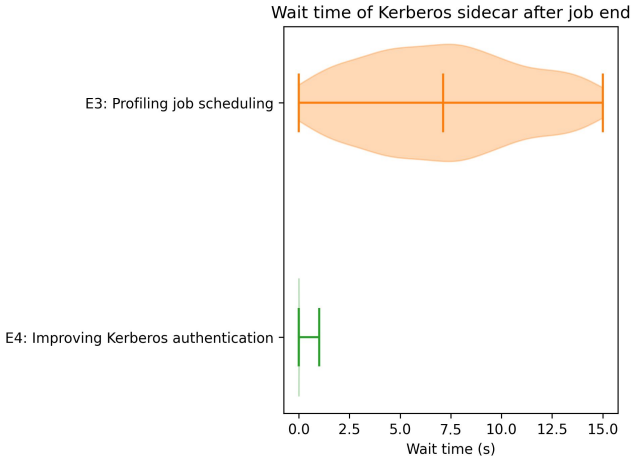
In REANA 0.9.3, the sidecar container also periodically checks if the job has finished, to stop the renewal loop. Periodic polling is done every 15s, so there can be some wait time between the end of the job (t1) and when the sidecar container actually stops (t2).

To reduce the wait time, the sidecar container is now notified when the job finishes (-9% execution time)

Lesson: periodic polling is easy but not always suitable; polling periods need to be carefully tuned





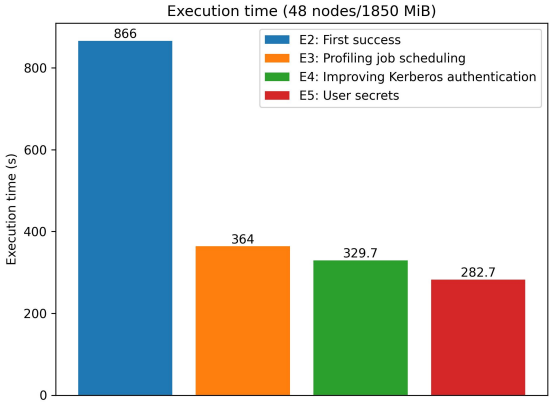


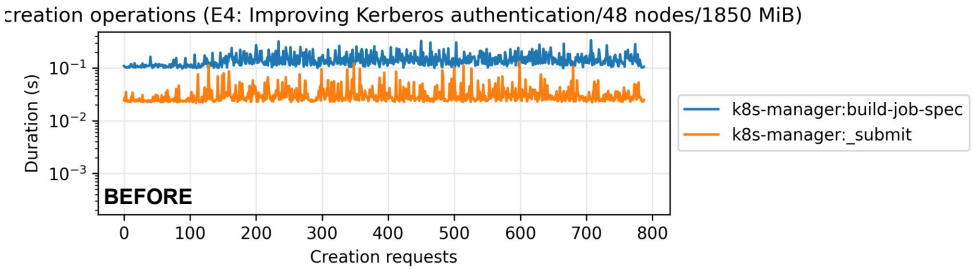
Episode 5: User secrets

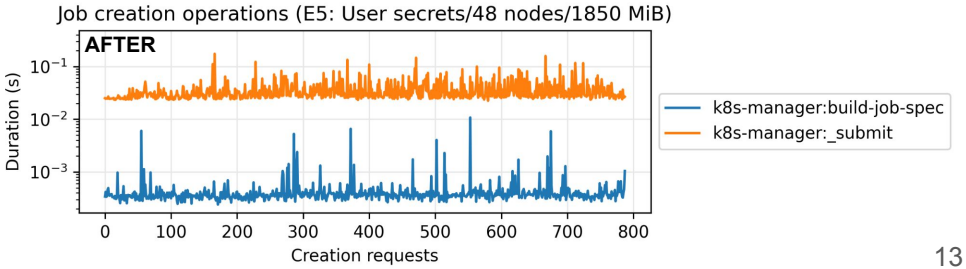
REANA stores user secrets needed by workflow runs (e.g. keytab files) as Kubernetes secrets

Latest version of REANA improves secrets handling by fetching secrets from Kubernetes *once per workflow* instead of *once per job* (-14% execution time), thus speeding up job creation

Lesson: cache and re-use data from external systems if possible to avoid network calls







Episode 6: Database connections

REANA spawns one “orchestrator” pod per workflow, which needs database access

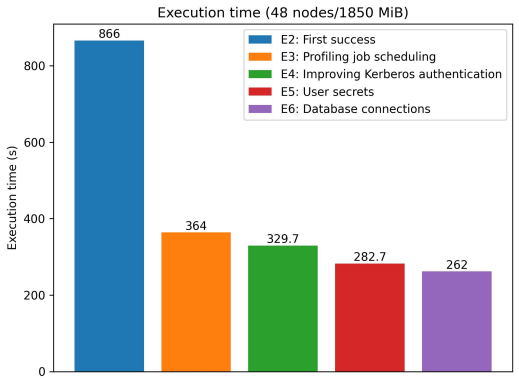
When running hundreds of workflows, many concurrent connections to the database become an issue

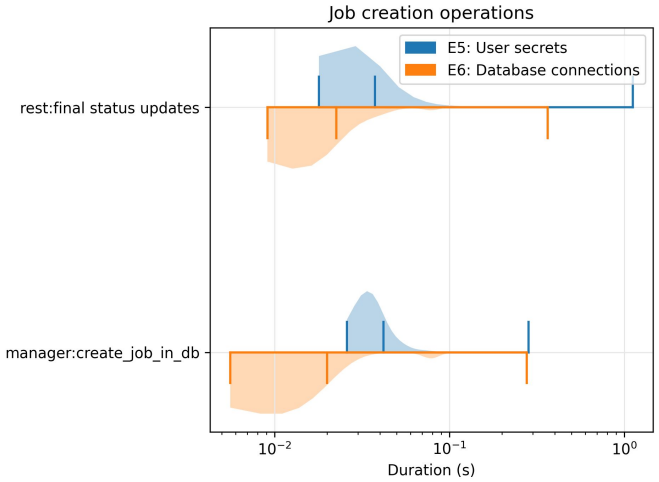
In REANA 0.9.3, the orchestrator pod closes the database connection after each transaction

- good for long lasting jobs, as most of the time connection is idle
- more overhead when spawning many hundreds of jobs in a short amount of time

As of the latest version, REANA supports pgBouncer to allow the pooling of many more concurrent connections (-7% execution time)

Lesson: avoid or optimise database access along critical path



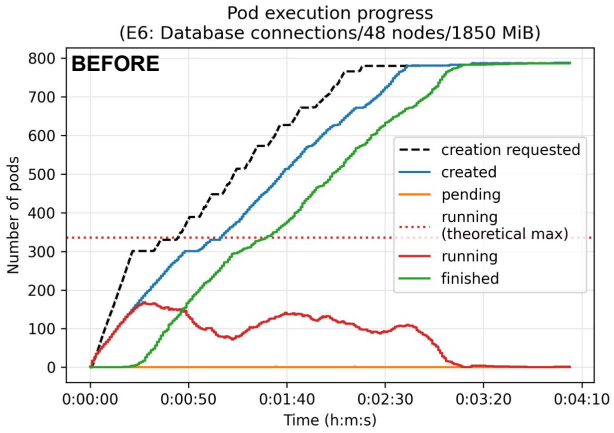


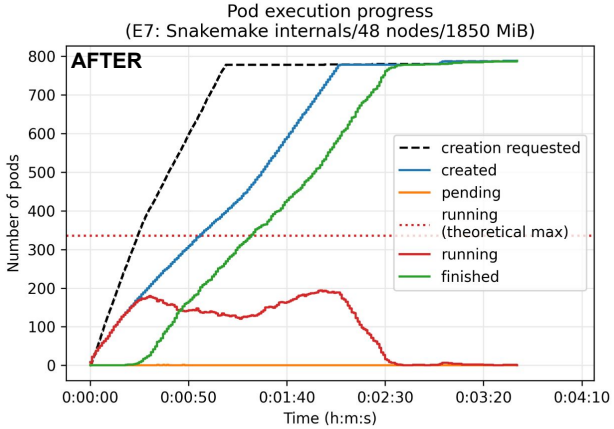
Episode 7: Snakemake internals

REANA configures Snakemake so that there are never more than 300 running jobs at the same time

This is good to avoid overwhelming small or local clusters, but can limit the performance of clusters with many nodes

Limit was raised to 1000 jobs. Cluster utilisation slightly improved, creating jobs faster and reaching 200 jobs running concurrently (-4% execution time)

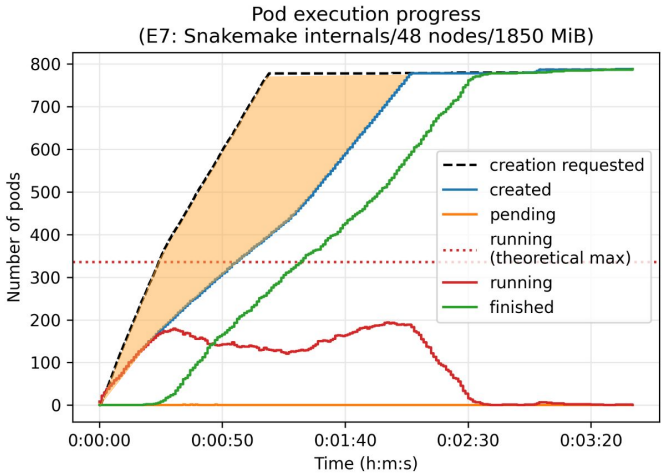




Episode 8: Kubernetes optimisation

Mismatch between when REANA requests the pod creation and when the pod is created in the cluster (orange area)

Checking Kubernetes logs, `kube-controller-manager` is throttling requests to the Kubernetes API server




```
Oct 09 08:23:20 reana-test-[...]-master-0 bash[269122]: I1009 08:23:20.762301
      1 request.go:629] Waited for 88.20434ms due to client-side throttling, not
priority and fairness, request:
PATCH:https://127.0.0.1:6443/api/v1/namespaces/default/pods/reana-run-job-[...]
```

Episode 8: Kubernetes optimisation (2)

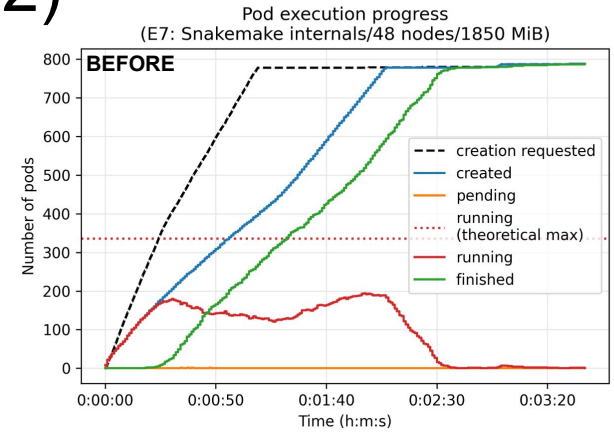
Tuned some parameters of `kube-controller-manager`

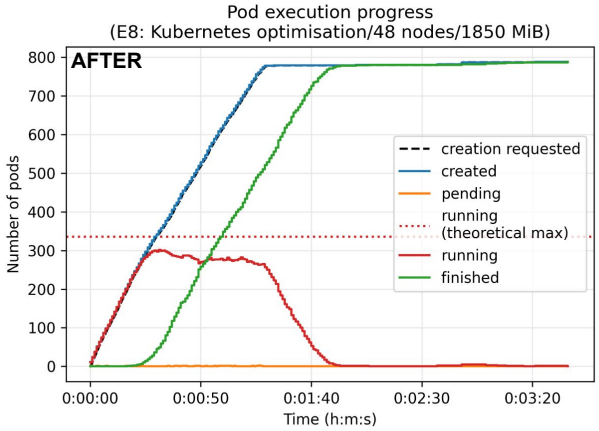
- QPS to Kubernetes API server
  - --kube-api-qps=200
- Burst to Kubernetes API server
  - --kube-api-burst=300
- Jobs that can sync concurrently
  - --concurrent-job-syncs=50
- Garbage collector workers that can sync concurrently
  - --concurrent-gc-syncs=200

Cluster is now close to being fully utilised with more than 300 running jobs at the same time

Full analysis runtime showed small improvements (-7% execution time), but cleanup of jobs is now bottleneck

Lesson: tuning application and system settings for each deployment can have a big impact

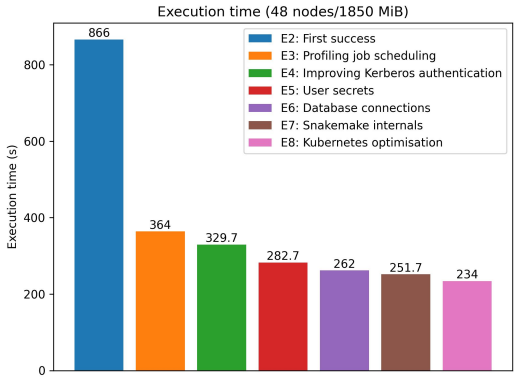


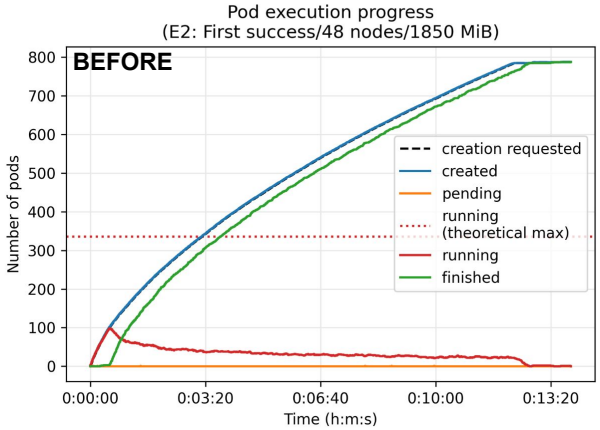


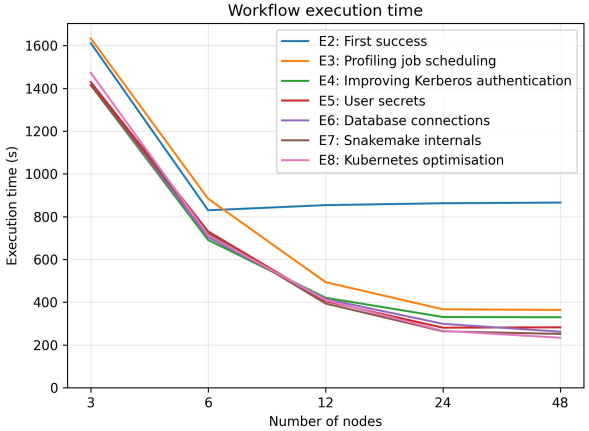
Final results

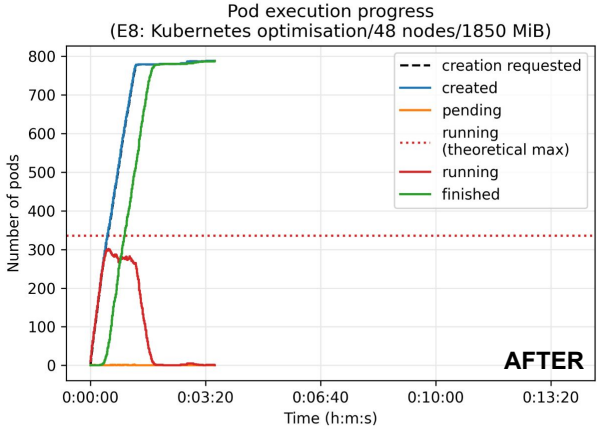
Runtime reduced from 14m26s to 3m54s (3.7x faster) when tested with 48 nodes

Reached 323 peak concurrent jobs, from initial 102 jobs (3.2x more) when tested with 48 nodes









Conclusions

https://snakemake.readthedocs.io

- Snakemake and declarative workflows can efficiently express massively-parallel particle physics computational paradigms
- Analysis Grand Challenge project is very useful to optimise performance of analysis platforms
- Optimisations allowed to improve REANA performance for massively-parallel workflows by a factor of ~3x

https://www.reana.io/

https://home.cern/

https://iris-hep.org/

This work was partially supported by the U.S. National Science Foundation (NSF) under Cooperative Agreement OAC-2226378, OAC-2226379 and OAC-2226380 (FAIROS-HEP) in collaboration with OAC-1836650 and PHY-2323298 (IRIS-HEP)

Backup slides

All results

Execution time (s)	3 nodes*	6 nodes*	12 nodes*	24 nodes*	48 nodes+
E1: First run	-	-	-	-	-
E2: First success	1612	830	854	863	866
E3: Profiling job scheduling	1634	884	494	367	364
E4: Improving Kerberos authentication	1415	691	421	331	330
E5: User secrets	1430	731	405	281	283
E6: Database connections	1417	704	415	299	262
E7: Snakemake internals	1415	724	394	264	252
E8: Kubernetes optimisation	1473	714	410	266	234

*execution time of one run +average execution time of three runs

Theoretical maximum number of running jobs

- Each job requests 1850MiB of RAM
- Each node has 15GB of memory, but not all of it is available to jobs
- When cluster is idle, we have measured up to 14 GiB of free memory in a single node
- 14GiB / 1850MiB = at most 7 jobs can run concurrently in a single node
- In the whole cluster:
  - at most 21 jobs with 3 nodes
  - at most 42 jobs with 6 nodes
  - at most 84 jobs with 12 nodes
  - at most 168 jobs with 24 nodes
  - at most 336 jobs with 48 nodes