Scaling TraceWin beam dynamic simulations on Kubernetes for Reinforcement Learning training

L. Bellan[1], M. Biasotto[1], M. Comunian[1], S. Fantinel[1], M. Gulmini[1], Q.Y. Jian[1,4], D. Lupu[3], D. Marcato[1], G.A. Susto[4], L. Zangrando[2]

1. INFN - LNL 2. INFN - PD 3. INFN - CNAF 4. UNIPD

Istituto Nazionale di Fisica Nucleare

Finanziato dall'Unione europea NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

terabit

WHY?

- At INFN Legnaro National Laboratories we manage a wide particle accelerator complex for research in fundamental physics
- These are monitored and controlled by thousands of sensor and actuators, which need to be tuned in order to keep the beam on its ideal trajectory with specific parameters such as its intensity
- Manual setup by expert operators is usually performed, but can't be enough to take into account all variables. Real-time testing is expensive: steals time to experiments' data-acquisition and because of operation costs
- We want to find the **optimal setpoint** of each beam transport element to optimize beam properties in an automated manner

Acceleratore Di Ioni a Grande Carica Esotici (**ADIGE**) is a beam line located in the middle between SPES and ALPI. It receives the 1+ radioactive ion beams, produced in the SPES TIS, with the goal of increasing their charge state thanks to a charge breeder, so that they can be accelerated more effectively in the ALPI accelerator. The ADIGE Medium Resolution Mass Separator (**MRMS**) electrostatic multipole (AD.EM) is used to correct the beam emittance between two bending dipoles.

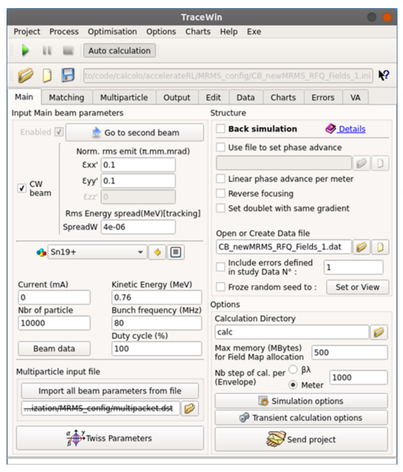
Beam dynamic simulations

TraceWin is a simulator of beam dynamics in particle accelerators with integrated optimization algorithms and is developed by CEA in France.

- It has a command-line interface available, but some properties are accessed only through the GUI

```
$ TraceWin project.ini path_cal=outdir hide nbr_part=10000 .
```

- Not extensible and closed source:
  - No API/library for any programming language
  - **Input and outputs through files**
- **Long start-up operations** for each new simulation: loading of large configuration files in RAM (hundreds of MBs even for shorter beam lines)



Output file	Type	Description
partran1.out	csv	Main beam properties at each beam line element
part_dtl1.dst	binary	Distributions of beam properties at the end of the beam line
dtl1.plt	binary	Distributions of beam properties at each beam line element
Many more…		

HOW: **PY**TRACEWIN

We developed **pytracewin**: a Python wrapper for TraceWin executables to launch simulations from python and jupyter notebooks in an easy way. It doesn't require a GUI and can be run on remote servers:

- Python module to be installed with `pip`
- It defines an homonymous `TraceWin` class to interact with the simulator:
  - It invokes the TraceWin executable with the `subprocess` command
  - It handles runtime parameters, raise exceptions and timeouts to avoid blocking calls
- Parses the output binary file of the TraceWin application to return Pandas DataFrames

Usage

```
from pytracewin import TraceWin

tracewin = TraceWin("MRMS_config/CB_newMRMS_RFQ_Fields_1.ini",
                    executable="TraceWin",
                    hide=True, debug=False)

tracewin.run(params={"nbr_part1": 10000, "ele[27][6]": -1200},
             timeout=30)

tracewin.dst()
tracewin.plt()
tracewin.results()
```

✓ 11.6s

ESS Beam Physics / ess-python-tools

ess-python-tools

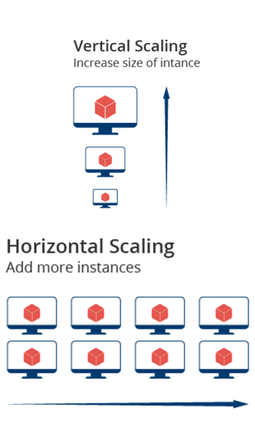
426 Commits 6 Branches 40 Tags 15 Releases

Various Python scripts for ESS work in Beam Physics

	##	z(m)	gama-1	x0	y0	p0	x'0	y'0	W0	SizeX	…
0	0	-0.000000	0.000006	1.245505e-08	4.161037e-08	-0.017999	5.401000e-09	1.686846e-07	-5.822923e-08	4.475022	…
1	1	0.000000	0.000006	1.245505e-08	4.161037e-08	-0.017999	5.401000e-09	1.686846e-07	-5.822923e-08	4.475022	…
2	2	0.214000	0.000006	1.361086e-08	7.770887e-08	2.677506	5.401000e-09	1.686846e-07	-5.822923e-08	8.968381	…

We can launch simulations programmatically from Python, for example to test many values of a parameter to find when it gets close to a minimum and refine from that point. Still, we are limited to a single machine, and to the execution of a single TraceWin instance at any given time because of concurrent I/O operations on the same output files. We could run the application on *fat* servers, but this doesn't scale much, and often simulations are independent from one another. It's an **embarrassingly parallel** type of workload, which would benefit from a distributed environment. For so we need:

- A **coordination mechanism** between tasks
- A **simple interface** to send tasks to workers only when they have finished the previous computation and return the results
- Possibly **backend-agnostic** of the specific platform used (multiple VMs, Cloud, etc.) and the computing power available



SCALING TO THE CLOUD: DASK & KUBERNETES

- **Dask** is an open-source Python library for parallel and distributed computing, thanks to which we can define multiple tasks (e.g. functions operating on Python objects) and submit them on available workers

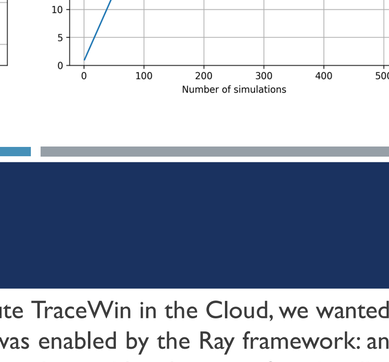
Dask Cluster

- **Docker image** with the environment to run TraceWin:
  - From *gchr.io/dask:latest*, add a new user, ML Python packages and the TraceWin executable
- Automatic setup through **YAML** files
  - Dask cluster on top of a k8s cluster
- **Scale** to a large number of worker nodes
  - Dynamic cluster
- Common interface
  - Do not depend on the VM provider
  - **Portable**

To avoid conflicts because of concurrent access on I/O files we need to force a single TraceWin execution per worker => `@nthreads=1`
Only one available thread, such that no more than one Dask task can be executed at anytime per container or pod

- Automatically distributes tasks which employ native Python data structures (DataFrames and Numpy Arrays) and schedules user-defined jobs in parallel

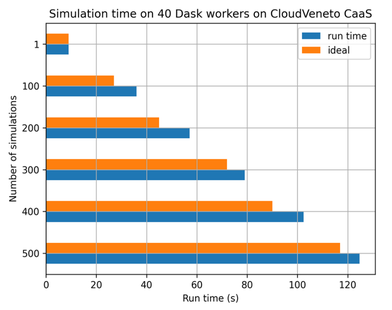
Dask as a task scheduler

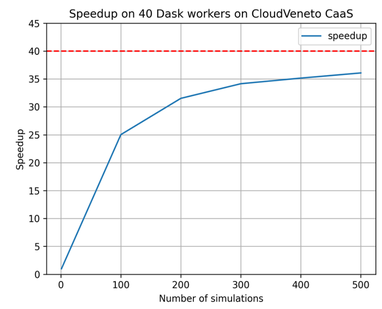


- **INFN—Cloud**: an open, extensible, federated Cloud infrastructure and service portfolio targeted to scientific communities based on OpenStack
- CloudVeneto: federated site of INFN-Cloud that offers fully managed orchestration platform as a cloud service, e.g. users just have to define and deploy their containers using the Container-as-a-Service (CaaS)

Scaling to 40 workers

1 Scheduler (8CPU, 16GB) - **Up to 40 Workers** (1CPU, 1GB)

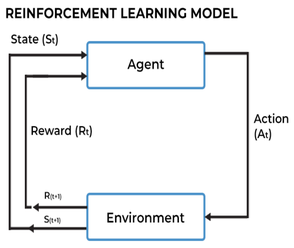




REINFORCEMENT LEARNING WITH RAY

Having deployed an arbitrary number of docker containers to execute TraceWin in the Cloud, we wanted to harness the available computing power for ML purposes, and it was enabled by the Ray framework: an enterprise-grade *unified framework for scaling AI and Python applications*. It provides the same functionality as Dask, that is the scheduling of TraceWin instances in parallel on a k8s cluster thanks to the **Ray Clusters** library, with the support for a wide range of ML algorithms to be executed in a distributed environment, thanks to the **Ray Rlib** library.

Goal: learn a model to reduce the emittance of the beam by acting on MRMS multipole voltage terminals

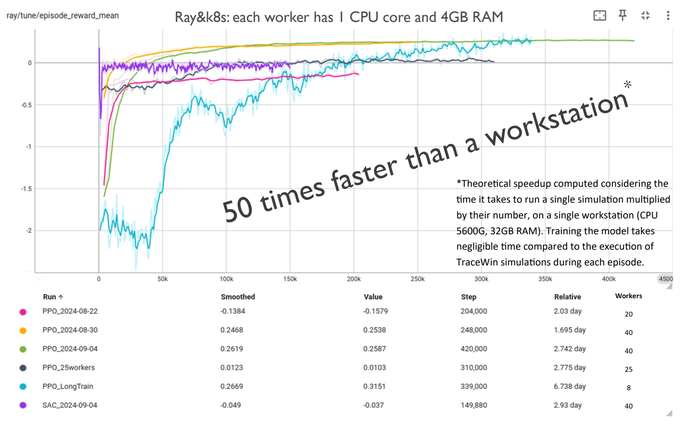


The **State** represents the emittance of the beam with a matrix of 42x42.
An **Environment** class implements standard RL methods by using the TraceWin simulator and performs the following operations:

- **Reset**: resets the multipole to random values
- **Step**: called at each iteration of the episode, it computes the new multipole parameters given the Actions of the agent, retrieves results and computes the Reward

The **initial approach** was based on the Stable-Baseline3 library within Pytorch. The Proximal-Policy-Optimization (PPO) algorithm constrains the policy update to avoid large changes at each iteration by clipping the ration between the new and old policy. With the vanilla algorithm we built a model which would converge in 2.2 steps, but needed ~5 days of training with a single workstation. By executing multiple instances of TraceWin in parallel on a k8s cluster we could train the model considerably longer and took into account **real-world conditions**, such as:

- random initial emittance
- different energy spread
- variation of the input distribution with a different initial emittance



WHAT'S NEXT

TraceWin@INFN Cloud

- Manage distribution of I/O files
  - Kubernetes volumes
  - In-memory volumes to get fast I/O
- Improve accessibility for end users
  - Dask/Ray Cluster as a service?
  - Authentication?
- Better debug messagges
  - e.g. failed TraceWin computation

As for RL training

- Ongoing testing with different off-policy RL algorithms
- Develop a custom policy to exploit previous emittance results
- Extend RL learning to other components in the beam path
- Extend the application to other beam lines

daniel.lupu@cnaf.infn.it

Conference on Computing in High Energy Physics — Kraków (Poland) October 19-25, 2024

Istruzione e Ricerca
PNRR Missione 4, Componente 2, Investimento 3.1