Artifacts Overview

Conference. Euro-Par 2024
Article. Light-weight prediction for improving energy consumption in HPC platforms
Quick links.
Preprint PDF on HAL. https://hal.science/hal-04566184
Artifact data on Zenodo. https://doi.org/10.5281/zenodo.11173631
Artifact Nix binary cache. https://lightweight-pred-sched-europar24.cachix.org
Artifact code Git repository. IRIT, Framagit
Artifact code permalink. Software Heritage

1. Introduction

This document shows how to reproduce the experimental sections (6.2 to 6.5) of article [1]. We hope that this document is enough to reproduce the whole experiments from scratch. However, as reproducing the exact analyses and experiments conducted by the authors requires to download and store lots of input trace data (≃ 300 Go) and to do some heavy computations, various intermediate and final results have been cached and made available on Zenodo to enable the reproduction of only subparts of the experimental pipeline. In particular, the final analyses of the article are done in standalone notebooks whose input data is available and small.

Unless otherwise specified, all commands shown in this document are expressed in sh and are thus compatible with `bash` and `zsh`. The disk/bandwidth/computation overhead of commands is specified in the footer part of each command box, and significant overheads are **emphasized**. Unless otherwise specified, execution times have been obtained on a powerful computation node that uses 2x Intel Xeon Gold 6130. A MD5 hash is given for the output files that we are think are important, and all these files can be downloaded on Zenodo. The MD5 hashes have been computed by GNU coreutils's `md5sum` command.

```
echo 'Commands should look like this'
echo 'Example command' > /tmp/example-output
sleep 1
```

md5 hash	output file
fb9807302a8a925bb7a3963d03cedd04	/tmp/example-output

Time: 00:00:01.

2. Getting Started Guide

All the software environments required to reproduce the analyses and experiments of article [1] are open source and have been packaged with Nix. Nix can build the **full** software stack needed for this experiment as long as source code remains available. As we also put most of the source code needed by this artifact on Software Heritage we hope that this artifact will have long-term longevity. For the sake of this artifact reviewers' quality of life, we have set up a binary cache with precompiled versions of the software used in the experiments.

No special hardware is required to reproduce our work. Our Nix environments are likely to work on future Nix versions, but for the sake of traceability we stress that we have used Nix 2.18.0 installed either by nix-portable 0.10.0 or directly available via NixOS using channel `23-11`. Our software environments likely work on all platforms supported by Nix (Linux on `i686`/`x86_64`/`aarch64` and MacOS on `x86_64`/`aarch64` as of 2024-05-07) but we have only tested them on Linux on `x86_64`. More precisely, we have used the Dahu Grid'5000 cluster (Dell PowerEdge C6420, 2x Intel Xeon Gold 6130, 192 GiB of RAM) on the default oper-

ating system available on Grid'5000 as of 2024-05-07 (Debian `5.10.209-2` using Linux kernel `5.10.0-28-amd64`).

2.1. Install Nix

If you are already using NixOS, Nix should already be usable on your system and you can go to Section 2.2.

Otherwise you must install Nix to use the software we have packaged. We recommend to use **up-to-date documentation on how to install Nix**. As of 2024-05-07 the recommended command to install Nix (on a Linux system running systemd, with SELinux disabled and `sudo` usable) is the following.

```
sh <(curl -L https://nixos.org/nix/install) --daemon
```

Please note that you may need to launch a new shell, to source a file or to modify your shell configuration script as indicated by the Nix installer.

Testing your Nix installation.

- Launching `nix-shell --version` should run successfully and print you the Nix version you have installed.
- Launching `nix-build --version` should run successfully and print you the Nix version you have installed.

2.2. Enable Nix flakes

Our Nix packages rely on **Nix flakes**, which are not enabled by default as of 2024-05-07. Nix flakes must be enabled to use the software we have packaged. We recommend to use **up-to-date documentation on how to enable flakes**. However for the sake of this artifact guide self-containedness, steps to enable flakes are given below. Please note that the way to enable flakes depend on whether you are on NixOS or not.

If **you are using NixOS** flakes can be enabled by setting at least the `nix-command` and `flakes` Nix experimental settings in your NixOS configuration file. In other words, your NixOS configuration file should have a content similar to the following.

```
nix.settings.experimental-features = [
  "nix-command" "flakes"
];
```

If you are **not using NixOS** flakes can be enabled by setting at least the `nix-command` and `flakes` `experimental-features` in your Nix configuration file. The Nix configuration file path is `~/.config/nix/nix.conf` on non-NixOS Linuxes. In other words, your Nix configuration file should have content similar to the one below.

```
experimental-features = nix-command flakes
```

Testing your Nix flakes.

- Launching `nix --version` should run successfully and print you the Nix version you have installed.
- Launching `nix flake --version` should run successfully and print you the Nix version you have installed.
- Launching `nix build 'github:nixos/nixpkgs?ref=23.11#hello'` should create a `result` symbolic link in your current directory. Then, launching `./result/bin/hello` should print `Hello, world!`.
- Launching `nix shell 'github:nixos/nixpkgs?ref=23.11#hello' --command hello` should print `Hello, world!`.

2.3. Using our Nix binary cache (optional)

Using our binary cache is recommended as it enables to download precompiled versions of our software environments instead of building them on your own machine. Our cache has the following properties.

- URL. **`https://lightweight-pred-sched-europar24.cachix.org`**
- Public key. `lightweight-pred-sched-europar24.cachix.org-1:dHsm8geVskEOsZIjzXtVCmPvh0L2zwTlLm8V4QoJdgI=`

Once again, we recommend to use **up-to-date documentation on using a Nix binary cache**, but instructions are given below on how to use our cache as of 2024-05-07.

If **you are using NixOS**, you must edit your NixOS configuration file to add our cache URL in the `nix.settings.substituters` array, and our cache public key in the `nix.settings.trusted-public-keys` array.

If you are **not using NixOS**, you must edit your Nix configuration file (`~/.config/nix/nix.conf` on Linux) to add our cache URL in the `substituters` array, and our cache public key in the `trusted-public-keys` array. Please find below a **2-line** example configuration file that only enables the NixOS binary cache and ours.

```
substituters = https://cache.nixos.org https://lightweight-pred-sched-europar24.cachix.org
trusted-public-keys = cache.nixos.org-1:6NCHdD59X431o0gWypbMrAURkbJ16ZPMQFGspcDShjY= lightweight-pred-sched-europar24.cachix.org-1:dHsm8geVskEOsZIjzXtVCmPvh0L2zwTlLm8V4QoJdgI=
```

2.4. Version traceability and quick Nix flake explanation

All the software versions use in this artifact are **fully and purely defined** thanks to Nix flakes. More concretely, **our artifact Git repository** at commit `5a15139dadde8d923703ece93745fa250b1a0c53` defines how to build and use the software environments used to reproduce all our work. These environments are named *shells* in Nix terminology. Nix builds software in isolated (filesystem, network…) sandboxes to remove most sources of non-determinism, and forces inputs (source code, dependencies) to have well-defined versions (well defined content hash and version control commit).

Our artifact Git repository **directly** defines how the scripts used to reproduce Article [1] should be built, as the source code of these scripts is inside our artifact Git repository. Software that we manage but whose source code is stored in another repository (*e.g.*, the scheduler implementation used in our scheduling experiment, the Batsim simulator…) define how they should be built in their own Git repository. Software that we do not manage but that we use is either imported from the repository of the software itself if they use flakes (*e.g.*, Typst), imported from **nixpkgs** if possible (*e.g.*, gzip), or otherwise defined in our artifact Git repository (*e.g.*, Python's fastparquet library).

Nix flakes enable to link together several Nix software descriptions that are distributed in different repositories. This is done by (recursively) tracing the *inputs* (Flake dependencies) needed by the main flake of our artifact Git repository. Consequently, the flake of our artifact Git repository **indirectly** defines all the softwares needed and their versions.

For the sake of traceability, here are the software versions that we think are the most important.

- **Our artifact Git repository** commit `5a15139dadde8d923703ece93745fa250b1a0c53`
- Nix 2.18.0
- Nixpkgs commit `057f9aecfb71c4437d2b27d3323df7f93c010b7e`
- NUR-kapack commit `4d8ca88fd8a0a2287ee5c023877f14d53d4854c1`
- SimGrid release 3.34 (commit `036c801d55e3ab07b470c79640109080fed049a1`)
- intervalset commit `13d8f2d7000d4dc4e3202422658b0b5d83f83679`
- batprotocol commit `25bc5bbf039c18a8024c4ab326047ba56800376a`
- easy-powercap release europar24 (commit `659660c35650e9f46ec47e8c0743d75649e68d7b`)
- Batsim commit `ee797ccebbb95410479663ee0547e752112fc83e`
- Python 3.11.6
  - Pandas 2.1.1
  - fastparquet release 2024.2.0 (commit `eec9e614603f9be3cb495409ccb263caff53fe9d`)
- R 4.3.2
  - tidyverse 2.0.0
- Typst commit `21c78abd6eecd0f6b3208405c7513be3bbd8991c` (after 0.11.0)

3. Step-by-Step Instructions

All the scripts strongly related to the experiments of Article [1] are available on **the Framagit GitLab instance**, and on **Software Heritage** for long-term longevity.

The repository can be cloned with the following commands. The repository is explicitly set to the commit we have tested to write this artifact overview. Please note that updating the repository may be useful – *e.g.*, if errors have been found and fixed, or if other parts of the experimental pipeline have been added.

```
git clone https://framagit.org/batsim/artifact-europar24-lightweight-power-pred-sched.git artifact-repo
cd artifact-repo
git checkout 5a15139dadde8d923703ece93745fa250b1a0c53
```

All commands below should be executed from the **root of the cloned Git repository**.

The step-by-step instructions of this document can be used in several ways **depending on your goal**.

1. You can **check** the final analyses (code + plots) done in Article [1] by reading the provided pre-rendered notebooks available on Zenodo.
2. You can **reproduce** the **final analyses** by first downloading the provided aggregated results of the experiments from Zenodo, and then by running the notebooks yourself. This enables you to **edit** our notebooks before running them, so that you can to modify the analyses done or add your own.
   - Refer to Section 3.4 for instructions to analyze the results of the machine learning experiment.
   - Refer to Section 3.5.3 for instructions to analyze the results of the scheduling experiment.
3. You can **reproduce** our **experimental campaigns** by downloading the provided experiment input files from Zenodo, and then by running the experiment yourself. This can enable you to make sure that our experiment can be reproduced with the **exact same parameters and configuration**.
   - Refer to Section 3.5.2 for instructions to reproduce the scheduling experiment.
4. You can **fully reproduce** our **experimental campaigns** by downloading original traces of the Marconi100, by generating the experimental campaigns parameters yourself (enabling you to hack provided command-line parameters or provided code), and then by running the experiment yourself. You can follow all steps below in this case, but **please do note that this is disk/bandwidth/computation-intensive.**

3.1. Analysis and modeling of the power behavior of Marconi100 nodes

3.1.1. Get power and job Marconi100 traces on your disk

This section downloads parts of the Marconi100 trace as archives from the ExaData Zenodo files, checks that the archives have the right content (via MD5 checksums), extracts the data needed by later stages of the pipeline (node power usage traces, jobs information traces), then finally removes unneeded extracted files and the downloaded archives.

```
nix develop .#download-m100-months --command \
            m100-data-downloader ./m100-data \
                                 22-01 22-02 22-03 22-04 22-05 22-06 22-07 22-08 22-09
```

md5 hash	output file
604aa2493d688a77a7f771ad1dc91621	m100-data/22-01_jobs.parquet
53e5939579412cb99347d14c62ce789e	m100-data/22-02_jobs.parquet
4da725eb59b311c7b7f5568bd389d120	m100-data/22-03_jobs.parquet
6091df746cf94d346a3900153777496d	m100-data/22-04_jobs.parquet
7f1e442f59203b990217ecefb56aec4b	m100-data/22-05_jobs.parquet
f8f3fa87a6310f73f8c2e8ac013cebaa	m100-data/22-06_jobs.parquet
350040cbc9532184679f226eff73c6f5	m100-data/22-07_jobs.parquet
11eebd414fbbbe2b4d9f3aa1568260ef	m100-data/22-08_jobs.parquet
9d60ba75bd53ab8e689097f2ccfe2f42	m100-data/22-09_jobs.parquet
9a0a5a883862889ea29ebe866038aacf	m100-data/22-01_power_total.parquet
a13b1a287197cdaf18ca172c0cf6eec8	m100-data/22-02_power_total.parquet
f4c3f05ff5a6b28da48d56c11f8a5146	m100-data/22-03_power_total.parquet
f02745d785f6afa812a67bd70ca8090f	m100-data/22-04_power_total.parquet
2969a1a6f501f35b12f80ec4f3c7b298	m100-data/22-05_power_total.parquet
4bd100c4ebd048c80dea58f064670e1a	m100-data/22-06_power_total.parquet
2631979125b4454e177977da6a482073	m100-data/22-07_power_total.parquet
b36373acddc0fbf41e7171ded786e877	m100-data/22-08_power_total.parquet
82c3f6f013c9254cabfd23c67a3e7b0f	m100-data/22-09_power_total.parquet

Download+temporary disk: 254 Go. Final disk: 928 Mo. **Time: 00:40:00.**

3.1.2. Aggregate power traces per node

The following command traverses all the Marconi100 power traces and counts how many times each node was at each power value.

Required input files.

- All power parquet files outputted by Section 3.1.1.

```
nix develop .#py-scripts --command \
            m100-agg-power-months ./m100-data/ ./m100-data/22-agg_ \
                                  22-01 22-02 22-03 22-04 22-05 22-06 22-07 22-08 22-09
```

md5 hash	output file
20e5d7b3f941efb1c5b6083e4752b647	m100-data/22-agg_power_total.csv

Disk: 1 Mo. Time: 00:03:00.

3.1.3. Analyze Marconi100 power traces

The following commands runs a notebook that analyses the node power consumption of the Marconi100 trace. The notebook also generates a power model of the Marconi100 nodes, which is required to generate a simulation platform.

Required input files.

- `m100-data/22-agg_power_total.csv` (output of Section 3.1.2).

```
nix develop .#r-notebook --command \
            Rscript notebooks/run-rmarkdown-notebook.R \
                    notebooks/m100-power-trace-analysis.Rmd
```

md5 hash	output file
a2ebebb21586d1adfa63fc917e1517bd	m100-data/22-powermodel_total.csv
9829bb1ebb9ca5811676db3c56b6458c	notebooks/m100-power-trace-analysis.html

We could not make HTML notebook binary reproducible despite our best efforts. Their content should be completely reproducible though.

Disk: 1.7 Mo. Time (laptop): 00:00:10.

3.2. Job power prediction

The experimental workflow consists of three parts, (i) preprocessing of the original data, and (ii) prediction of the mean and maximum power consumption. Please note that reproducing this section involves **heavy computations** and **big data**. We have **not** made intermediate files available on Zenodo as they were too big.

3.2.1. Pre-processing

3.2.1.1. Step 1

```
for month in 22-01 22-02 22-03 22-04 22-05 22-06 22-07 22-08 22-09; do
  nix develop .#py-scripts --command m100-pred-preprocess1 \
                                   -j ./m100-data/${month}_jobs.parquet \
                                   -m ./m100-data/${month}_power_total.parquet
done
```

Memory: 128 Go. Time (sequential): 18:00:00.

3.2.1.2. Step 2

```
for month in 22-01 22-02 22-03 22-04 22-05 22-06 22-07 22-08 22-09; do
  nix develop .#py-scripts --command m100-pred-preprocess2 \
                                   -js ./m100-data/${month}_filter12_singlenode.csv \
                                   -jm ./m100-data/${month}_filter12_multinode.csv \
                                   -m ./m100-data/${month}_power_total.parquet
done
```

Memory: 128 Go. Time (sequential): 66:00:00.

3.2.2. Aggregate step 2 output into a single file

```
find . -name '*filter123*' | \
  tar -zcvf exadata_job_energy_profiles.tar.gz --files-from -
```

Disk: 32 Go.

3.2.3. Compute power metrics and add job information

```
for month in 22-01 22-02 22-03 22-04 22-05 22-06 22-07 22-08 22-09; do
    nix develop .#py-scripts --command m100-pred-jobs-extract-power-metrics \
                                       -d ./m100-data/${month}
done
```

Disk: 32 Go.

3.2.4. Merge files into a single CSV file

This will output the `filter123_all_jobs_aggmetrics.csv.gz` needed for the prediction script.

```
nix develop .#py-scripts --command m100-pred-merge-jobfiles -d ./m100-data/
```

Disk: 82 Mo.

3.3. Predicting Job mean and maximum power consumption

```
mkdir ./m100-data/total_power_mean_predictions_users_allmethods_mean
mkdir ./m100-data/total_power_mean_predictions_users_allmethods_max

nix develop .#py-scripts --command \
            run-prediction-per-user-allmethods-mean \
              -j ./m100-data/filter123_all_jobs_aggmetrics.csv.gz \
              -o ./m100-data/total_power_mean_predictions_users_allmethods_mean

nix develop .#py-scripts --command \
            run-prediction-per-user-allmethods-max \
              -j ./m100-data/filter123_all_jobs_aggmetrics.csv.gz \
              -o ./m100-data/total_power_mean_predictions_users_allmethods_max
```

Memory: 128 Go. Time (sequential): 72:00:00.

3.3.1. Compressing prediction output into single files

The expected output data has been stored on **Zenodo**.

```
tar -cvzf ./m100-data/power_pred_users_allmethods_max.tar.gz \
          ./m100-data/total_power_mean_predictions_users_allmethods_mean
tar -cvzf ./m100-data/power_pred_users_allmethods_mean.tar.gz \
          ./m100-data/total_power_mean_predictions_users_allmethods_max
```

md5 hash	output file
fdcc47998a7e998abde325162833b23e	power_pred_users_allmethods_max.tar.gz
954f782a75c9a5b21c53a95c0218e220	power_pred_users_allmethods_mean.tar.gz

Disk: 82 Mo.

3.4. Analyzing prediction results

This analysis requires that the two job power prediction archives (outputs of Section 3.2, available on **Zenodo**) are available on your disk in the `./user-power-predictions` directory. The following command populates the `./user-power-predictions/data` by extracting the archives and uncompressing all the required files on your disk.

```
mkdir ./user-power-predictions/data
nix develop .#merge-m100-power-predictions --command \
  tar xf ./user-power-predictions/*mean.tar.gz --directory ./user-power-predictions/data
nix develop .#merge-m100-power-predictions --command \
  tar xf ./user-power-predictions/*max.tar.gz --directory ./user-power-predictions/data
nix develop .#merge-m100-power-predictions --command \
  gunzip ./user-power-predictions/data/*/*.gz
```

Disk: 519 Mo. Time: 00:00:05.

The analysis of the predictions, which also generates Figures 2 and 3 of Article [1], can be reproduced with the following command.

```
nix develop .#r-py-notebook --command \
            Rscript notebooks/run-rmarkdown-notebook.R \
                    notebooks/prediction-results-analysis.Rmd
```

md5 hash	output file
89e531e6b0e8b767acb58276f89267b4	notebooks/fig2a-distrib-job-power-mean.svg
3bcd5f8e515479f3f81eda23c0dc7291	notebooks/fig2b-distrib-job-power-max.svg
0bc88e65ae495a8d6ec7d3cbcfca12ae	notebooks/fig3a-pred-mape-mean-power.svg
a19b1a7c5dc72ec73a5349d85fc68fa3	notebooks/fig3b-pred-mape-max-power.svg
04c2d5ef412b791a4d5515ec0719b3d0	notebooks/prediction-results-analysis.html

We could not make HTML notebooks and Python-generated images binary reproducible despite our best efforts. Their content should be completely reproducible though.

Time (laptop): 00:00:20.

3.5. Job scheduling with power prediction

This section shows how to reproduce Sections 6.4 and 6.5 of article [1].

3.5.1. Prepare all the files required to run the simulation

3.5.1.1. Generate a SimGrid platform

The following command generates the SimGrid platform used for the simulations.

Required input files.

- `m100-data/22-powermodel_total.csv`, the M100 node power model (output of Section 3.1.3).

```
nix develop .#py-scripts --command \
            m100-generate-sg-platform ./m100-data/22-powermodel_total.csv 100 \
                                      -o ./expe-sched/m100-platform.xml
```

md5 hash	output file
b5c28261bbe6bcea017ac03b1ef97bd9	expe-sched/m100-platform.xml

Time: 00:00:01.

3.5.1.2. Generate simulation instances

The following commands generate workload parameters (*i.e.*, when each workload should start and end), taking start points at random during the 2022 M100 trace. Simulation instances are then generated from the workload parameters.

Required input files.

- `expe-sched/m100-platform.xml` (output of Section 3.5.1.1).

```
nix develop .#py-scripts --command \
            m100-generate-expe-workload-params -o ./expe-sched/workload-params.json
nix develop .#py-scripts --command \
            m100-generate-expe-params ./expe-sched/workload-params.json \
                                      ./expe-sched/m100-platform.xml \
                                      -o ./expe-sched/simu-instances.json
```

md5 hash	output file
e1b4475f55938ad6de4ca500bddc7908	expe-sched/workload-params.json
3a7e7d8183dcb733d6b49d86b2ab3b14	expe-sched/simu-instances.json

Disk: 1.3 Mo. Time: 00:00:01.

3.5.1.3. Merge job power predictions and jobs information into a single file

The job power predictions (outputs of Section 3.2, available on [Zenodo]) are two archives that we assume are on your disk in the `./user-power-predictions` directory. These archives contain gzipped files for each user. To make things more convenient for the generation of simulation inputs, all the job power prediction files are merged into a single file with the following commands.

```
mkdir ./user-power-predictions/tmp
nix develop .#merge-m100-power-predictions --command \
  tar xf ./user-power-predictions/*mean.tar.gz --directory ./user-power-predictions/tmp
nix develop .#merge-m100-power-predictions --command \
  tar xf ./user-power-predictions/*max.tar.gz --directory ./user-power-predictions/tmp
nix develop .#merge-m100-power-predictions --command \
  gunzip ./user-power-predictions/tmp/*/*.gz
nix develop .#merge-m100-power-predictions --command \
            m100-agg-power-predictions ./user-power-predictions/tmp \
                                       ./m100-data/22-job-power-estimations.csv
rm -rf ./user-power-predictions/tmp
```

md5 hash	output file
86a056a9d61cca59b80adf95fa8bff22	./m100-data/22-job-power-estimations.csv

Temporary disk: 519 Mo. Final disk: 25 Mo. Time: 00:00:30.

Similarly, Marconi100 job traces are also merged into a single file.

```
nix develop .#py-scripts --command \
            m100-agg-jobs-info ./m100-data/ ./m100-data/22-jobs.csv \
                               22-01 22-02 22-03 22-04 22-05 22-06 22-07 22-08 22-09
nix develop .#py-scripts --command \
            m100-join-usable-jobs-info ./m100-data/22-job-power-estimations.csv \
                                       ./m100-data/22-jobs.csv \
                                       ./m100-data/22-jobs-with-prediction.csv
```

md5 hash	output file
c7d00104663b13e2992ec10749d6a162	m100-data/22-jobs-with-prediction.csv

Final disk: 343 Mo. Time: 00:02:00.

3.5.1.4. Generate workloads

The following command generates all the workloads needed by the simulation. **This step is very long, even while using all the cores of a powerful computation node!**

```
nix develop .#py-scripts --command \
            m100-generate-expe-workloads ./expe-sched/workload-params.json \
                                         ./m100-data/22-jobs-with-prediction.csv \
                                         ./m100-data \
                                         -o /tmp/wlds
```

Output should be the `/tmp/wlds` directory, with should contain 1.4 Go of files.

- 30 Batsim workload files – *e.g.*, `/tmp/wlds/wload_delay_5536006.json`
- 30 unused `input_watts` files – *e.g.*, `/tmp/wlds/wload_delay_5536006_input_watts.csv`
- 1 directory per replayed job in `/tmp/wlds/jobs/` (total of 121544 jobs)
- 1 dynamic power trace per job in `/tmp/wlds/jobs/JOBID_STARTREPLAYTIME/dynpower.csv`

Disk: 1.4 Go. Time: 05:30:00.

3.5.2. Run the simulation campaign

The following command runs the whole simulation campaign. The main results of the simulations are aggregated *in situ* into a single file. Details about each simulation are stored in the `/tmp/simout` directory, one subdirectory per simulation instance – please refer to `expe-sched/simu-instances.json` for the mapping from unique simulation instances to the simulation parameters.

Required input files.

- `expe-sched/m100-platform.xml`, the SimGrid platform file (output of Section 3.5.1.1).
- `expe-sched/simu-instances.json`, the set of simulation instances (output of Section 3.5.1.2).
- The `/tmp/wlds` directory (**1.4 Go**) that contains all the workload files (output of Section 3.5.1.4).

Please note that all input files can be downloaded from **Zenodo** if you have not generated them yourself. In particular to populate the `/tmp/wlds` directory you can **download file** `workloads.tar.xz` and then **extract it** into `/tmp/` via a command such as the following. `tar xf workloads.tar.xz --directory=/tmp/`

```
nix develop .#simulation --command \
            m100-run-batsim-instances \
              ./expe-sched/simu-instances.json \
              -w /tmp/wlds \
              -o /tmp/simout \
              --output_state_file ./expe-sched/simu-campaign-exec-state.json \
              --output_result_file ./expe-sched/simu-campaign-agg-result.csv
```

md5 hash	output file
`2f31cf5a3ca6b2f46a2d426c9558f351`	`expe-sched/simu-campaign-agg-result.csv`

Disk: 7.6 Go. Time: 00:06:00.

3.5.3. Analyze the simulation campaign results

The following command runs a notebook that analyze the aggregated results of the simulation campaign, and outputs Figure 4 and Figure 5 of Article [1].

Required input files.

- `expe-sched/simu-campaign-agg-result.csv`, the simulation campaign main output of Section 3.5.2.

```
nix develop .#r-notebook --command \
            Rscript notebooks/run-rmarkdown-notebook.R \
                    notebooks/simulation-output-analysis.Rmd
```

md5 hash	output file
660144ea7340a7accf4eb8c7c2a7a3fa	notebooks/fig4-sched-mean-power-distribution.svg
df07dec01ea5dd176ef406b26638d180	notebooks/fig5-sched-mtt-diff-distribution.svg
e00304f9f2fd1819b72ca8b6b802db9c	notebooks/simulation-output-analysis.html

We could not make HTML notebook binary reproducible despite our best efforts. Their content should be completely reproducible though.

Time (laptop): 00:00:30.

Bibliography

[1] D. Carastan-Santos, G. Da Costa, M. Poquet, P. Stolf, and D. Trystram, "Light-weight prediction for improving energy consumption in HPC platforms," *European Conference on Parallel Processing (Euro-Par) 2024*, Aug. 2024, [Online]. Available: **https://hal.science/hal-04566184**