800 1222·2022 ANNI
UNIVERSITÀ DEGLI STUDI DI PADOVA

Distributed processing of JUNO datasets for improved energy reconstruction

Group: ***Città Romanze***

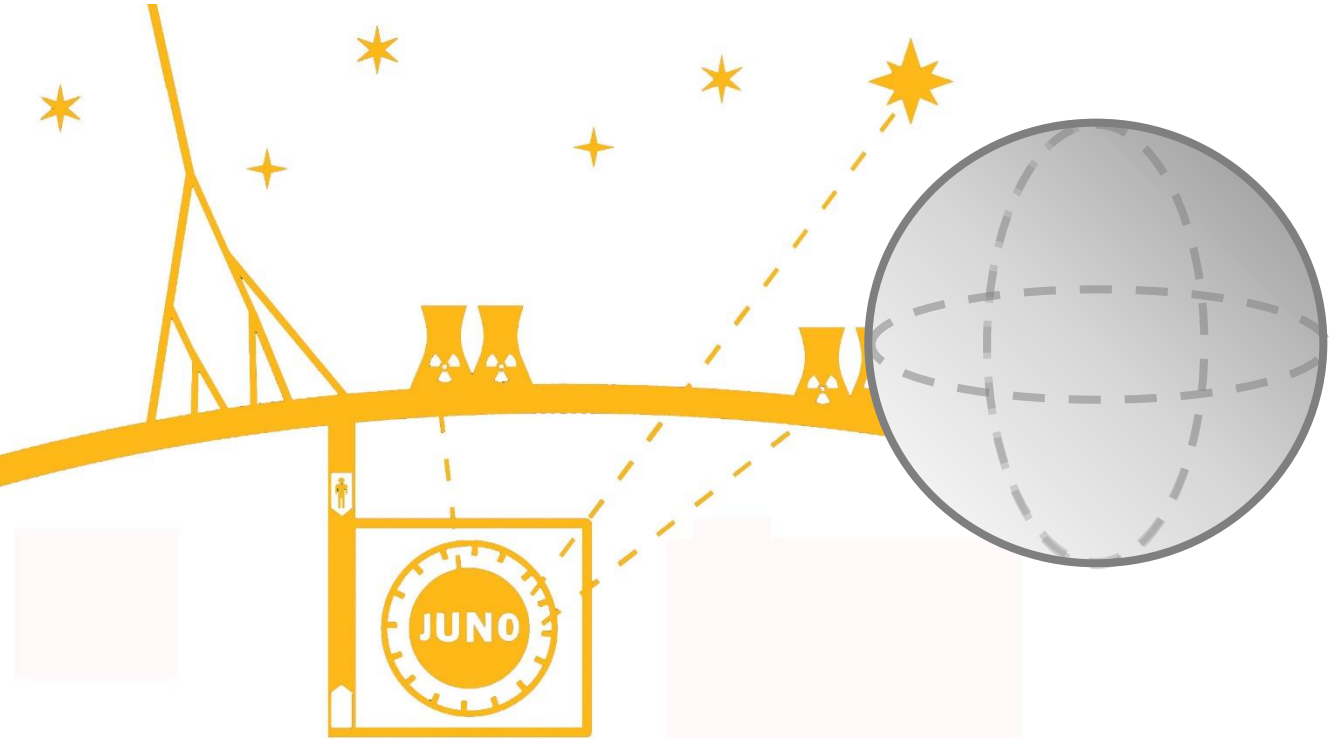
- Pietro Cappelli
- Alberto Coppi
- Giacomo Franceschetto
- Nicolò Lai

Date: 14/07/2022

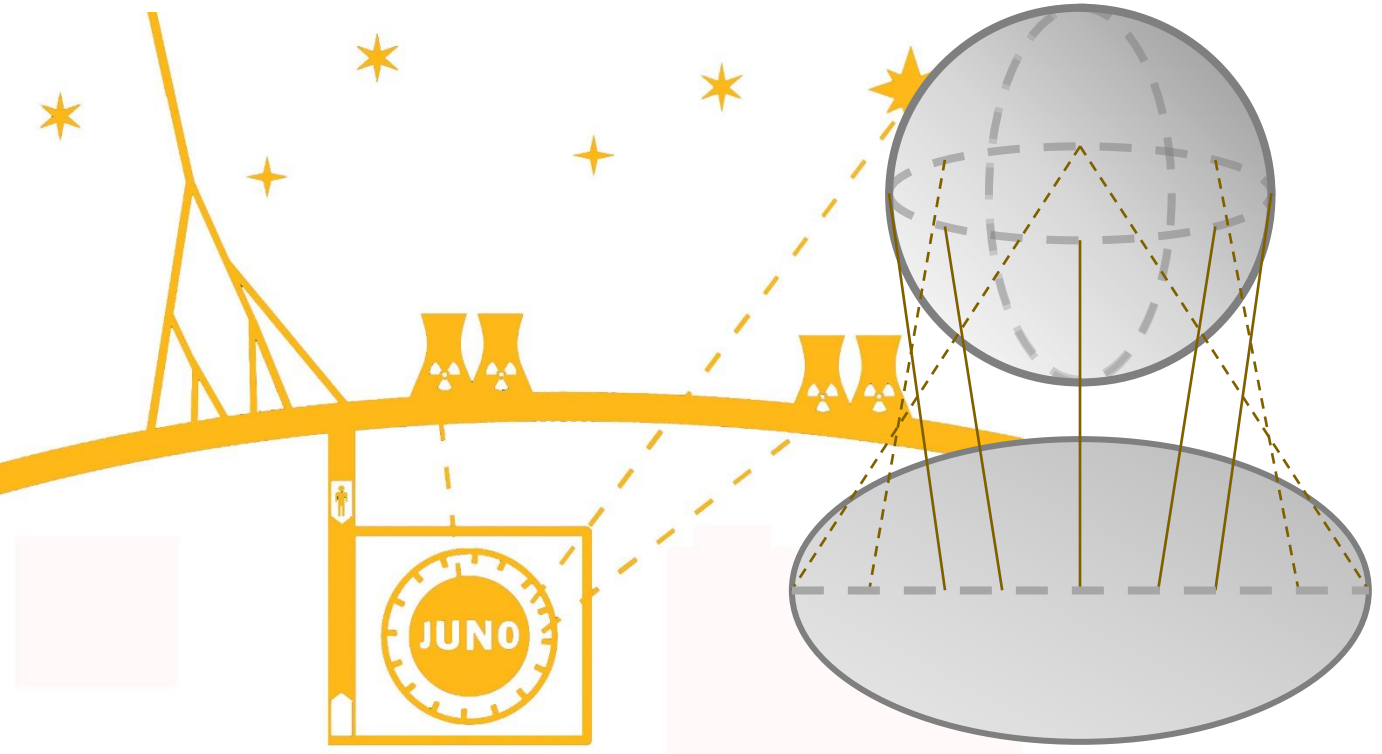
Introduction

JUNO experiment & task overview

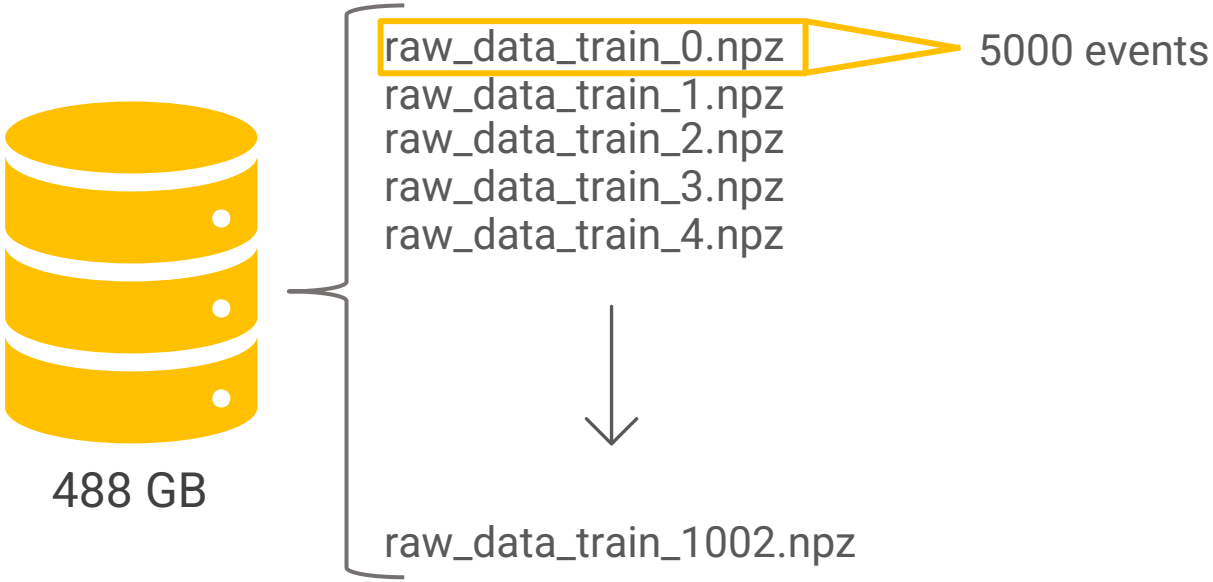
JUNO experiment & task overview



JUNO experiment & task overview



Dataset overview

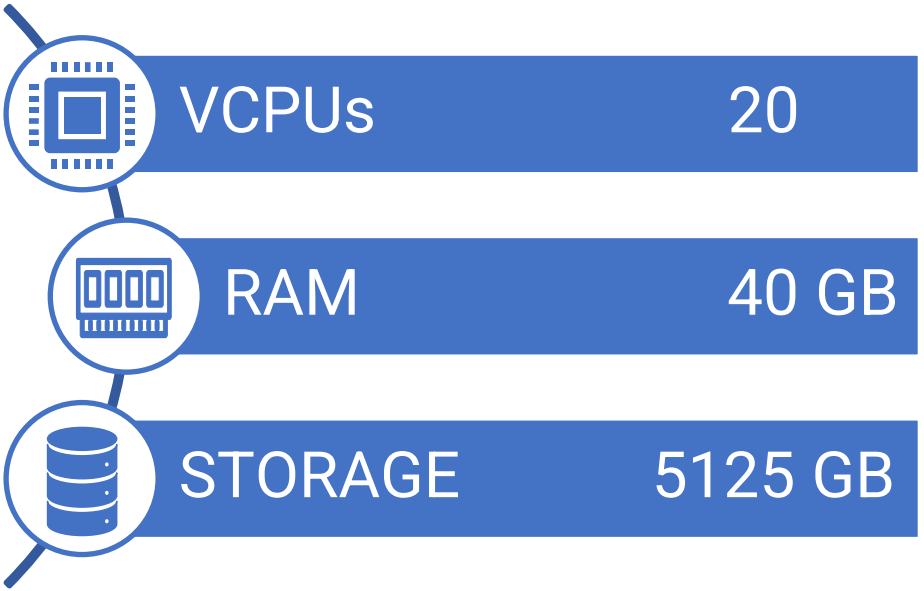



Storage &
Computing resources

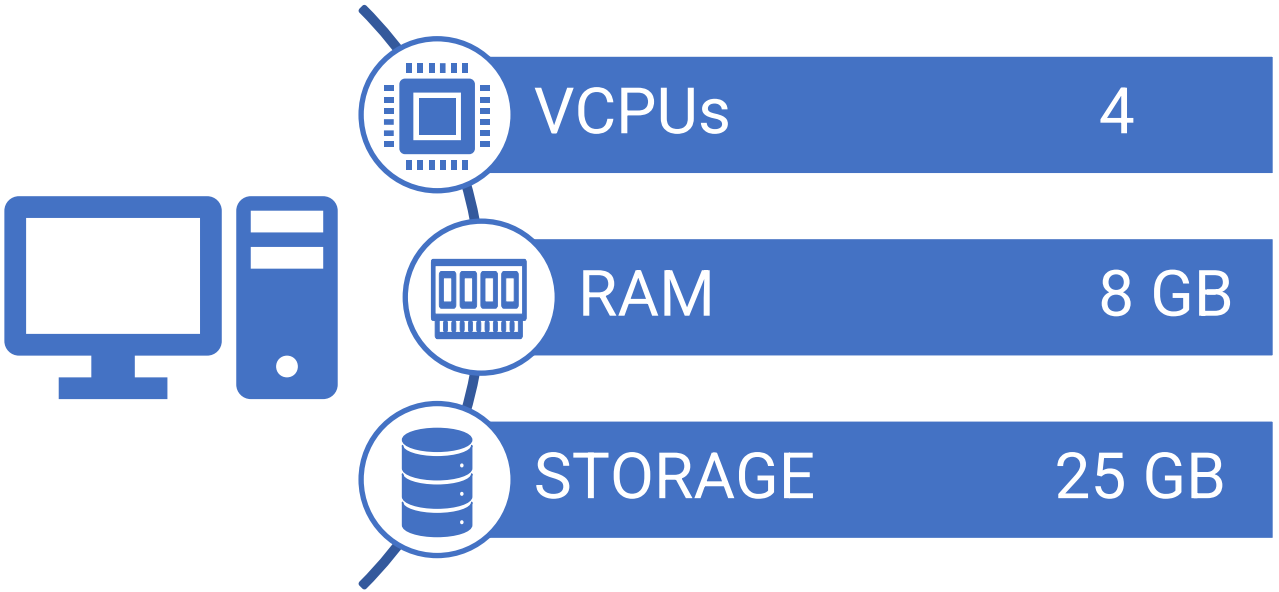
Computing resources

We made use of **CloudVeneto** resources, specifically we had at our disposal:

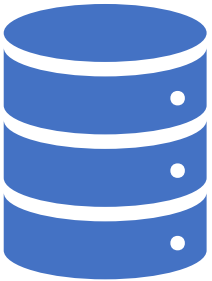
cloudveneto



Organized in **5 Virtual Machines**

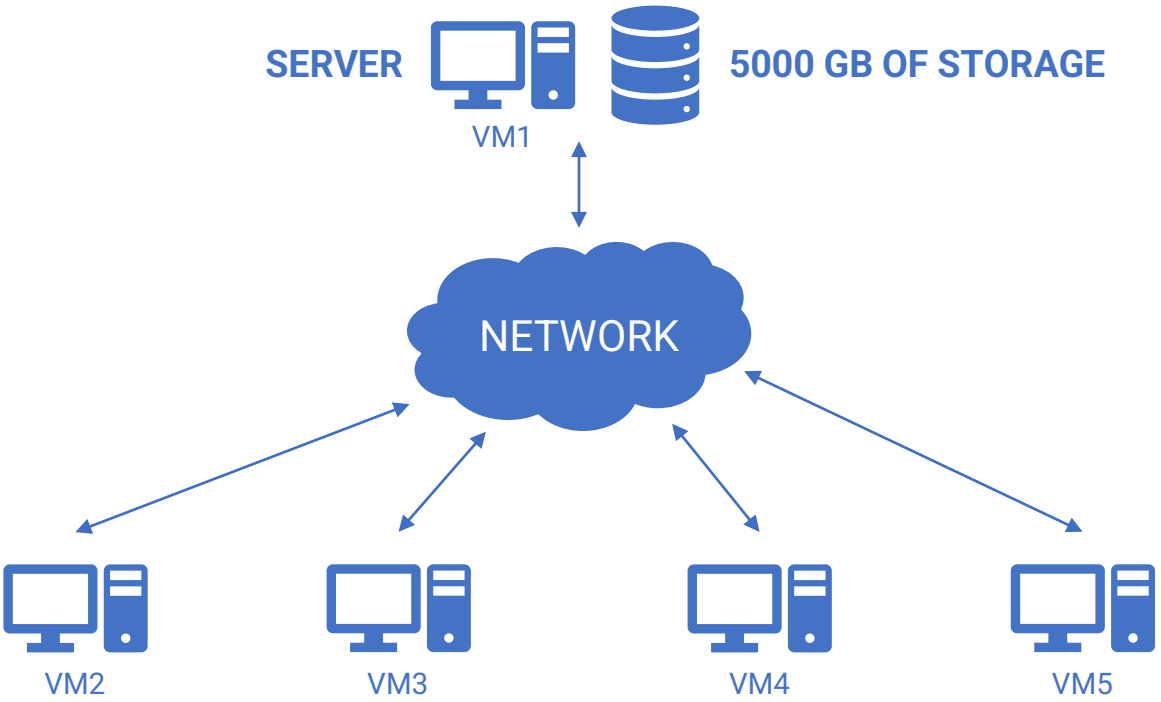


And **one volume** holding data



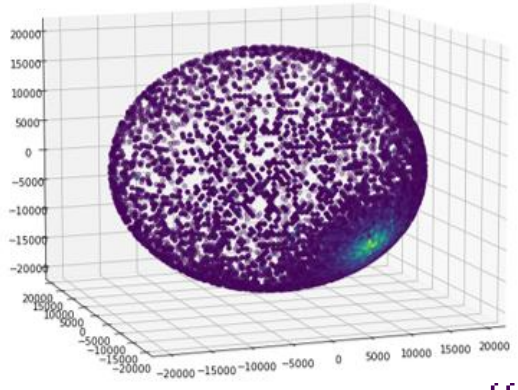
5000 GB OF STORAGE

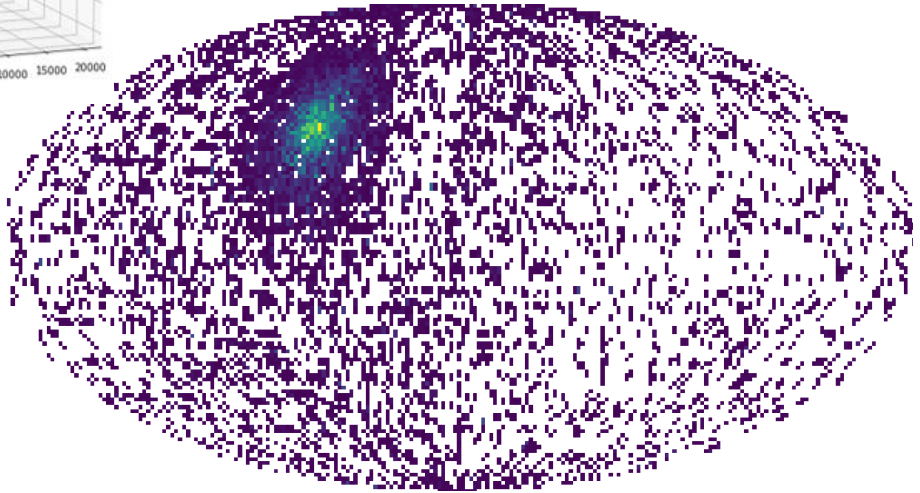
The **Network File System**



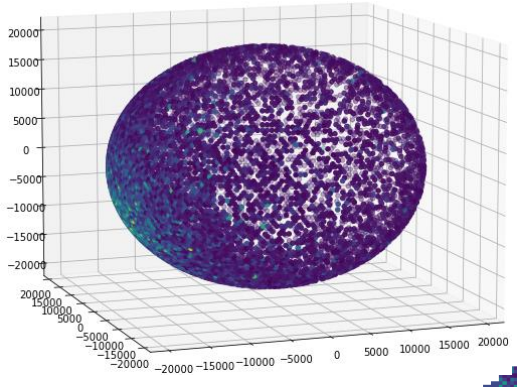
Motivations

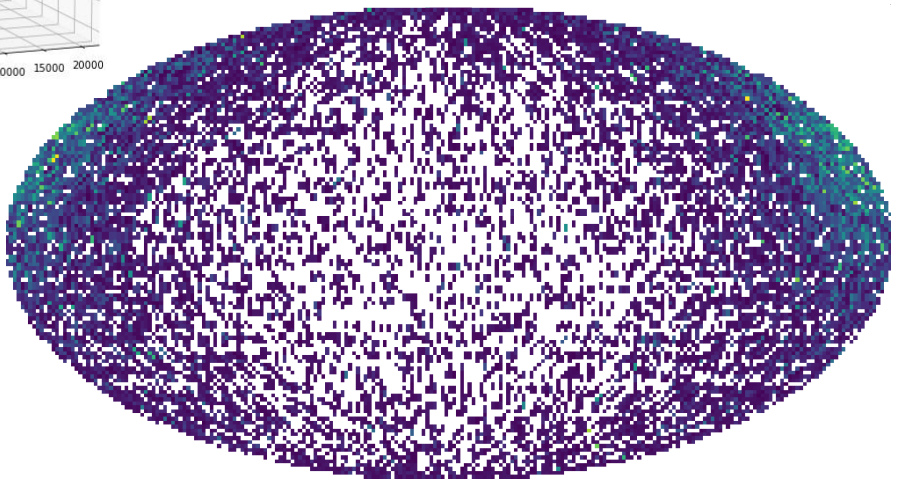
Standard projection





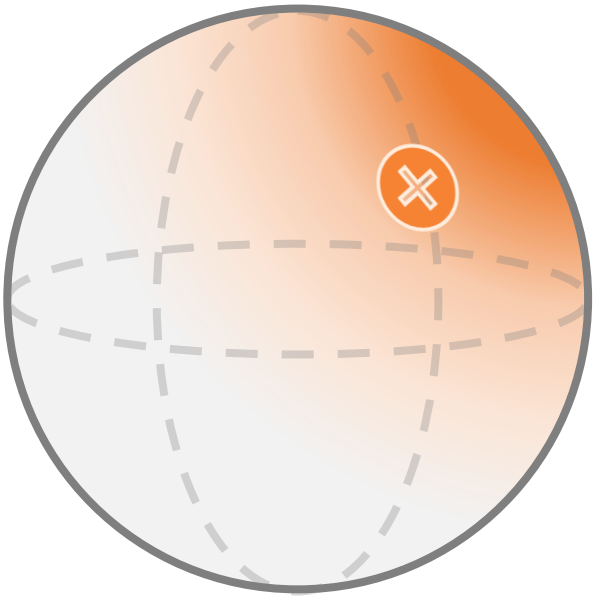
Projection **pitfalls**





Rotation & Projection methods

Processing operations



Center of charge

$$x_c = \frac{\sum_i x_i \cdot c_i}{\sum_i c_i} \qquad y_c = \frac{\sum_i y_i \cdot c_i}{\sum_i c_i} \qquad z_c = \frac{\sum_i z_i \cdot c_i}{\sum_i c_i}$$

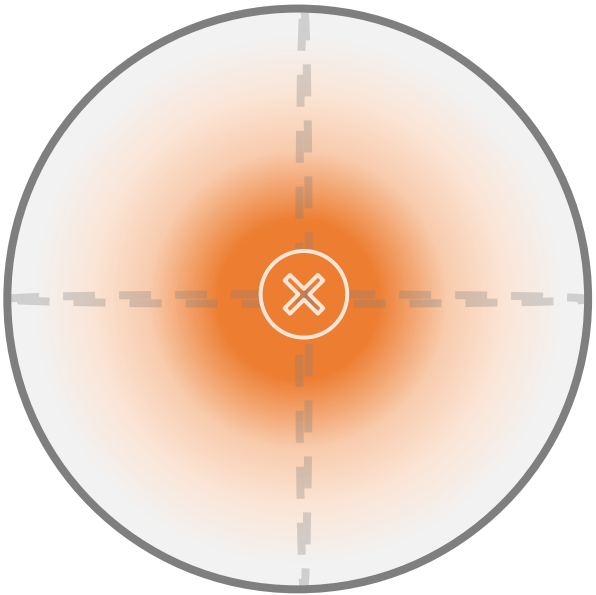
$$\varphi_C = \arctan\left(\frac{y}{x}\right) \qquad \theta_C = \arctan\sqrt{\frac{x_c^2 + y_c^2}{z_c}}$$

Rotation

$$\phi_R = -\phi_c$$
$$\theta_R = -\theta_c + \frac{\pi}{2}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_{yz}(\theta_R, \phi_R) \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Processing operations



Center of charge

$$x_c = \frac{\sum_i x_i \cdot c_i}{\sum_i c_i} \qquad y_c = \frac{\sum_i y \cdot c_i}{\sum_i c_i} \qquad z_c = \frac{\sum_i z_i \cdot c_i}{\sum_i c_i}$$

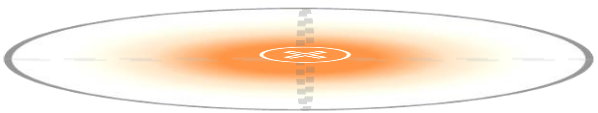
$$\varphi_C = \arctan\left(\frac{y}{x}\right) \qquad \theta_C = \arctan\sqrt{\frac{x_c^2 + y_c^2}{z_c}}$$

Rotation

$$\phi_R = -\phi_c$$
$$\theta_R = -\theta_c + \frac{\pi}{2}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_{yz}(\theta_R, \phi_R) \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Processing operations



Center of charge

$$x_c = \frac{\sum_i x_i \cdot c_i}{\sum_i c_i} \qquad y_c = \frac{\sum_i y \cdot c_i}{\sum_i c_i} \qquad z_c = \frac{\sum_i z_i \cdot c_i}{\sum_i c_i}$$

$$\varphi_C = \arctan\left(\frac{y}{x}\right) \qquad \theta_C = \arctan\sqrt{\frac{x_c^2 + y_c^2}{z_c}}$$

Rotation

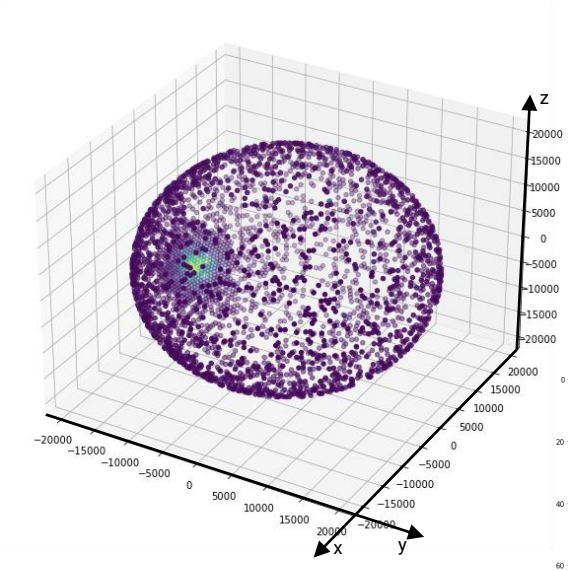
$$\phi_R = -\phi_c$$
$$\theta_R = -\theta_c + \frac{\pi}{2}$$

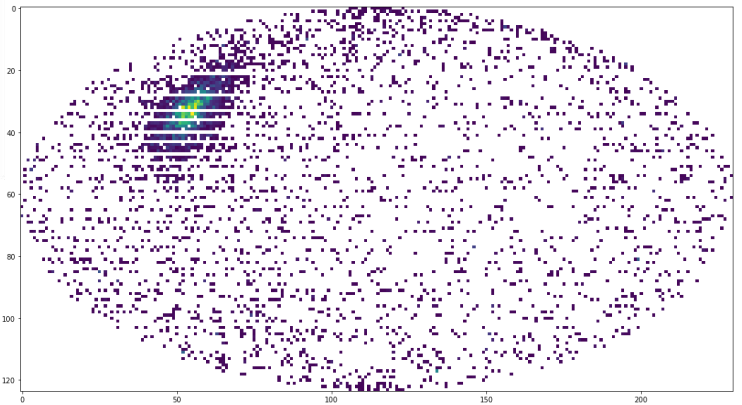
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_{yz}(\theta_R, \phi_R) \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Projection

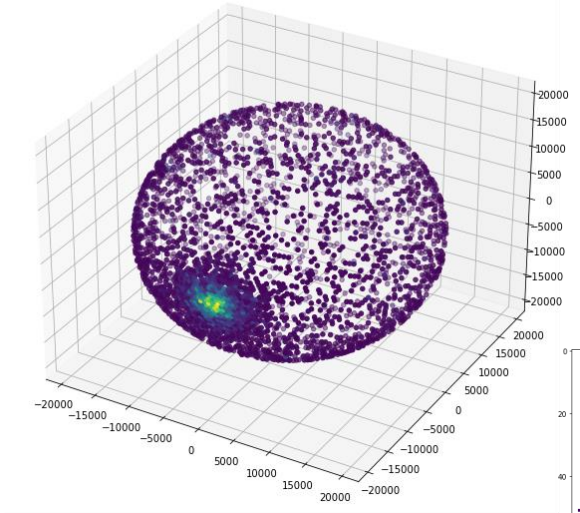
$$i_x = \left[N_{eff} \cdot \frac{\tan^{-1}(x/y)}{\pi} \right] + \frac{N_{max}}{2}$$

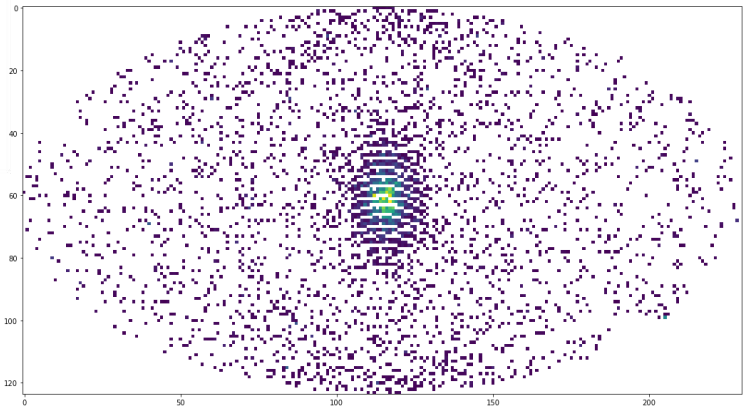
Standard mapping

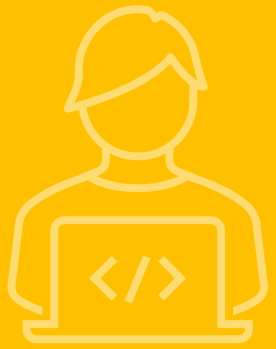




Rotated event projection







Distributed algorithm implementation

Cluster setup

VM1

VM2

VM3

VM4

VM5

Cluster setup

```
cluster = SSHCluster(
    ["10.67.22.39", "10.67.22.74", "10.67.22.27", "10.67.22.91", "10.67.22.60"],
    connect_options    = {"known_hosts": "/root/.ssh/known_hosts"},
    worker_options     = {"nthreads": n_threads_wk, "n_workers": n_workers_vm},
    scheduler_options = {"dashboard_address": ":8787"}
)
```

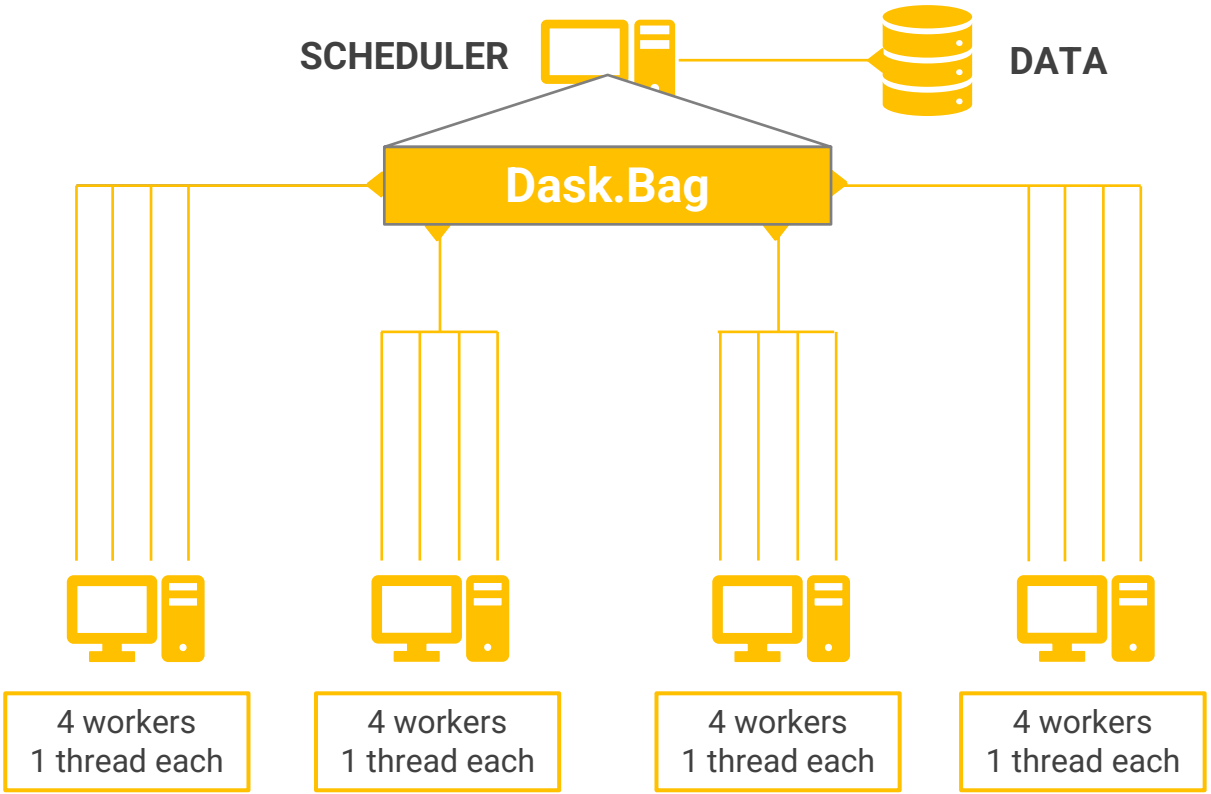
4 workers
1 thread each

4 workers
1 thread each

4 workers
1 thread each

4 workers
1 thread each

Data loading



The Distributed Processing

- *One-event* processing functions:

  Rotation : `rotate_ev()`
  Mapping : `mapping_single_event()`

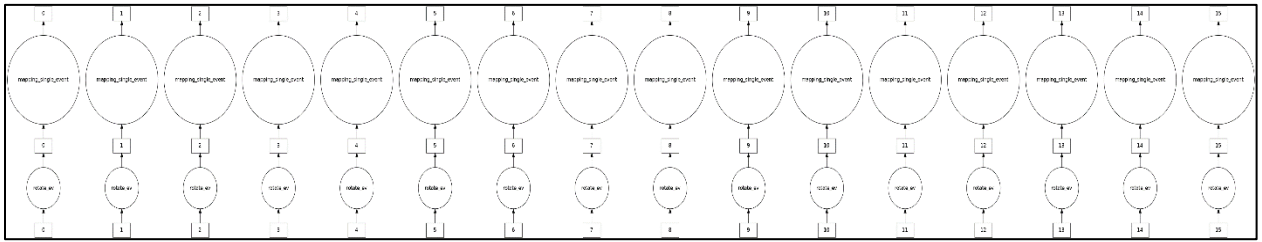
- Processing distribution through

  `Dask.Bag.map()`

Lazy!

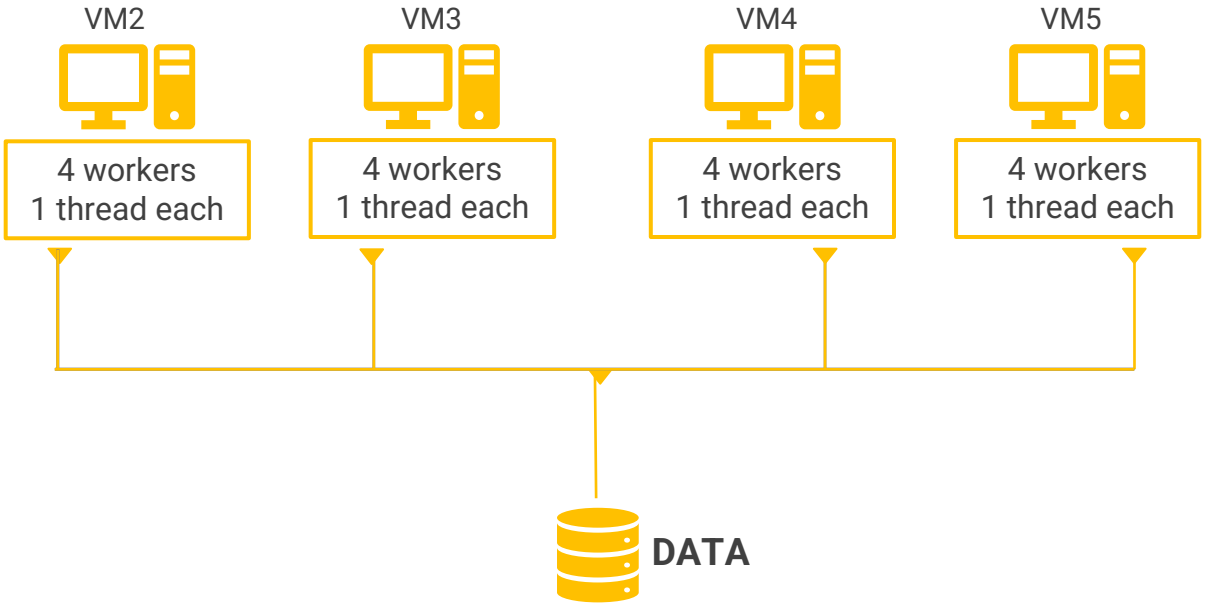
```
rotated = db.map(rotate_ev, data_db)
mapped  = db.map(mapping_single_event, rotated)
```

```
mapped.visualize()
```



Compute and Saving

```
images = mapped.compute()
```



Time performance benchmarks

The three parameters are

Workers

Number of workers to spawn on a single virtual machine

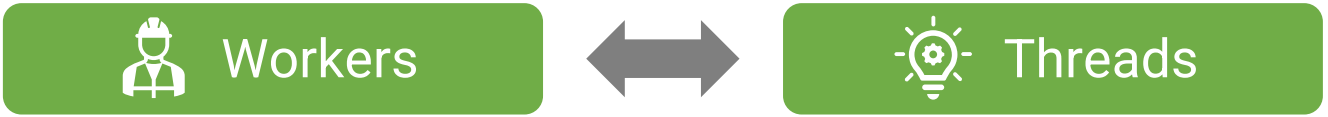
Threads

Size of the thread pool within a single worker process

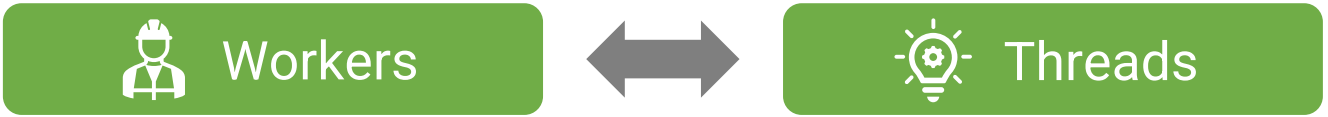
Partitions

The granularity of data parallelism

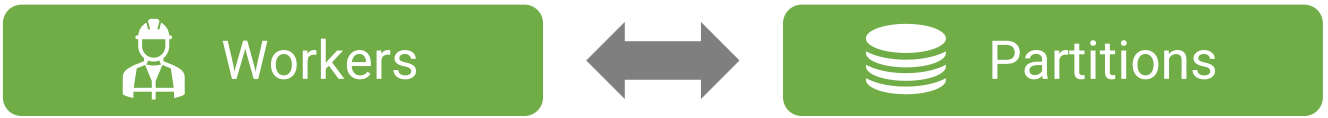
Benchmarks outline



Keeping the number of partitions fixed to sixteen

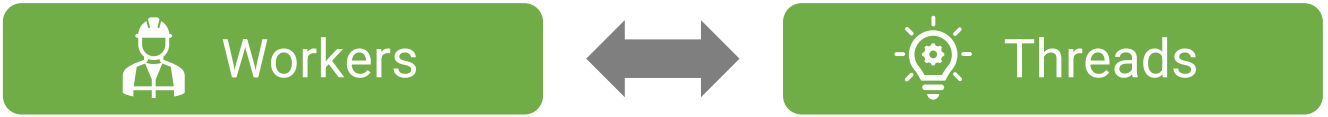


Setting the number of partitions equal to the total number of workers



Keeping the thread pool fixed to one single thread per worker

Benchmarks outline



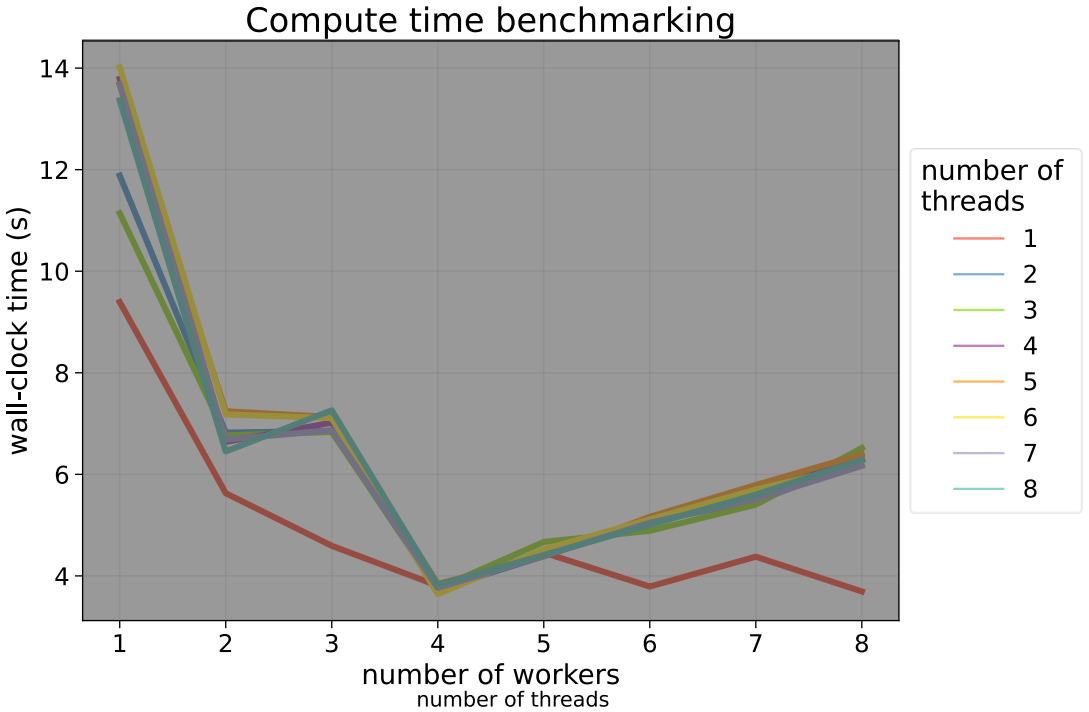
Keeping the number of partitions fixed to sixteen

Setting the number of partitions equal to the total number of workers

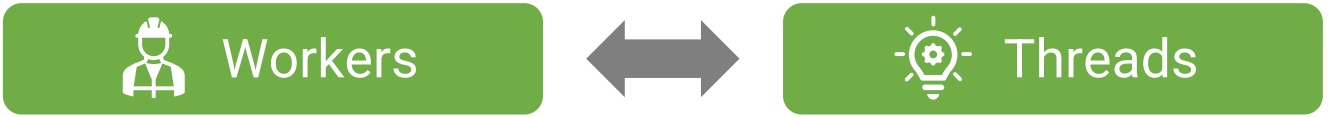
Keeping the thread pool fixed to one single thread per worker

Workers ⟷ Threads

Keeping the number of partitions fixed to sixteen



Benchmarks outline



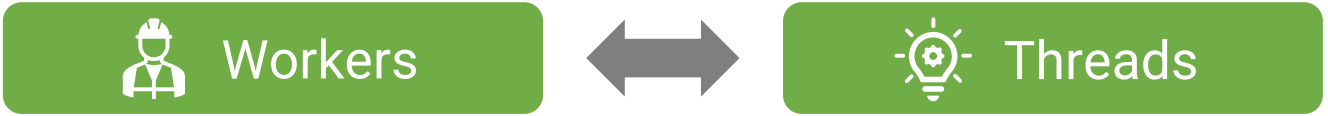
Keeping the number of partitions fixed to sixteen

Setting the number of partitions equal to the total number of workers

Keeping the thread pool fixed to one single thread per worker

Benchmarks outline

Keeping the number of partitions fixed to sixteen

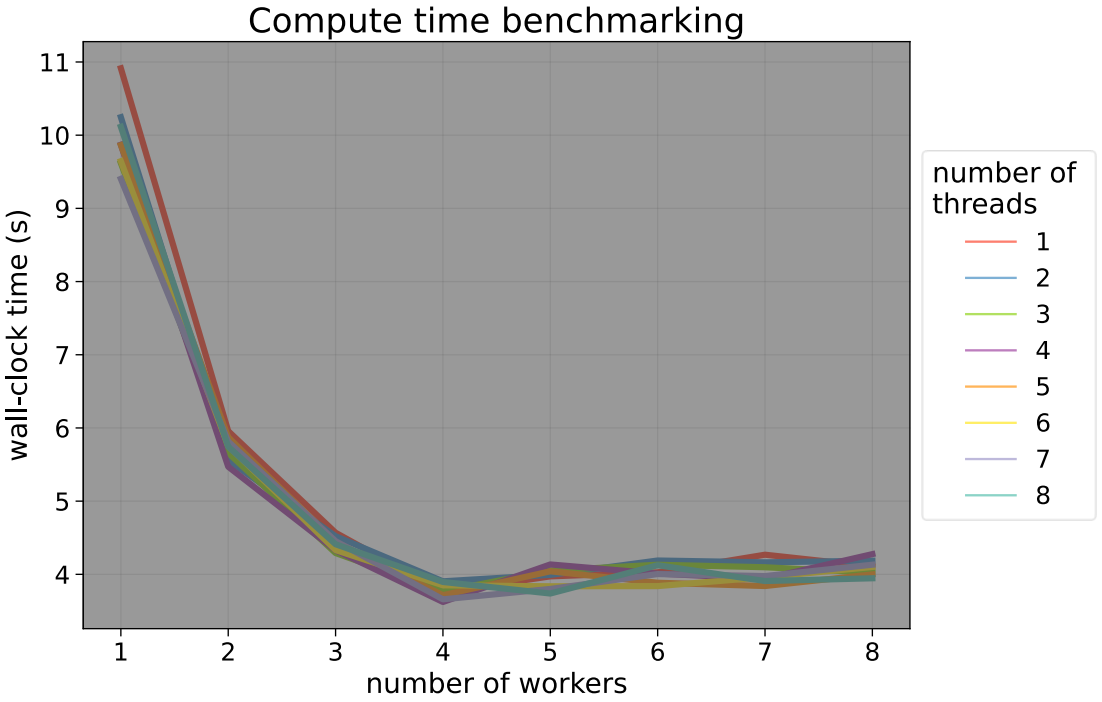


Setting the number of partitions equal to the total number of workers

Keeping the thread pool fixed to one single thread per worker

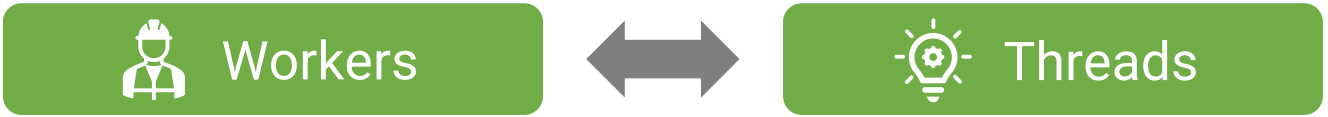
Workers ⟷ Threads

Setting the number of partitions equal to the total number of workers



Benchmarks outline

Keeping the number of partitions fixed to sixteen

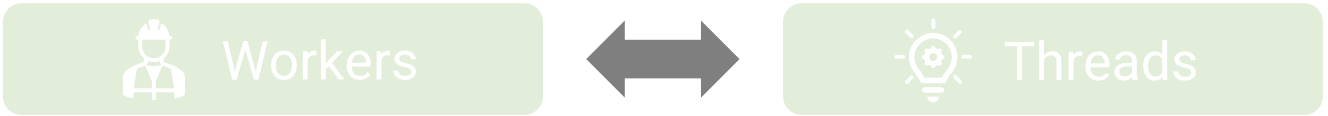


Setting the number of partitions equal to the total number of workers

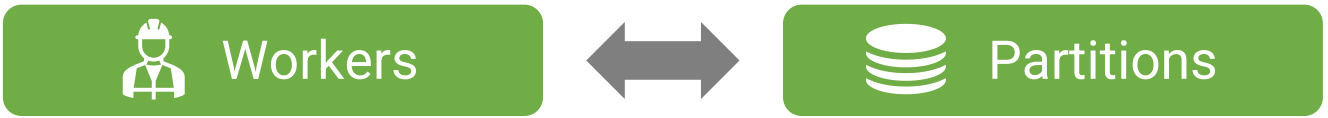
Keeping the thread pool fixed to one single thread per worker

Benchmarks outline

Keeping the number of partitions fixed to sixteen



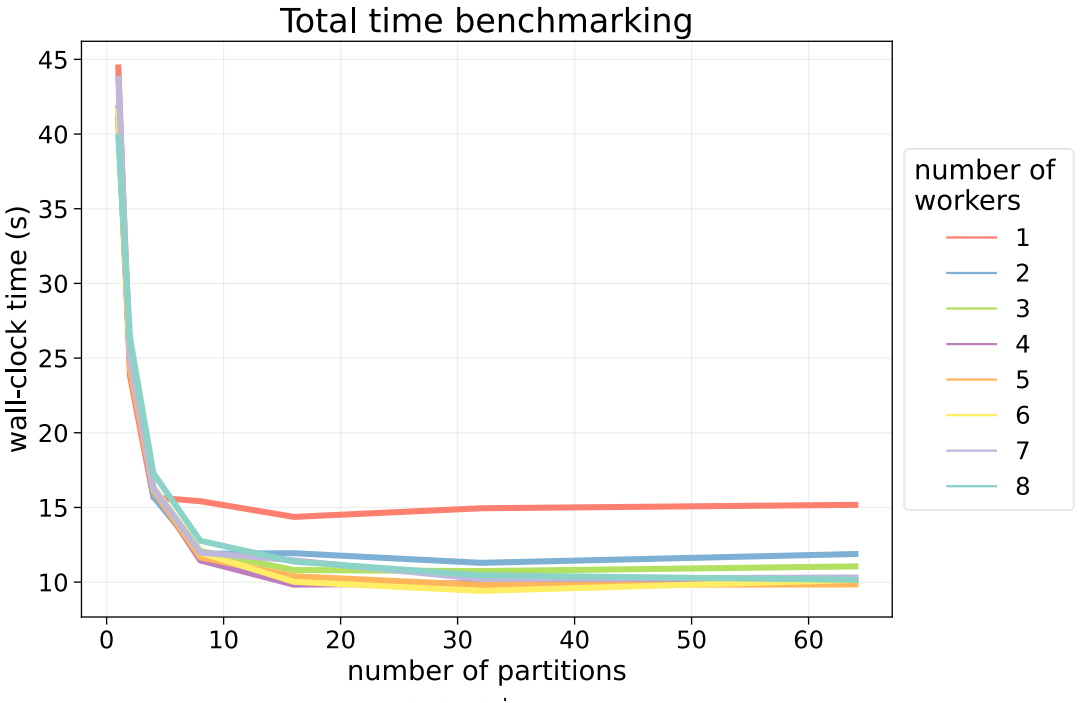
Setting the number of partitions equal to the total number of workers

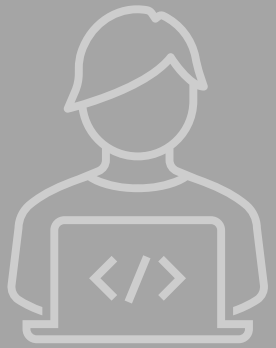


Keeping the thread pool fixed to one single thread per worker

Workers ⟷ Partitions

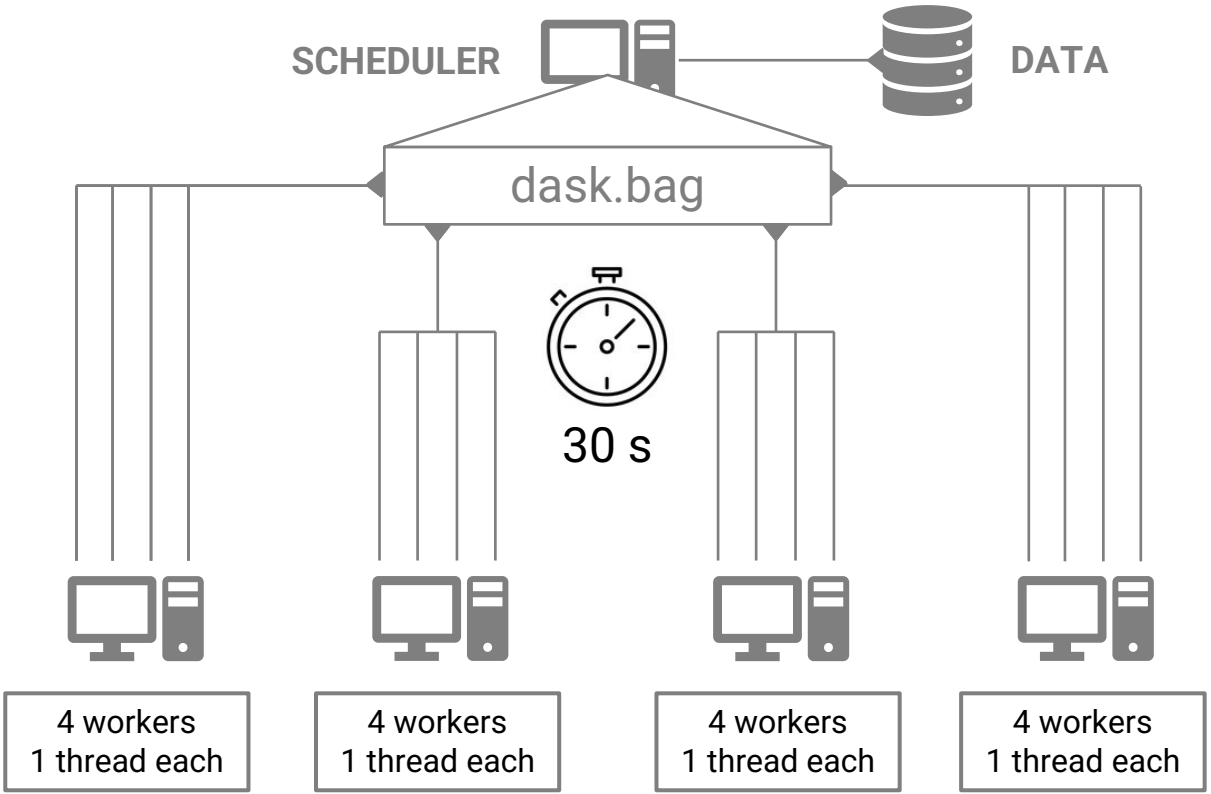
Keeping the thread pool fixed to one single thread per worker



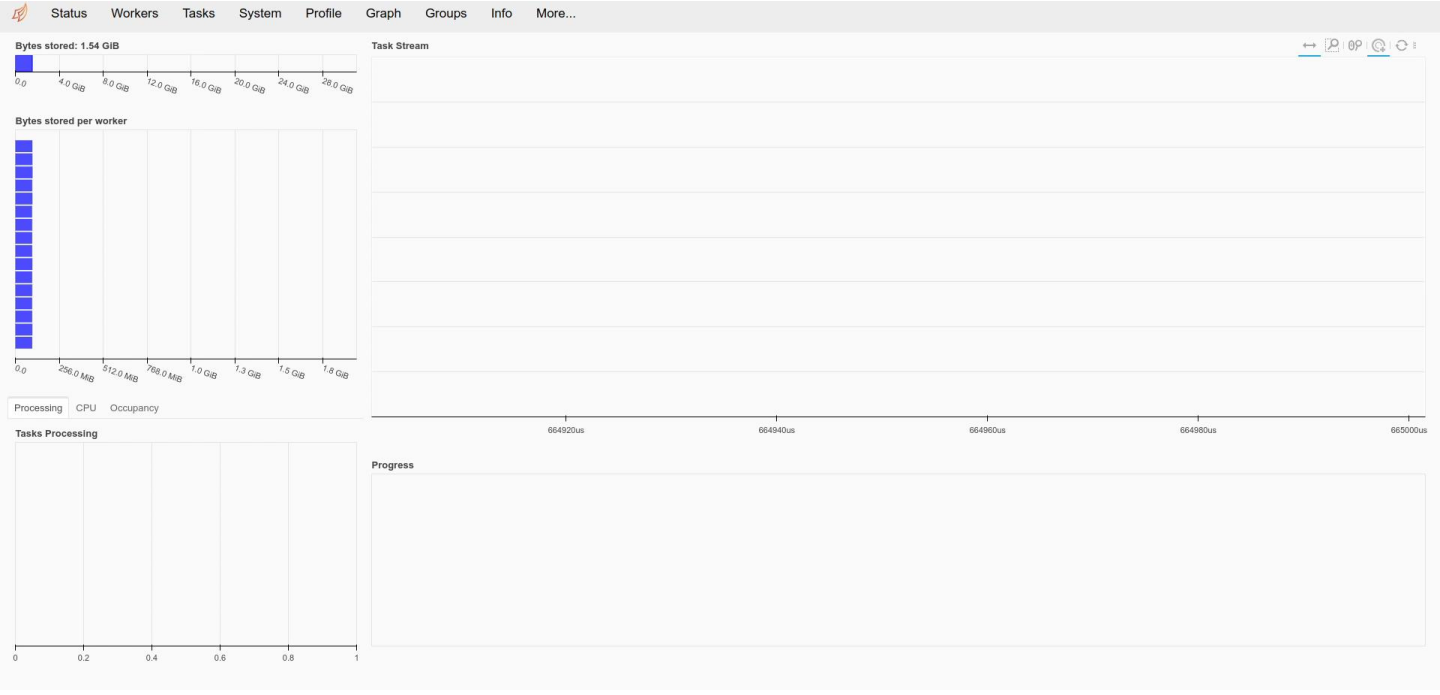


Distributed algorithm lazy implementation

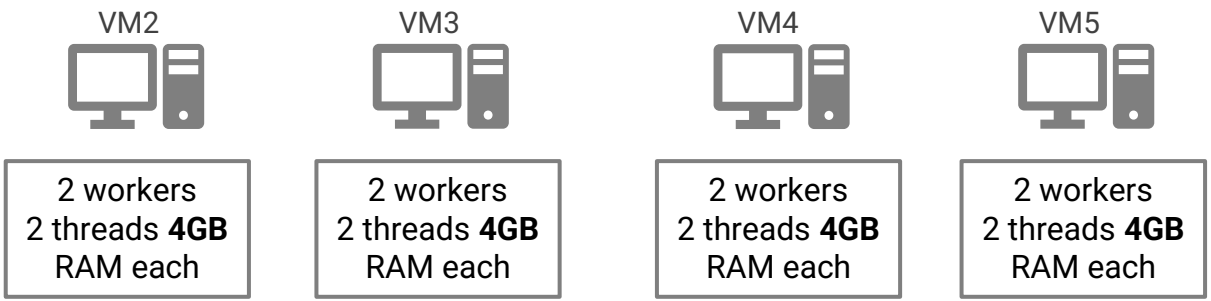
Eager implementation pitfalls



Eager implementation pitfalls



Lazy implementation cluster configuration



Lazy implementation gimmicks

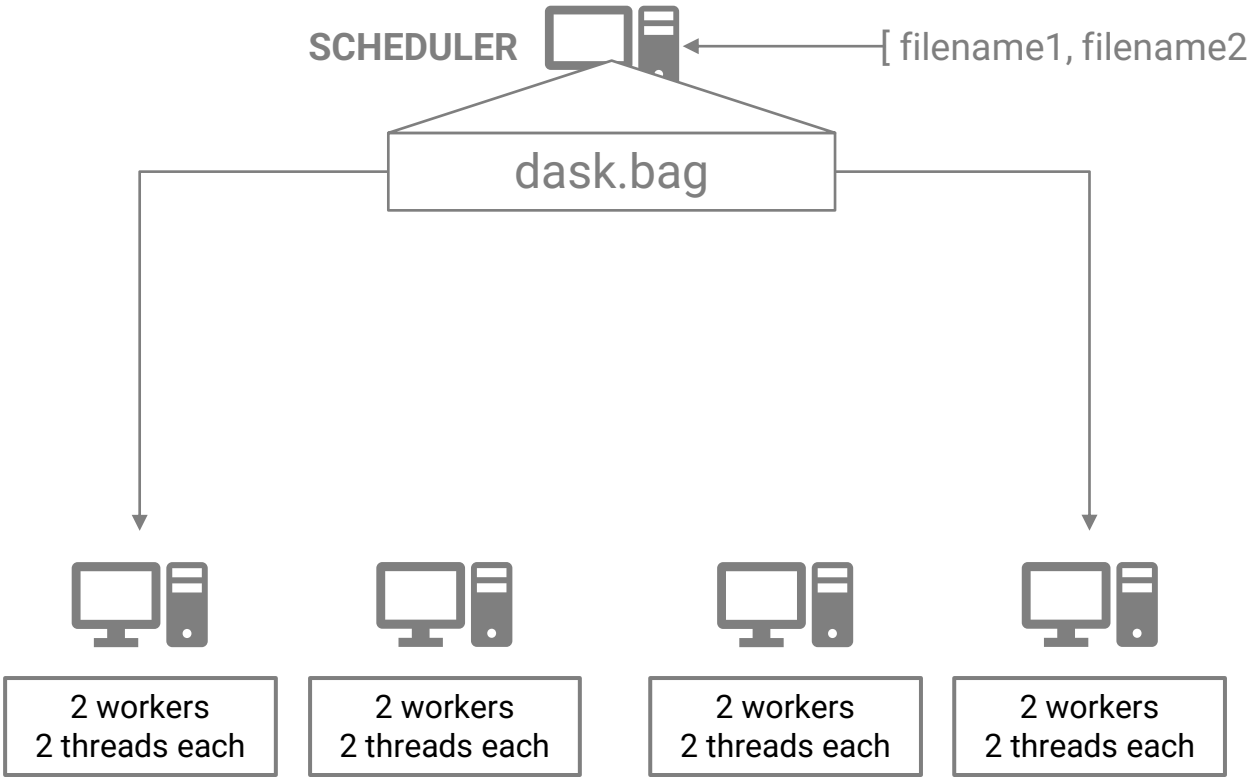
```python
def load_bag(path, Nevents):
    data_np = load(path)
    data_mask = data_np[:, :Nevents]
    bag = [np.vstack([ data_mask[j, i] for j in range(3)]) for i in range(data_mask.shape[1])]
    del data_mask
    del data_np
    return bag
lazy_load_bag = dask.delayed(load_bag)
```

- Better RAM management
- `delayed load_bag`

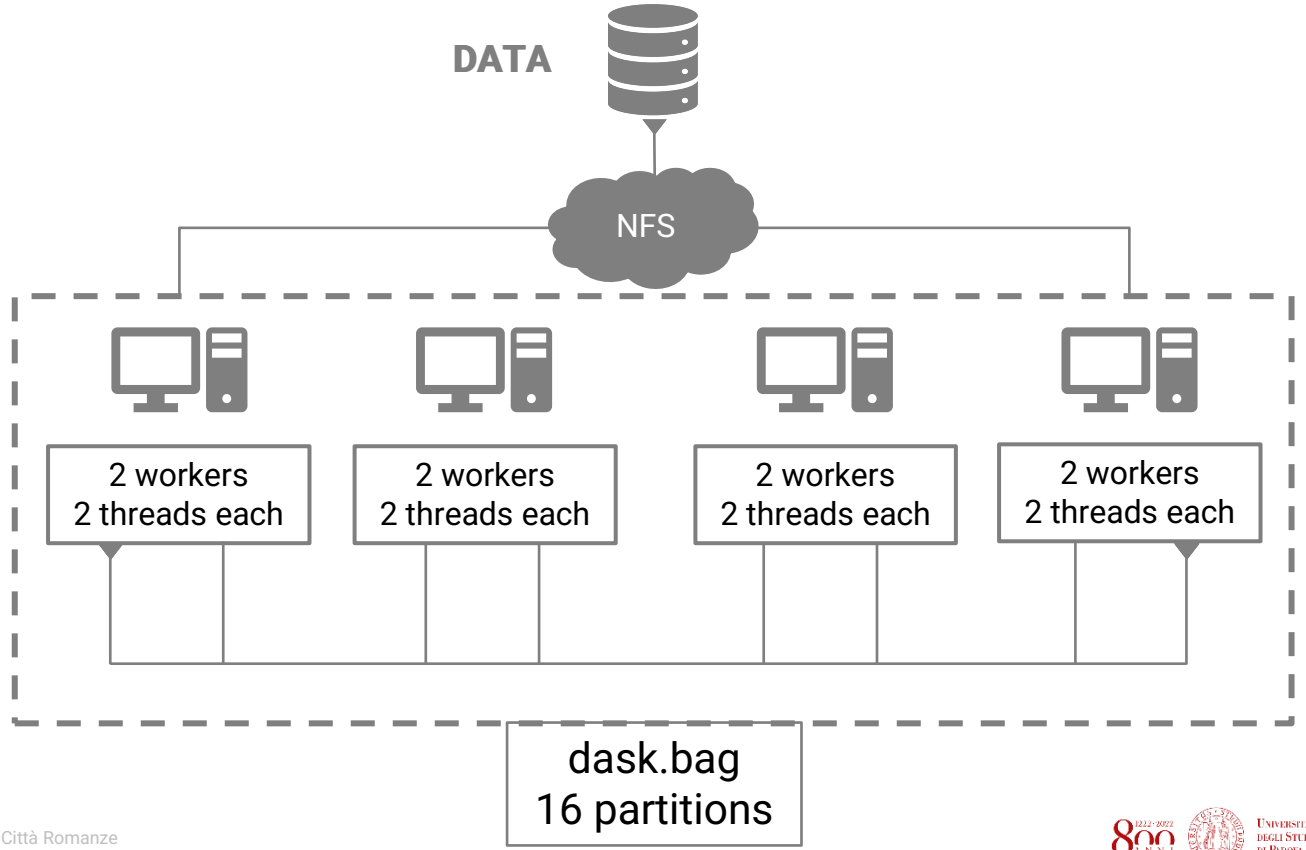
```python
lazy_loaded = db.from_delayed([lazy_load_bag(data_folder + file_name, None)
                               for file_name in name_list[i:i+nfiles_cycle]])\
                .repartition(n_workers_vm*n_threads_wk*4)
```

- `Bag` from `delayed` objects → Workers load files
- `repartition` over all available threads

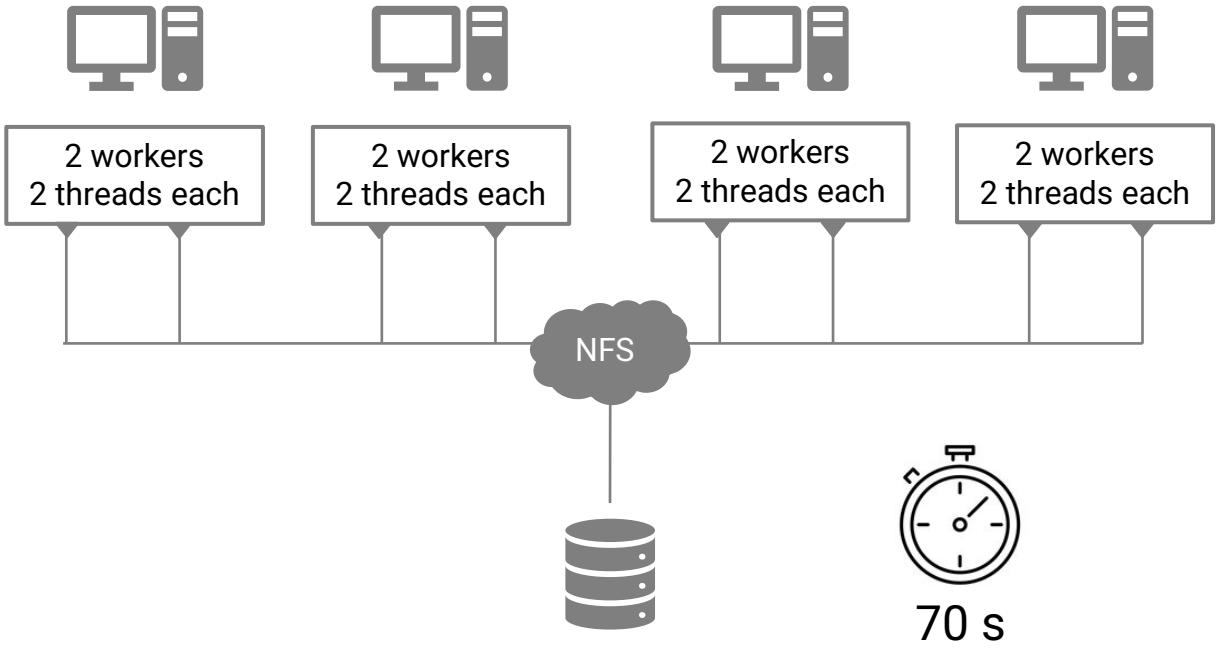
Lazy implementation



Lazy implementation



Lazy implementation



THANK YOU
FOR YOUR ATTENTION

Città Romanze

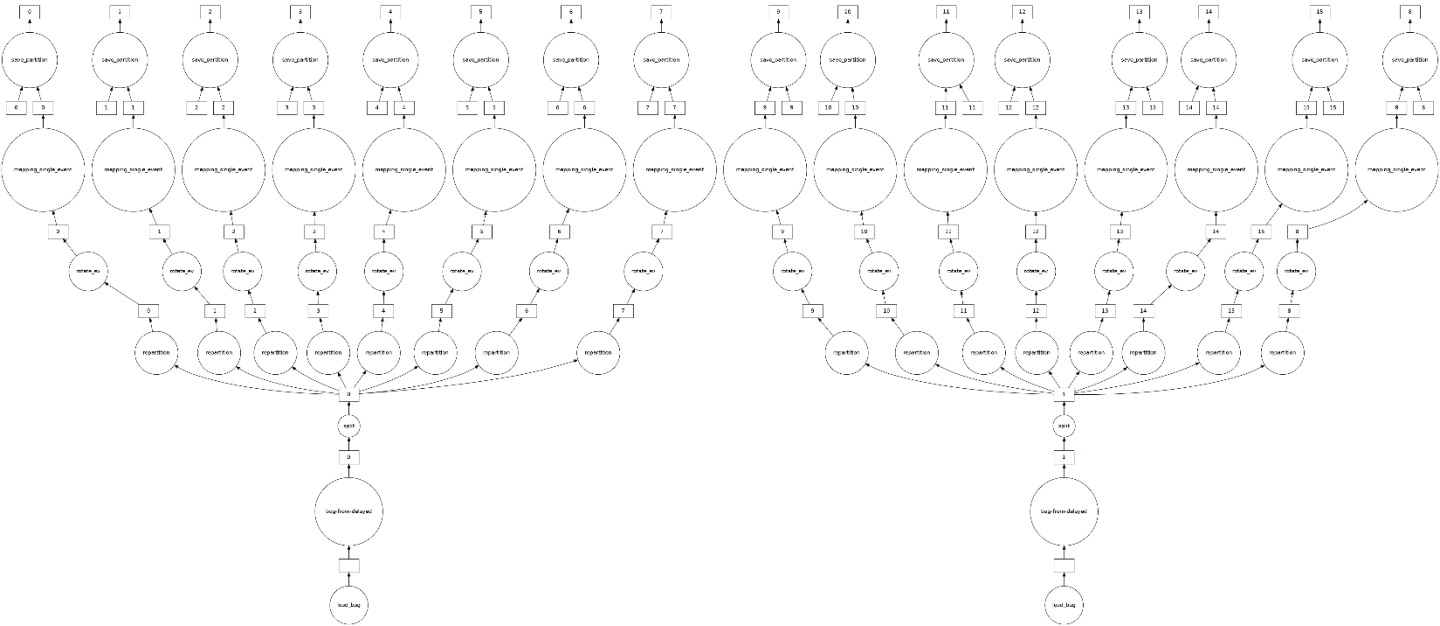
QUESTIONS

Backup Slides

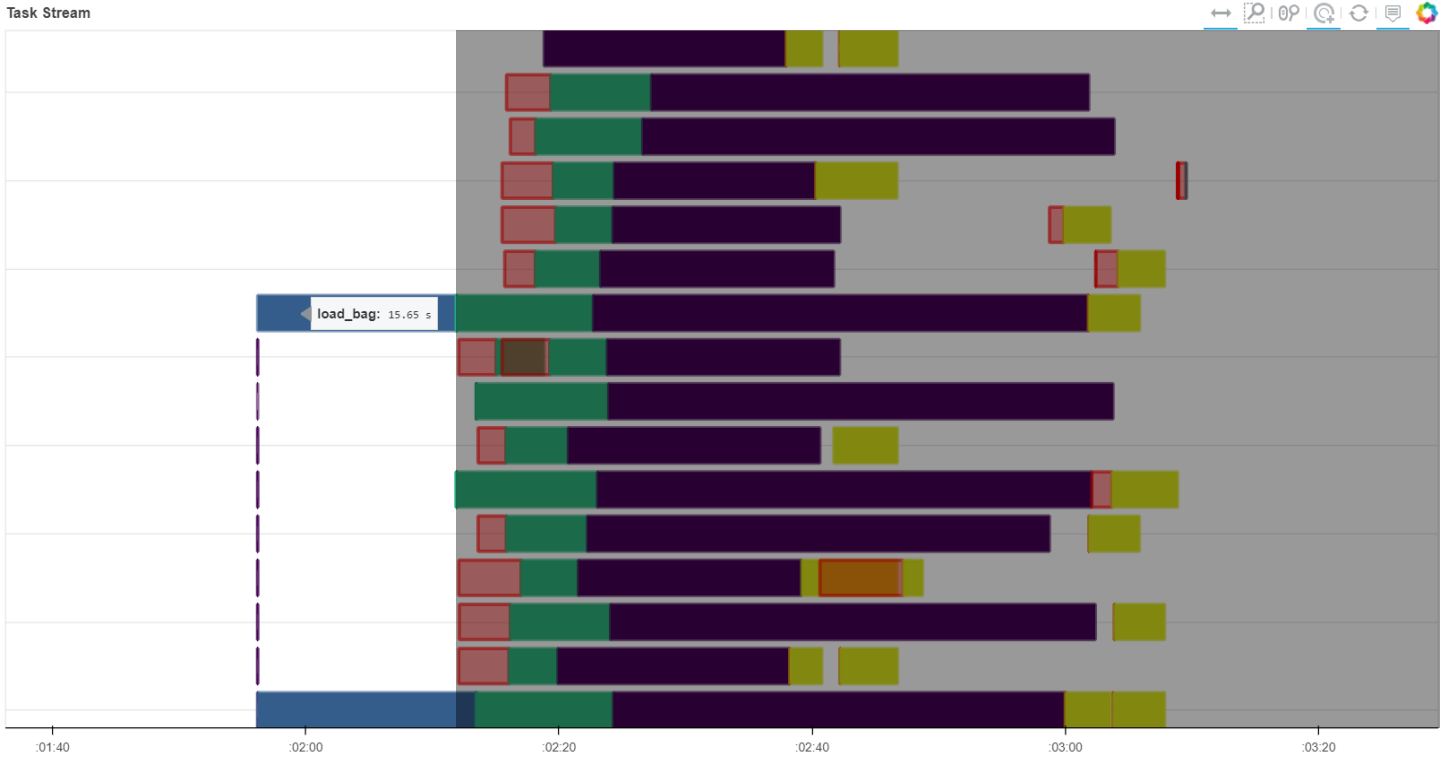
Lazy implementation gimmicks

```
1  future = client.compute(save, optimize_graph=False)
2  result = client.gather(future)
```

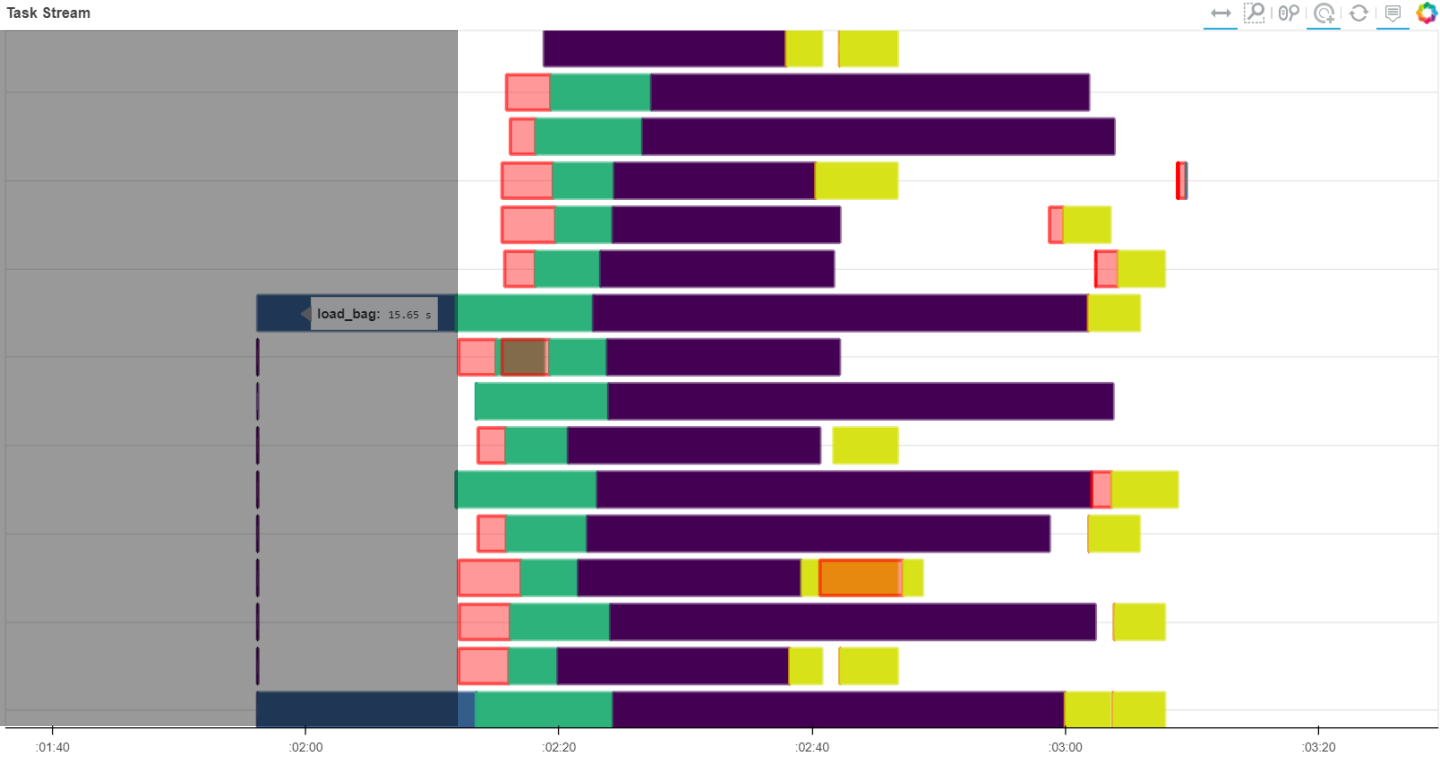
- Not optimized graph



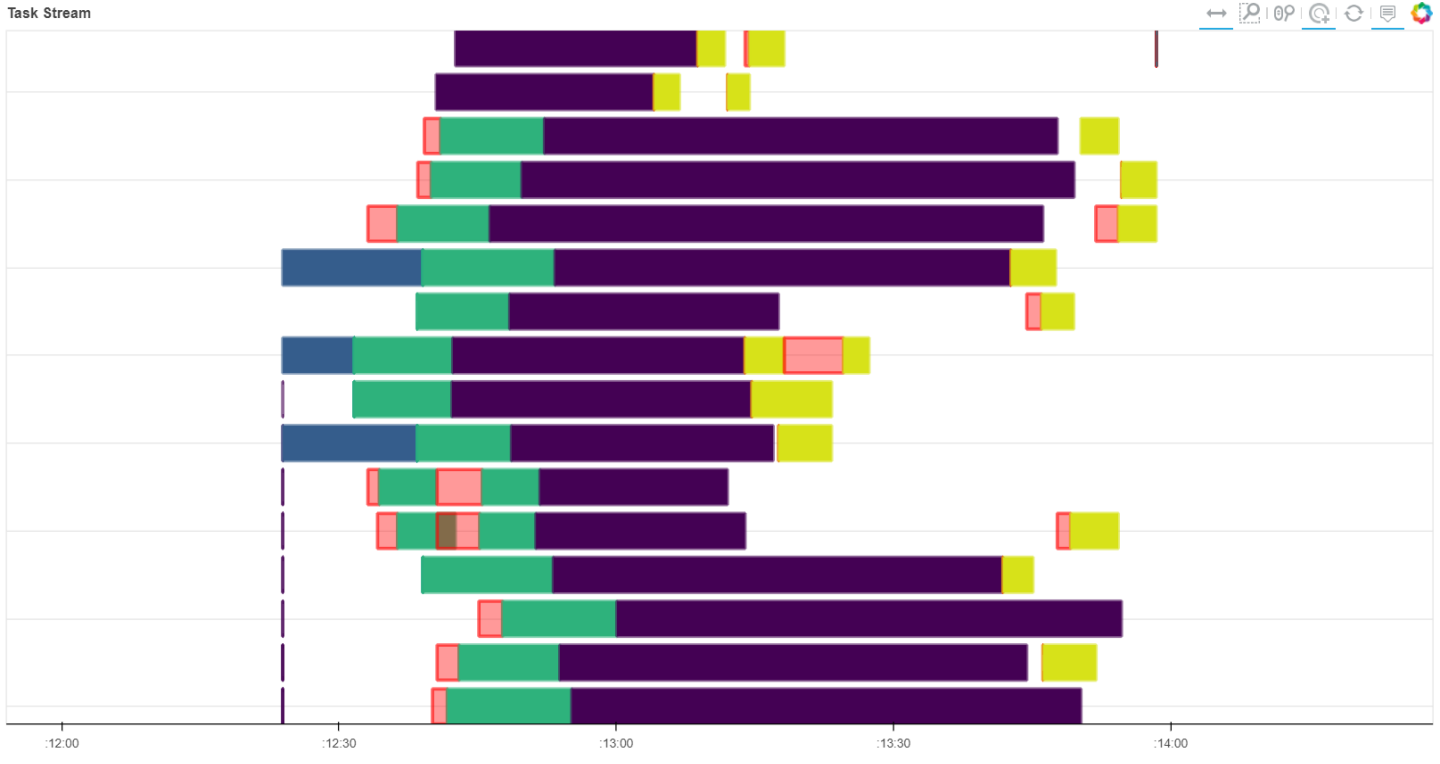
Lazy implementation run



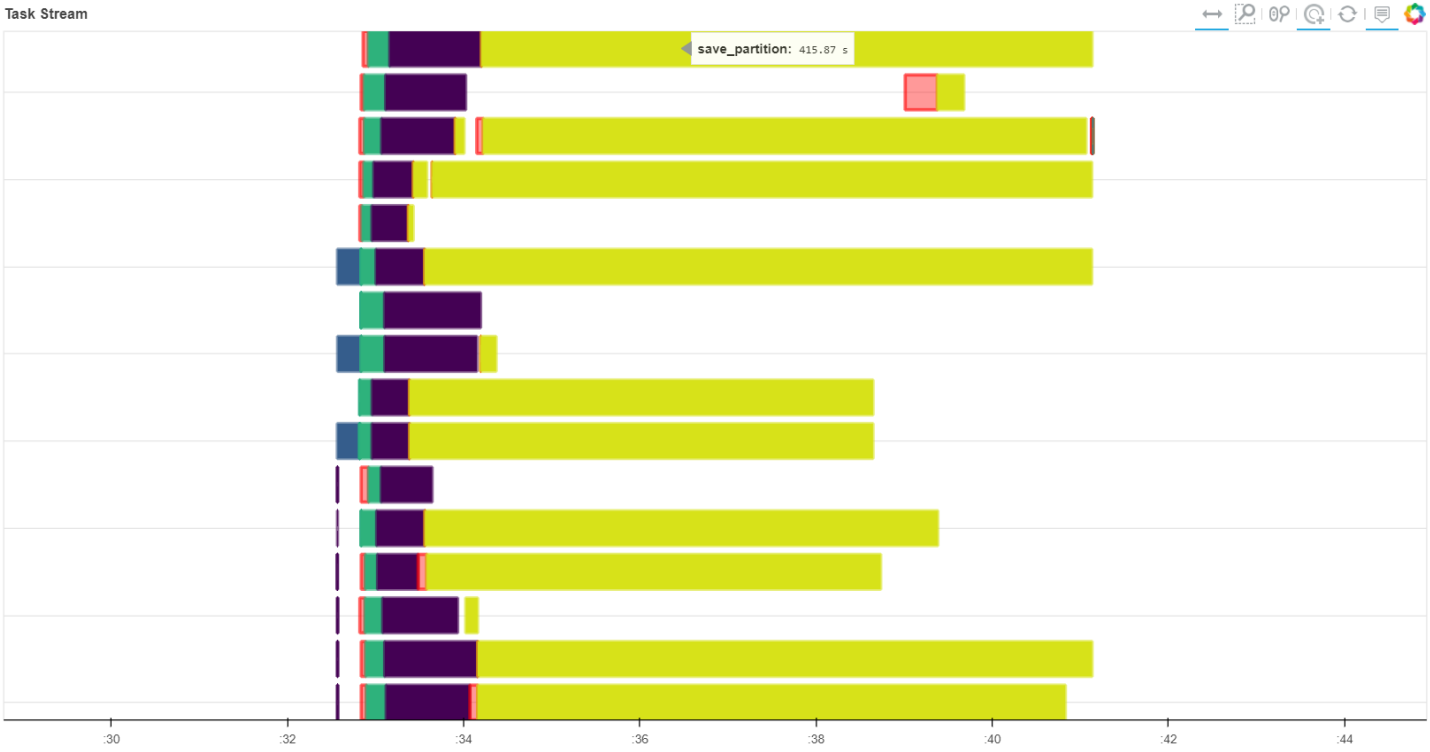
Lazy implementation run



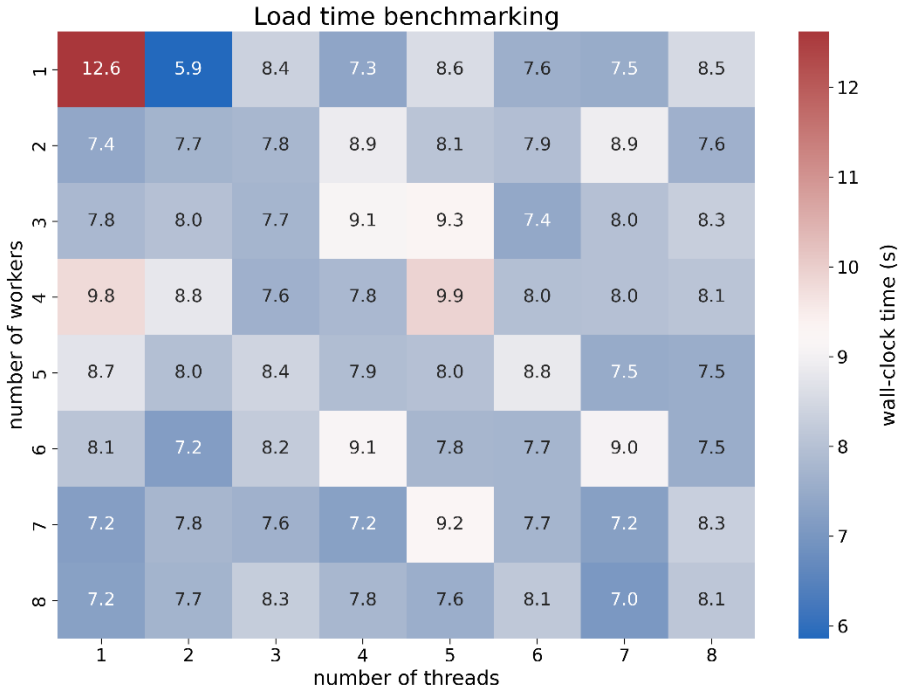
Lazy implementation pitfalls



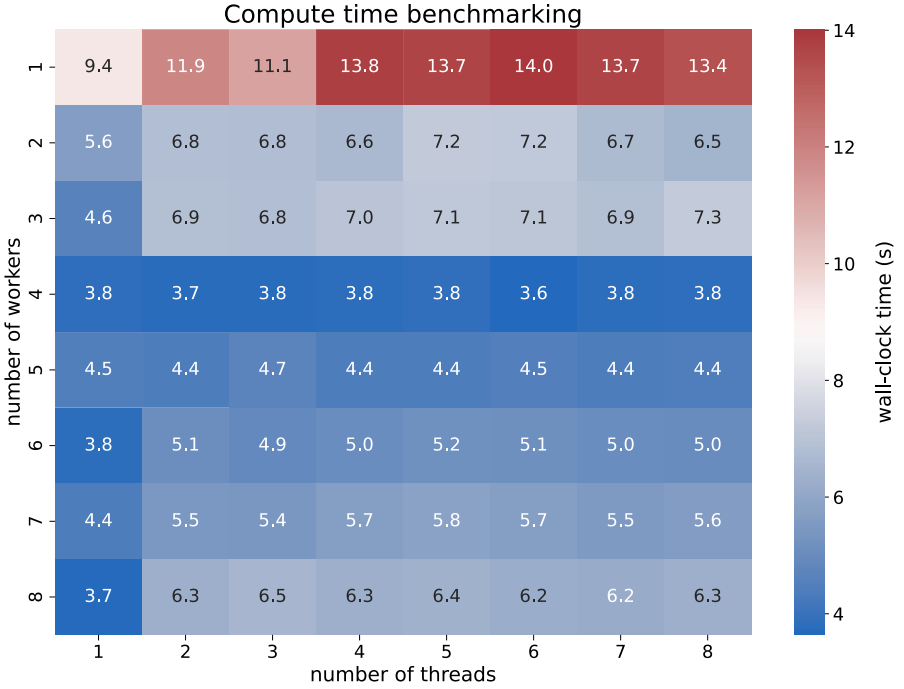
Lazy implementation pitfalls



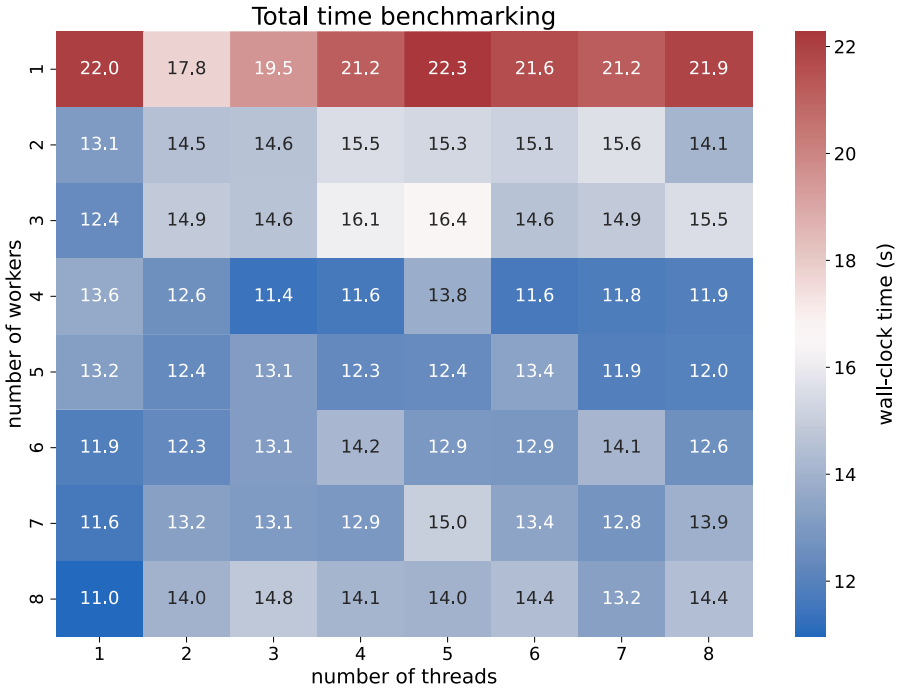
Workers vs threads – 16 partitions



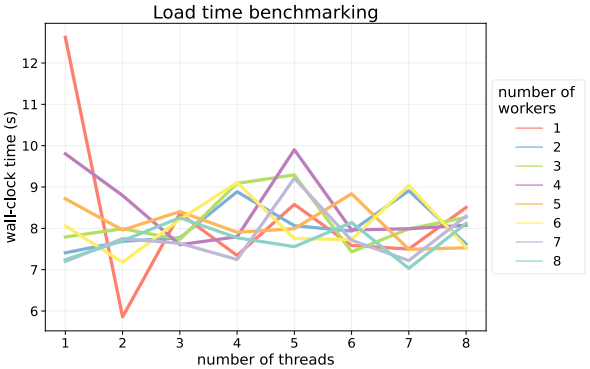
Workers vs threads – 16 partitions

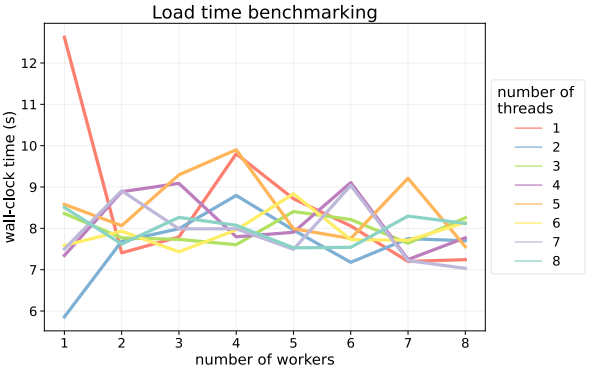


Workers vs threads – 16 partitions

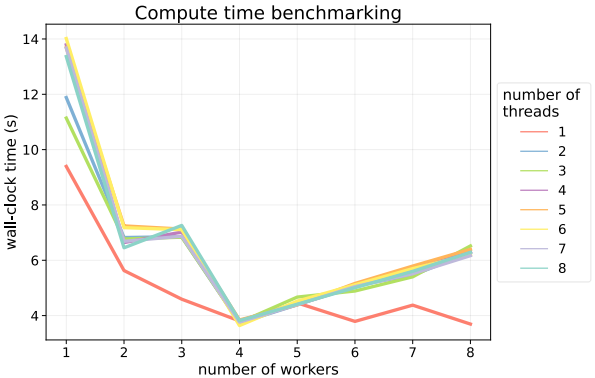


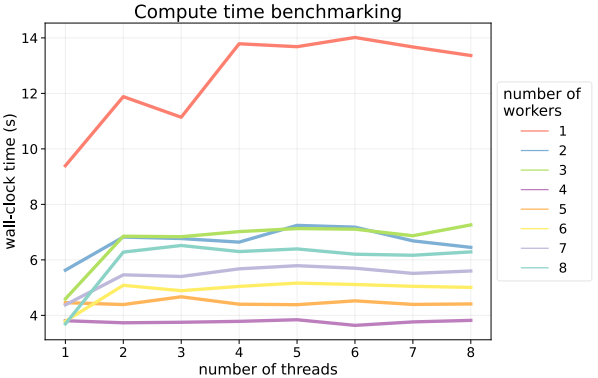
Workers vs threads – 16 partitions



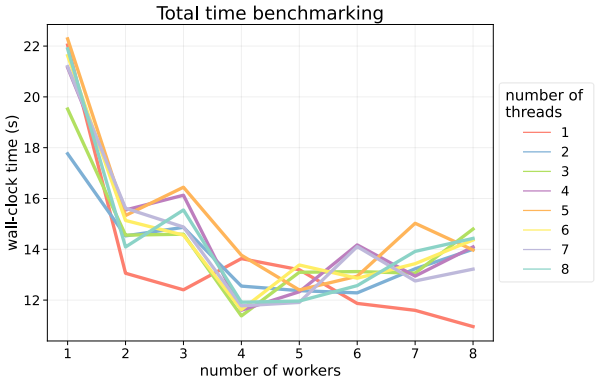


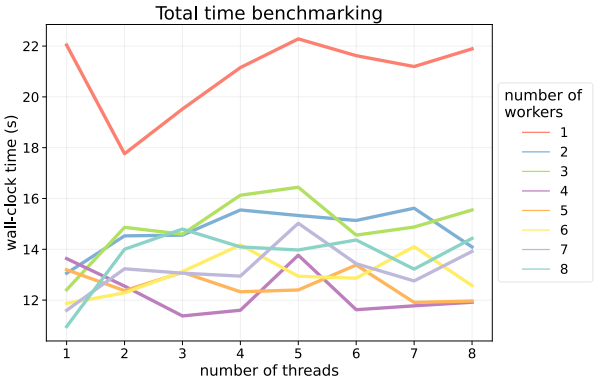
Workers vs threads – 16 partitions



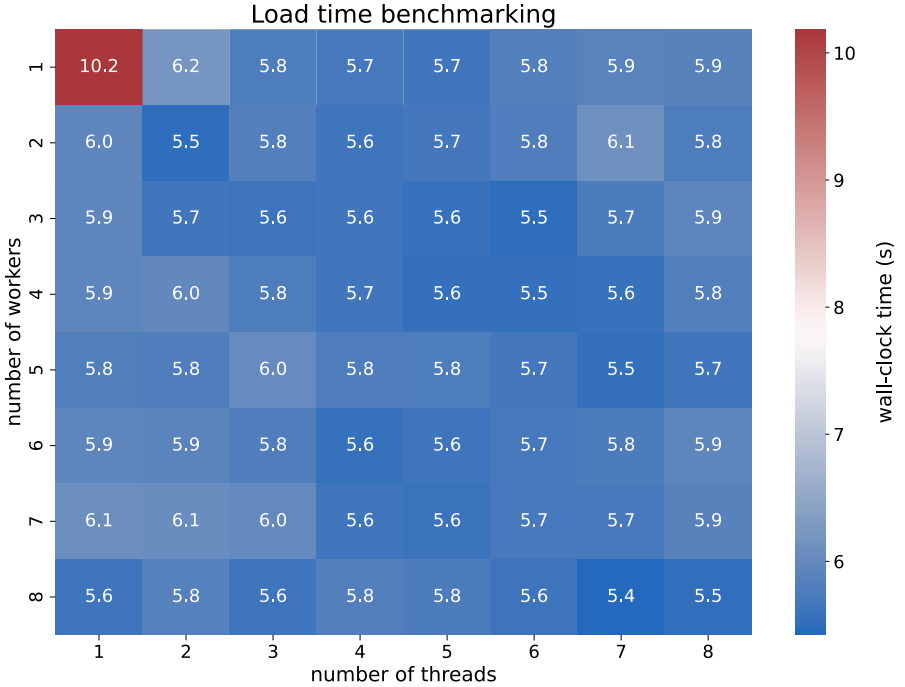


Workers vs threads – 16 partitions

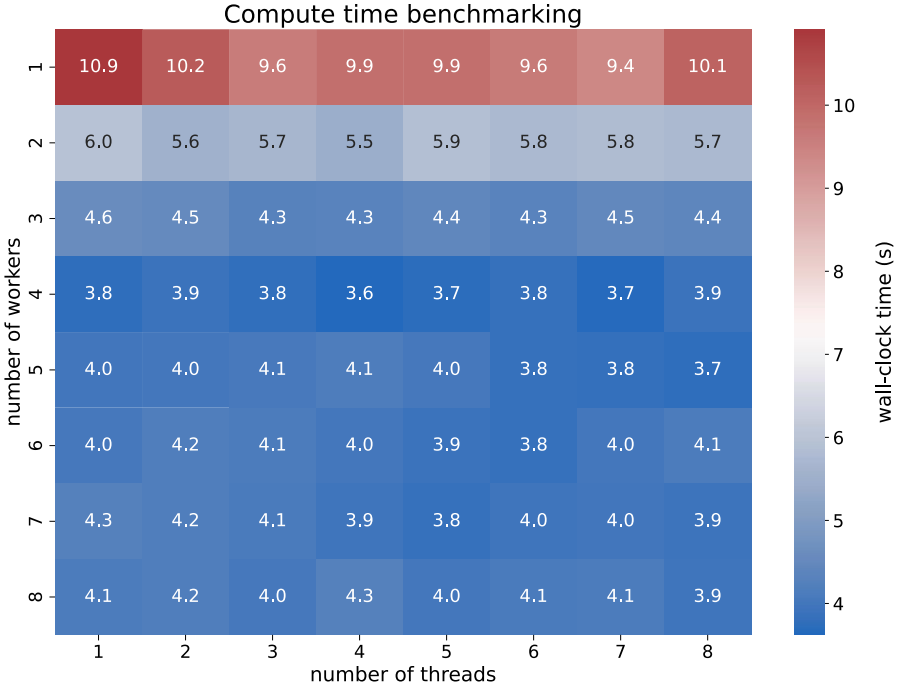




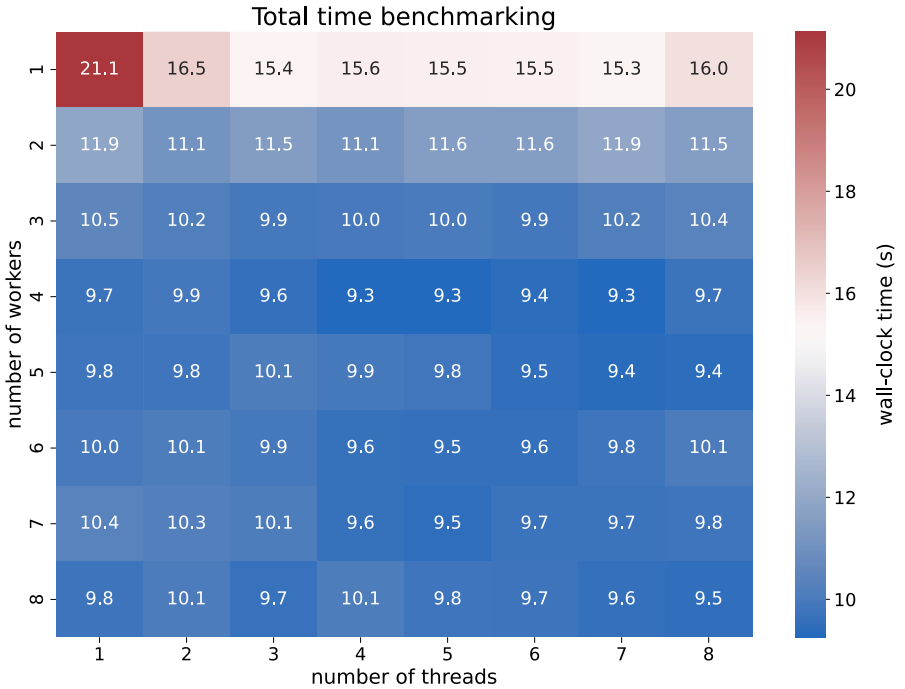
Workers vs threads – partitions = workers



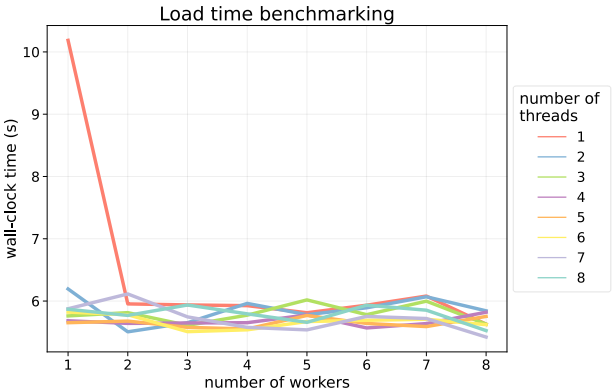
Workers vs threads – partitions = workers

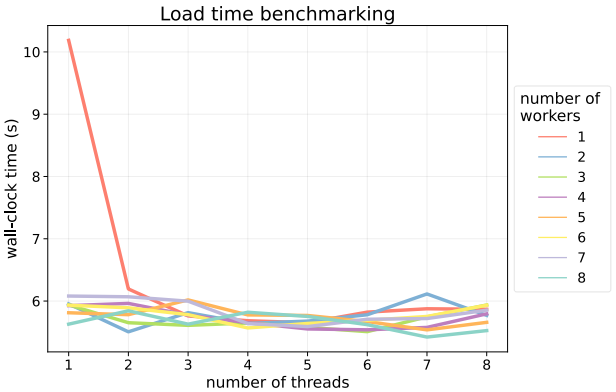


Workers vs threads – partitions = workers

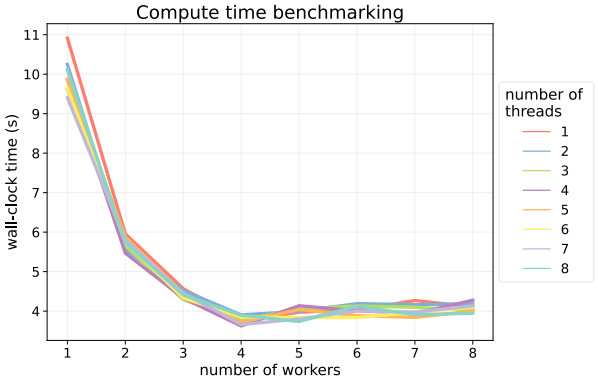


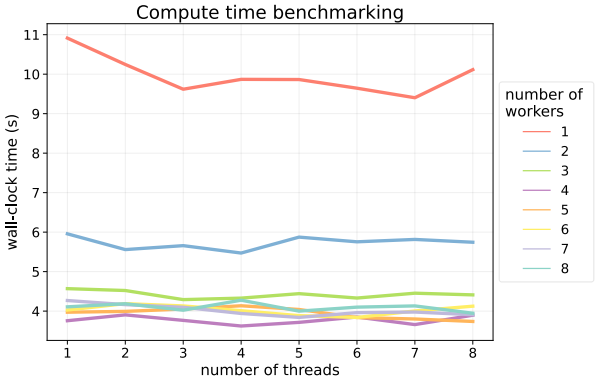
Workers vs threads – partitions = workers



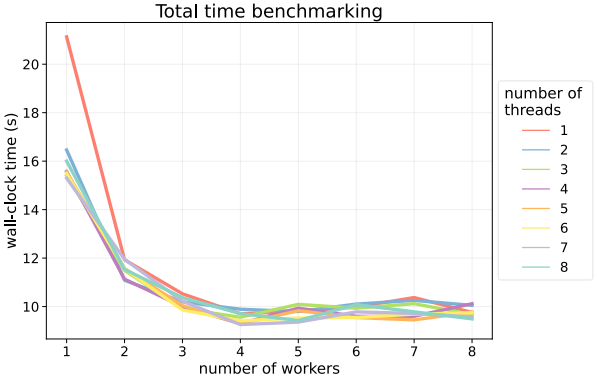


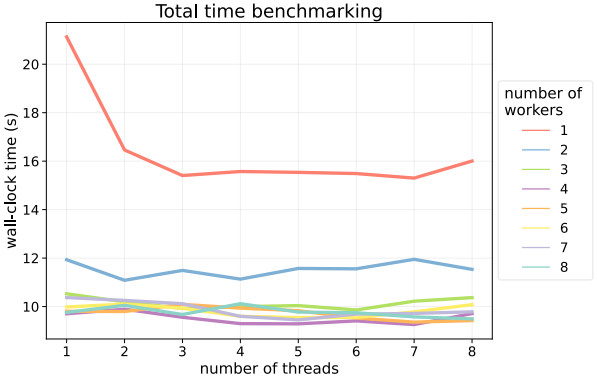
Workers vs threads – partitions = workers



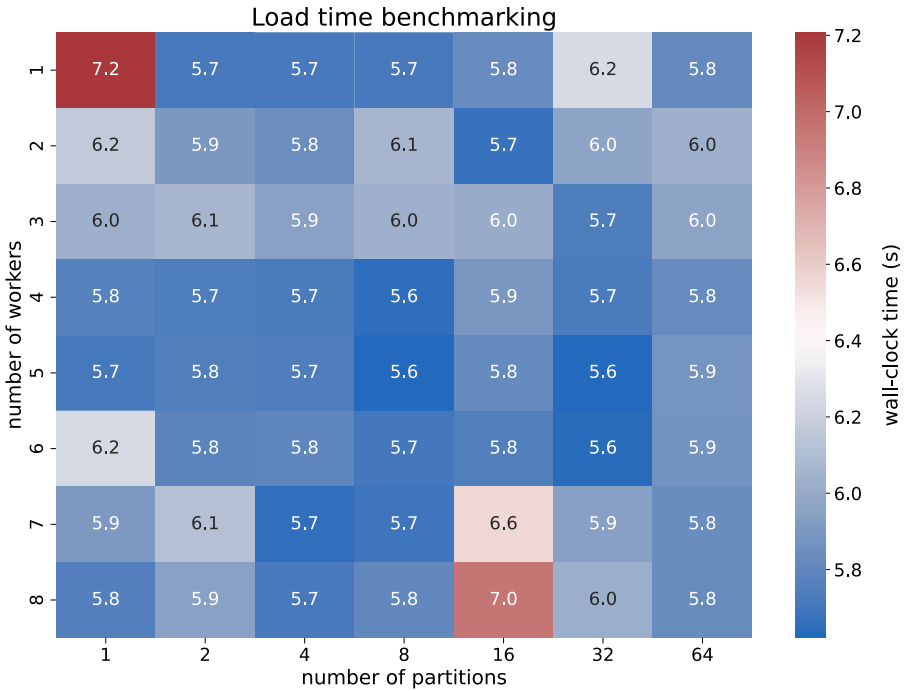


Workers vs threads – partitions = workers

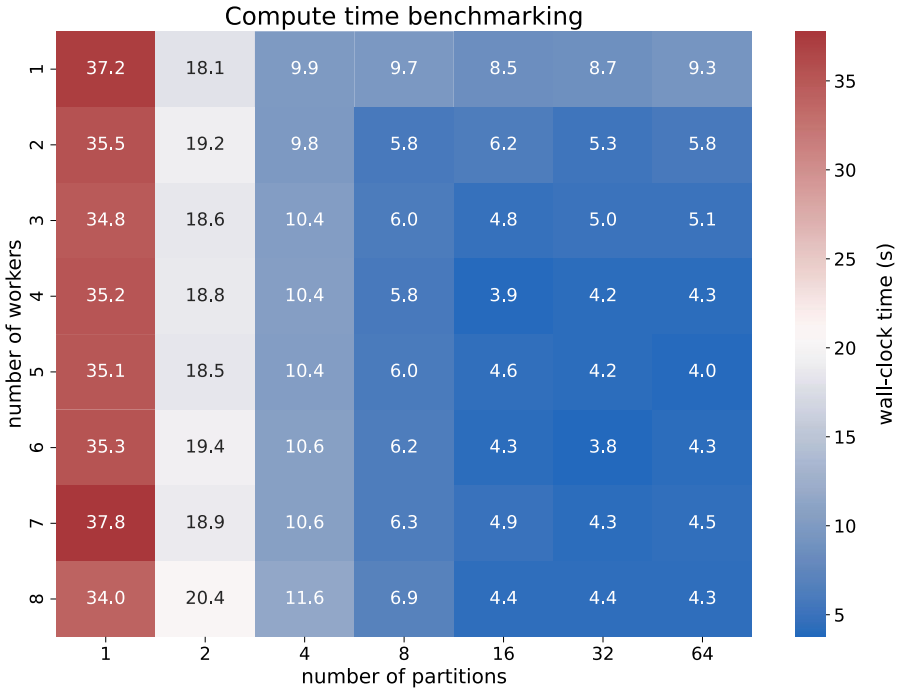




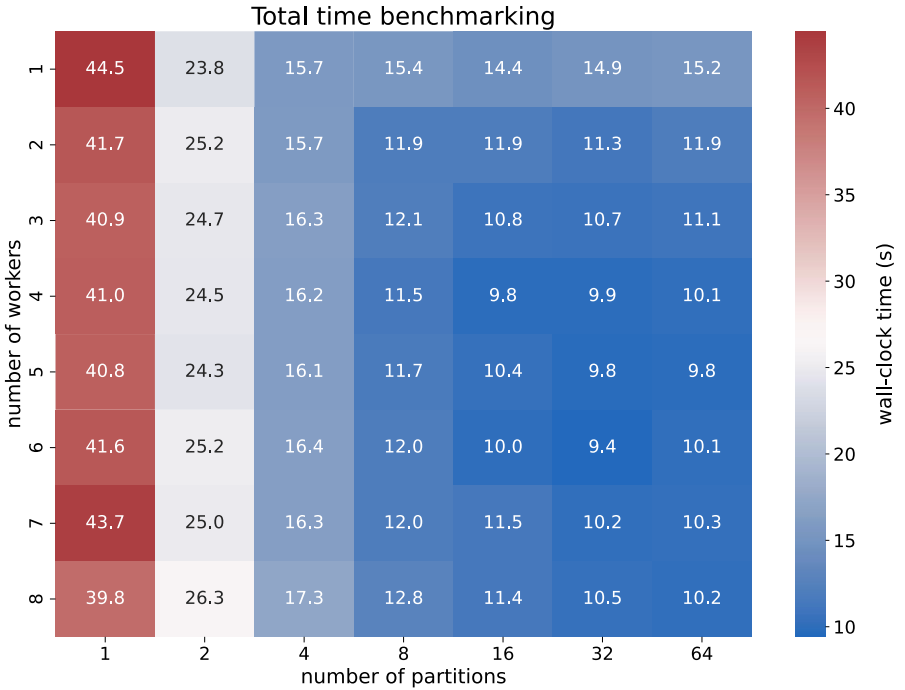
Workers vs partitions – 1 thread



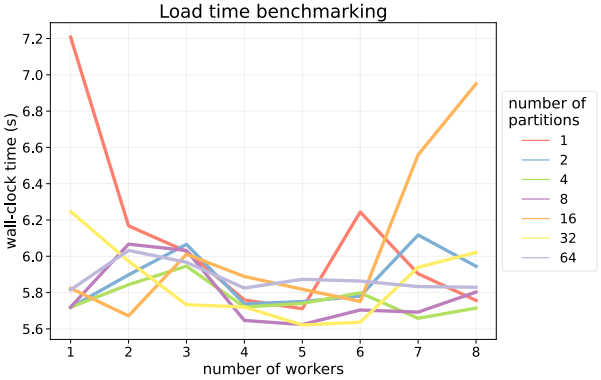
Workers vs partitions – 1 thread

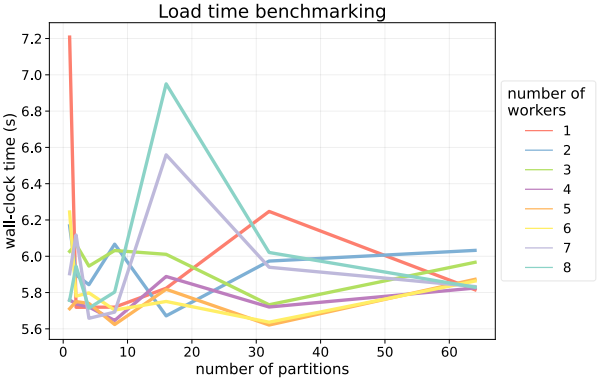


Workers vs partitions – 1 thread

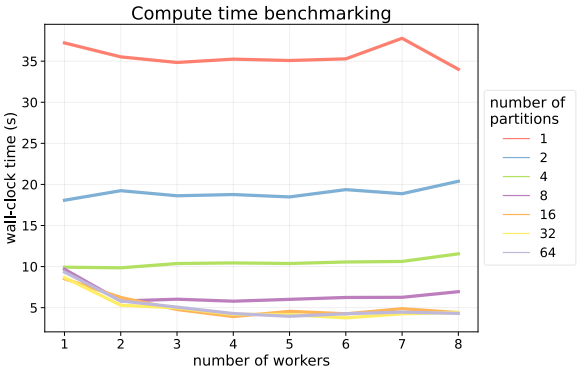


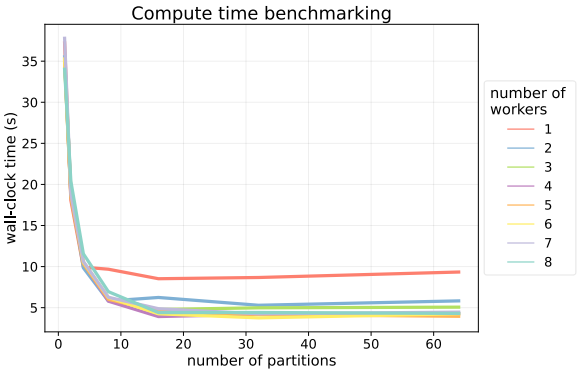
Workers vs partitions – 1 thread





Workers vs partitions – 1 thread





Workers vs partitions – 1 thread