Analysis of Covid-19 papers – Distributed Processing with Dask

Management and Analysis of Physics Dataset mod. B

Francesco Fontana Lorenzo Mancini Giulio Vicentini

Table of contents

1. Introduction

2. Cluster set-up

3. Analysis of Covid-19 papers

4. Conclusions

Introduction

Introduction

Cluster set-up

Analysis of Covid-19 papers

Conclusions

Overview

Exploit distributed processing in order to perform an analysis of several Covid-19 papers:

- Build a cluster using CloudVeneto resources and the Dask library of Python
- Distribute the processing over the nodes and perform some analysis over a dataset made up of Covid-19 papers.

Cluster set-up

Introduction

Cluster set-up

Analysis of Covid-19 papers

Conclusions

Build Cluster with SSH

- We are provided with 3 Virtual Machines on CloudVeneto
- Each VM features:
  - 4 Virtual CPUs
  - 8 GB of memory
  - Same network subnet (nodes must be able to connect and contact each other)
- In order to properly set up a cluster we also must:
  - implement a SSH password-less login
  - set up a unique mounted directory on all machines
  - install the same version of Python and its libraries

Build Cluster with SSH

We exploit *dask.distributed.SSHCluster*

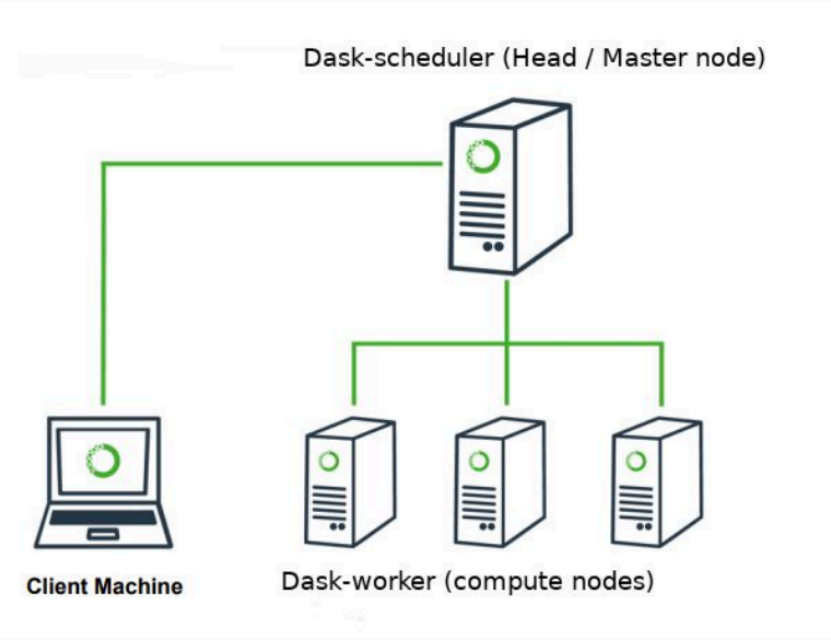
```
scheduler = "10.67.22.84"
nodes = ["10.67.22.84", "10.67.22.254", "10.67.22.111"]
...
c = SSHCluster( workers,
    connect_options={"known_hosts": None},
    worker_options={"nthreads" : nthreads, "memory_limit":limit},
    scheduler_options={"port": 8786, "dashboard_address": 8787 },
  )
...
client = Client(c)
```

in order to create a Cluster between 3 VMs

Parameters of the Cluster

We need to specify some parameters:

- *nprocs*: number of worker that will be used for the cluster
- *nthreads*: number of threads available per worker (deafult: 1)
- *memory*: memory size available for each worker (default: 1 GB)



Analysis of Covid-19 papers

Introduction

Cluster set-up

Analysis of Covid-19 papers

Conclusions

What do we want to study?

1000 papers as data-set (*.json* format) about **Covid-19**:

- **Word counter distributed algorithm**
- **Which are the worst and best represented countries in the research?**
- **Get the embedding for the title of the papers**
- **Cosine similarity**

As we're going to see, (dask bag) it's useful to convert each file into a single line *.json* file.

Word Counter (Map Phase)

Count the number of occurrences of each word, for each document

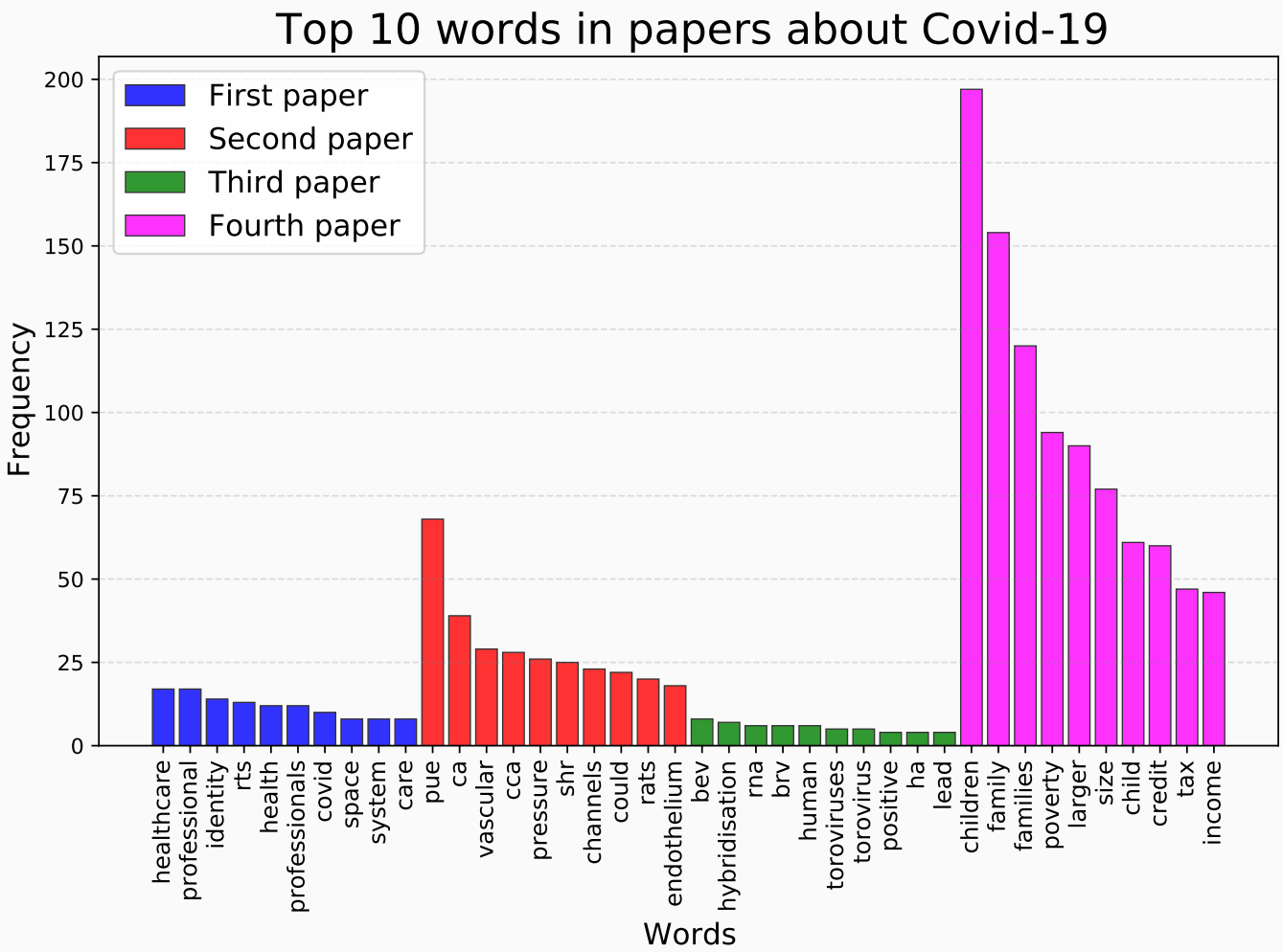
- Create a dask bag in order to read `.json` files (one partition per file by default):

```
b = db.read_text('<files_path>/*.json').map(json.loads)
```

- Concatenate all strings of `text` contained in the `body_text` field of each file and remove some special characters (RegEx)
- Count the words:

```
init_bag = b.pluck("body_text").repartition(50).map(
                                                concatText_list)
map_phase = init_bag.map(Counter)
results = map_phase.compute()
```

Map Phase results

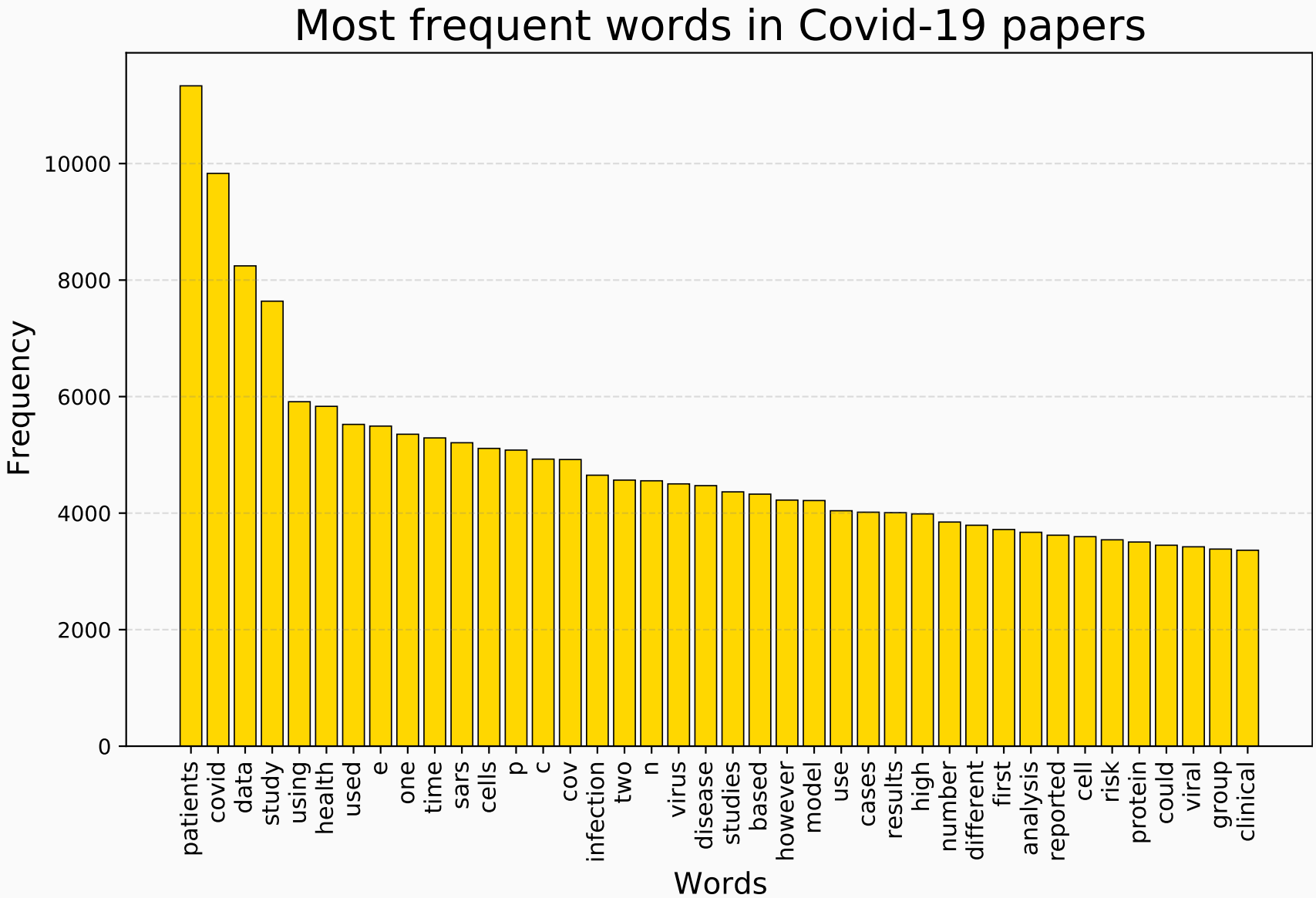


Word Counter (Reduce Phase)

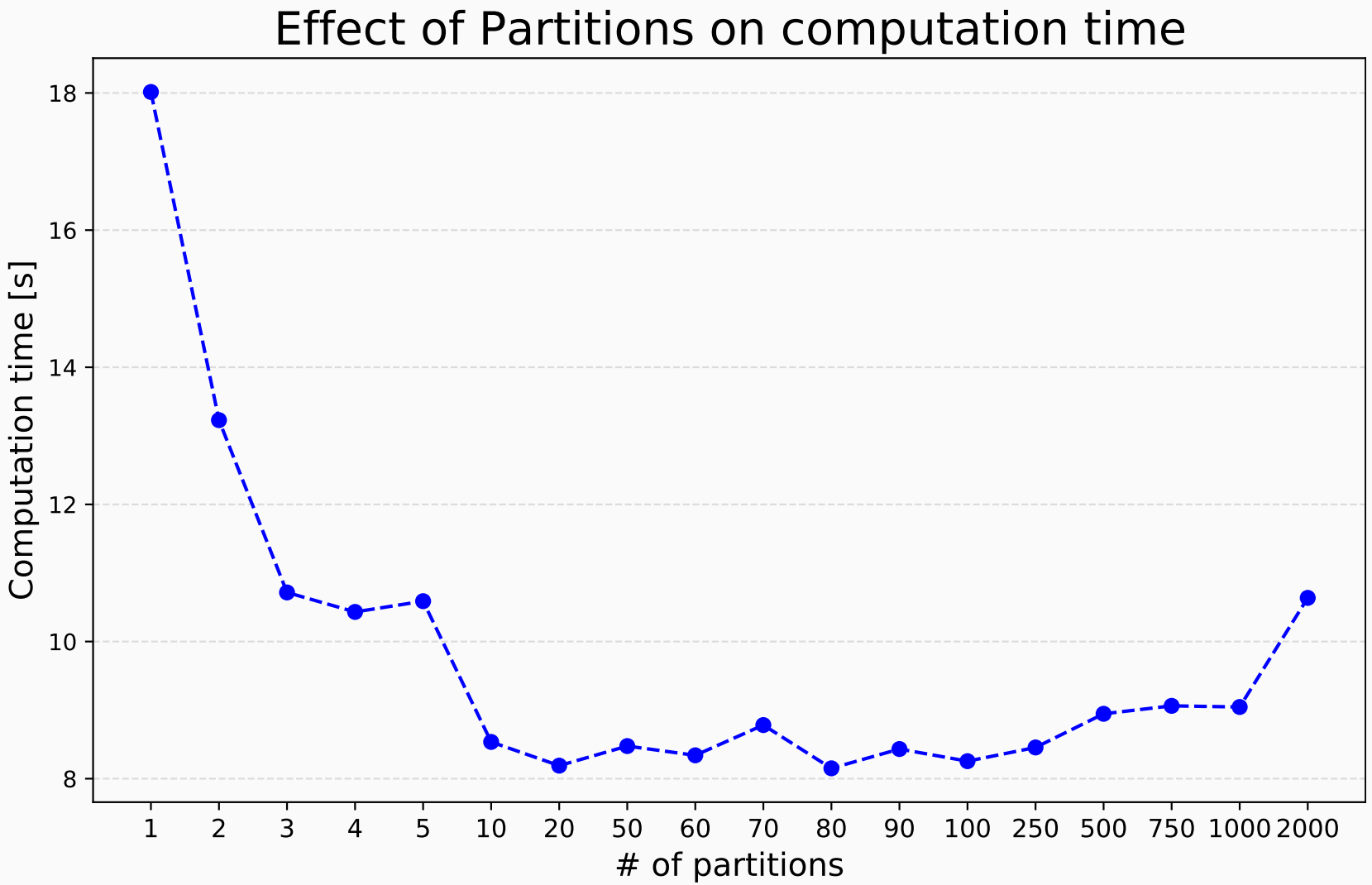
Count the number of occurrences of each word over all the papers.

```
reduce_phase = init_bag.flatten().frequencies(sort=True)
results = reduce_phase.compute()
```

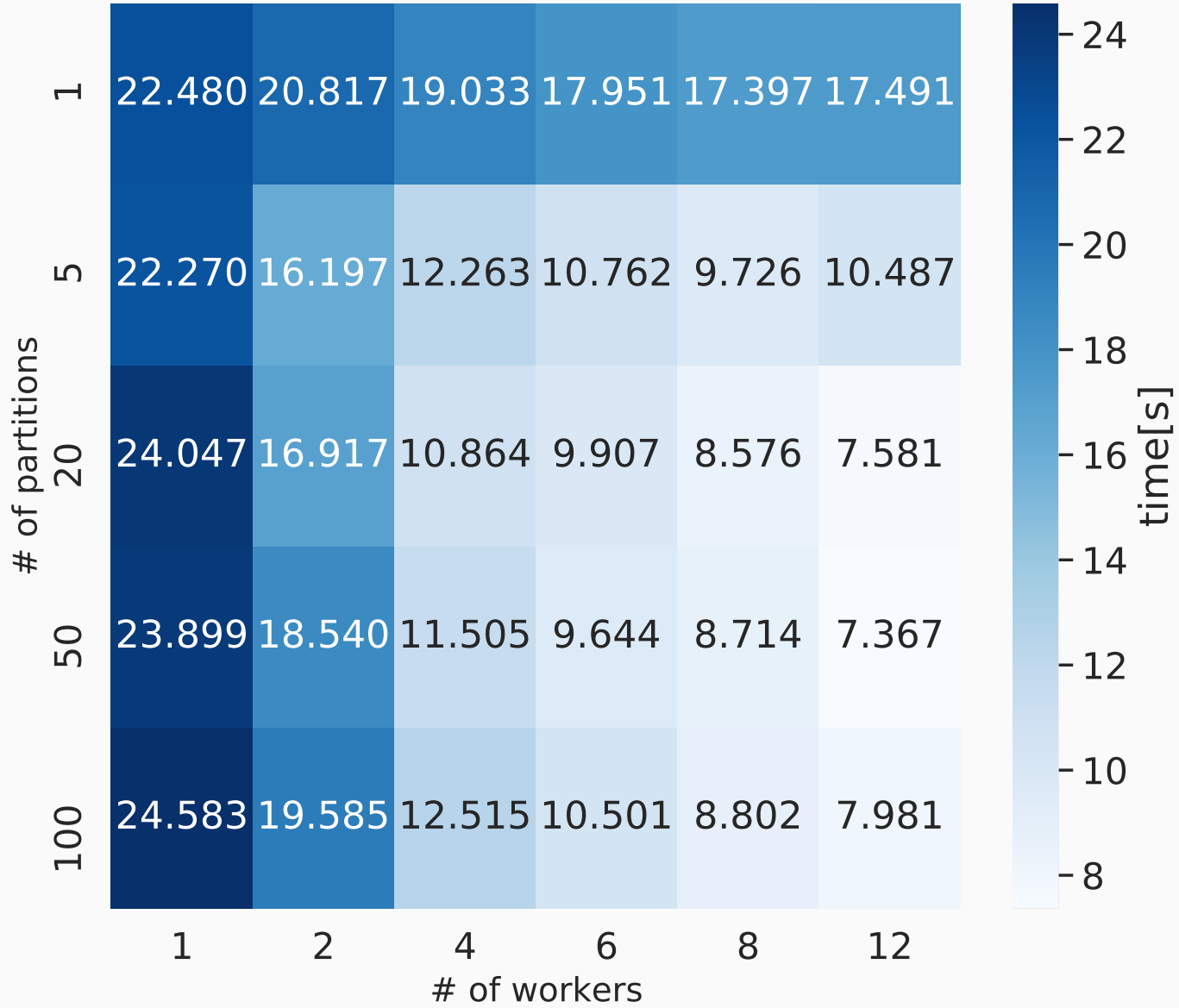
Reduce phase results



Different partitioning



Partitioning vs. workers

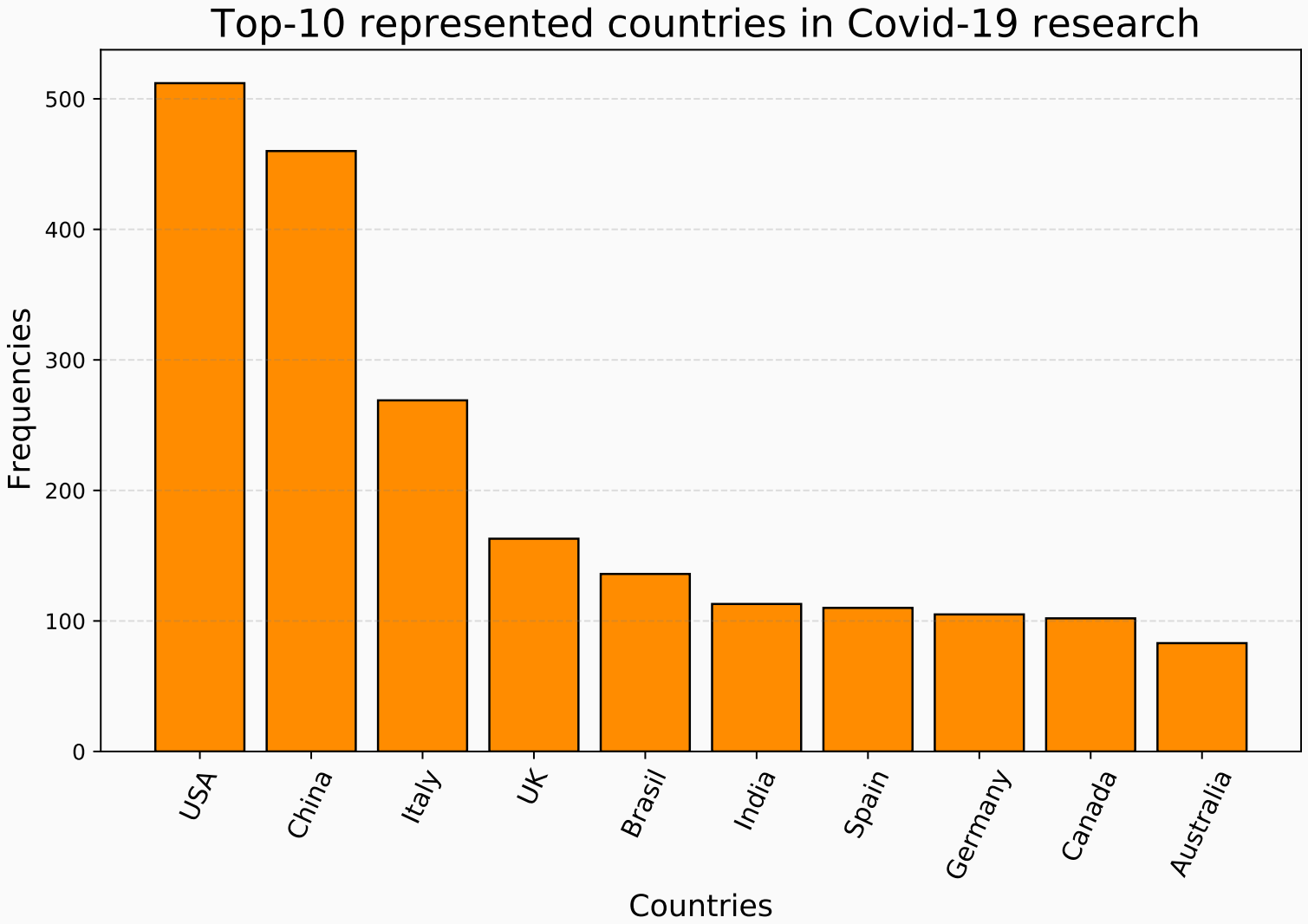


Best represented countries and institutions

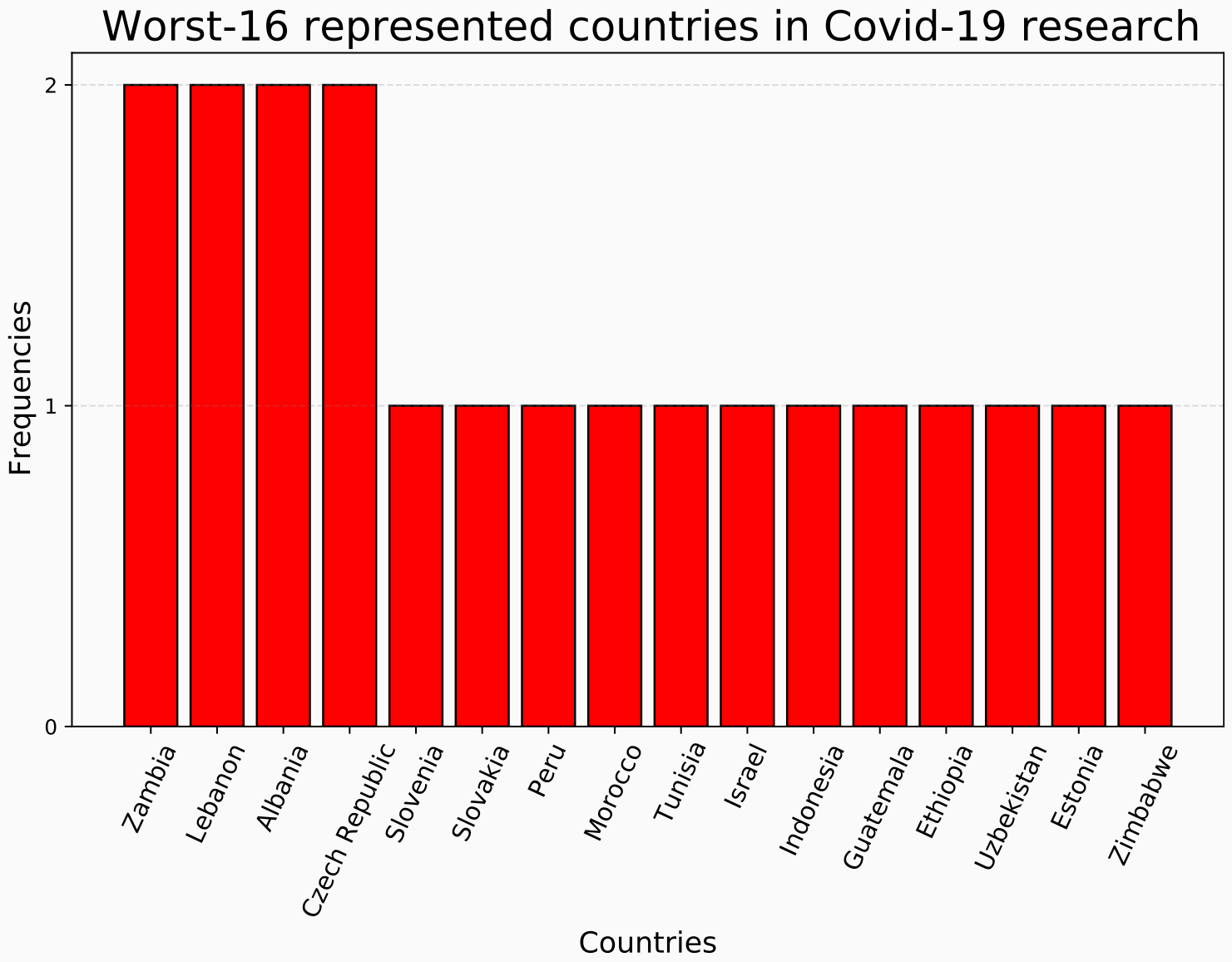
Figure out the countries and institutions that are most and less active in the research about Covid-19

- Convert the document into usable dataframe
- Keep the *metadata* → *authors* → *country* and *institution* fields
- Remove special characters, remove repetitions and check if names are correct

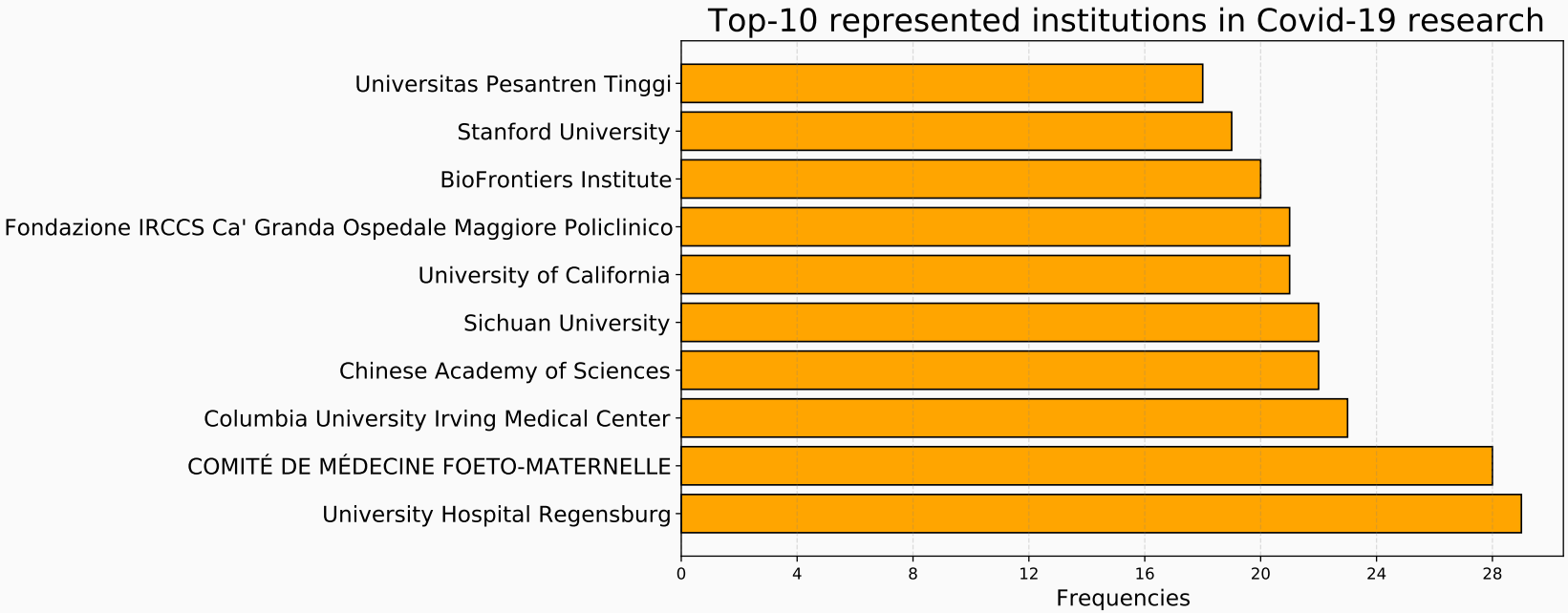
Top represented countries



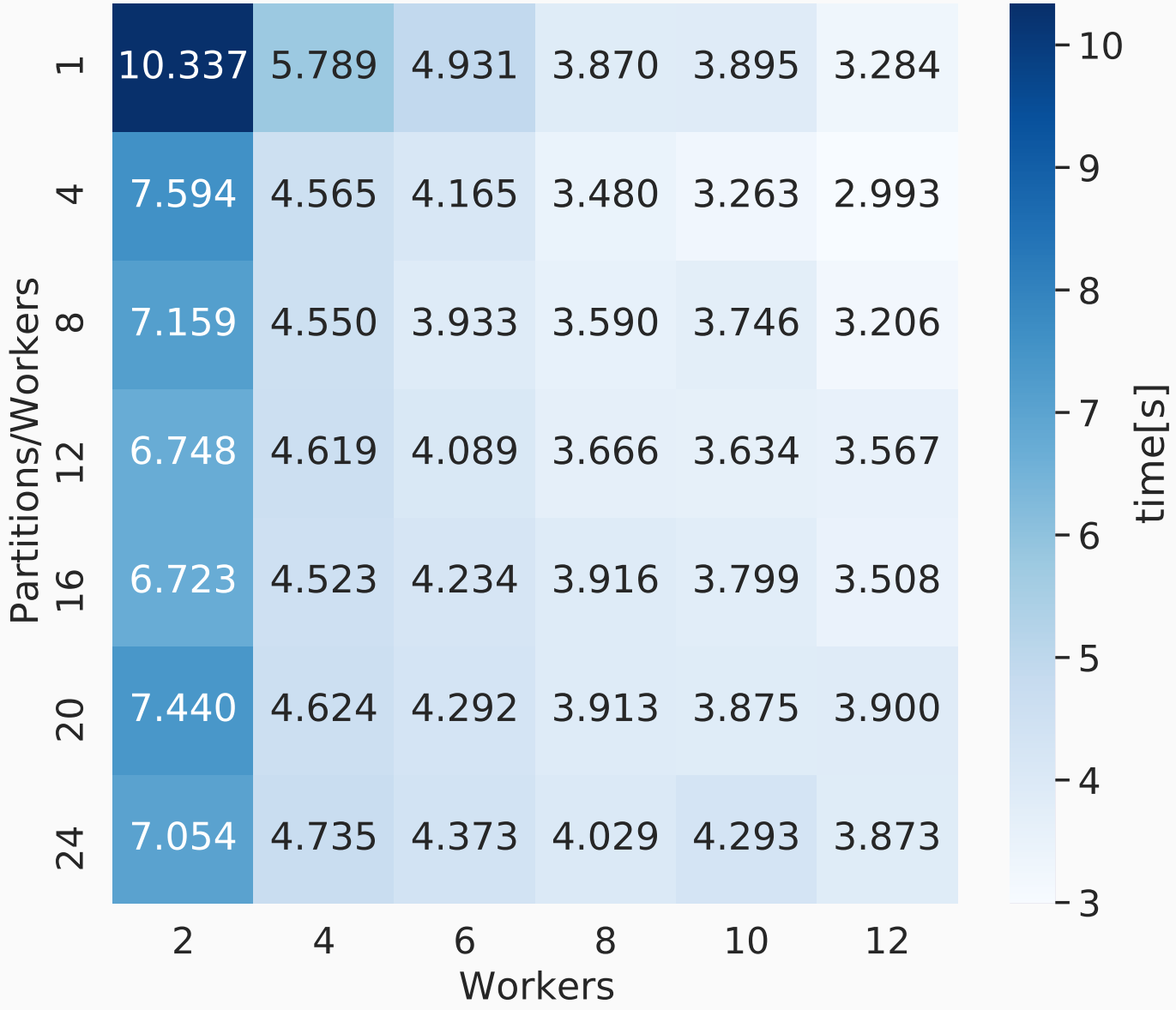
Worst represented countries



Top represented universities



Partitioning vs. workers



Embedding for the title

In this section we will perform a common technique in NLP which consist in transform the text to a set of vectors each one representing a word inside a document. Our goal is to transform the title of the papers into its embedding version.

- At the end of the pre-processing the document will be transformed into a list of vectors or a matrix of $n \times m$ where n is the number of words in the document and m is the size of the vector that represents the word n.
- In order to perform the embedding, a pre-trained model was used that is available at FastText page

Embedding for the title

```
  ({'paper_id': '0003ddc51c4291d742855e9ac56076a3bea33ad7',
'title': 'Journal Pre-proofs The Fire This Time:
'The Stress of Racism, Inflammation and COVID-19',
'title_emb':
[array([-1.484e-01,  7.040e-02,  3.700e-02, -3.310e-02, -1.068e-01,
         9.860e-02,  1.010e-01, -3.980e-02, -1.409e-01,  2.130e-02,
        -1.660e-02,  5.370e-02, -7.670e-02, -4.310e-02, -2.080e-01,
        -2.540e-02,  1.263e-01,  7.910e-02, -9.620e-02, -1.560e-02,
        -1.988e-01, -1.841e-01,  6.800e-03,  2.720e-02,  1.570e-02,
        -7.340e-02,  9.600e-03, -7.230e-02,  9.600e-03,  4.840e-02,
         1.018e-01, -4.440e-02,  8.140e-02, -2.910e-02, -2.881e-01,
         8.310e-02, -5.190e-02, -1.710e-02,  8.660e-02,  1.098e-01,
        -1.841e-01, -1.230e-01, -1.749e-01, -5.290e-02, -2.110e-02,
        -3.710e-02, -1.220e-02,  8.680e-02, -3.940e-02,  2.485e-01,
        -9.490e-02,  1.181e-01, -6.269e-01,  8.800e-03, -1.874e-01,
        -2.215e-01,  2.110e-02,  4.040e-02, -5.200e-03, -5.970e-02,
        -3.150e-02,  2.200e-01,  9.280e-02,  1.094e-01,  2.168e-01,
        -1.520e-02,  1.020e-02,  4.870e-02,  2.320e-02,  8.010e-02,
        -2.787e-01, -5.740e-02,  8.390e-02, -6.900e-02, -1.419e-01,
         1.674e-01, -7.120e-02,  3.110e-02,  6.890e-02, -1.323e-01,
         ...
```

Cosine similarity

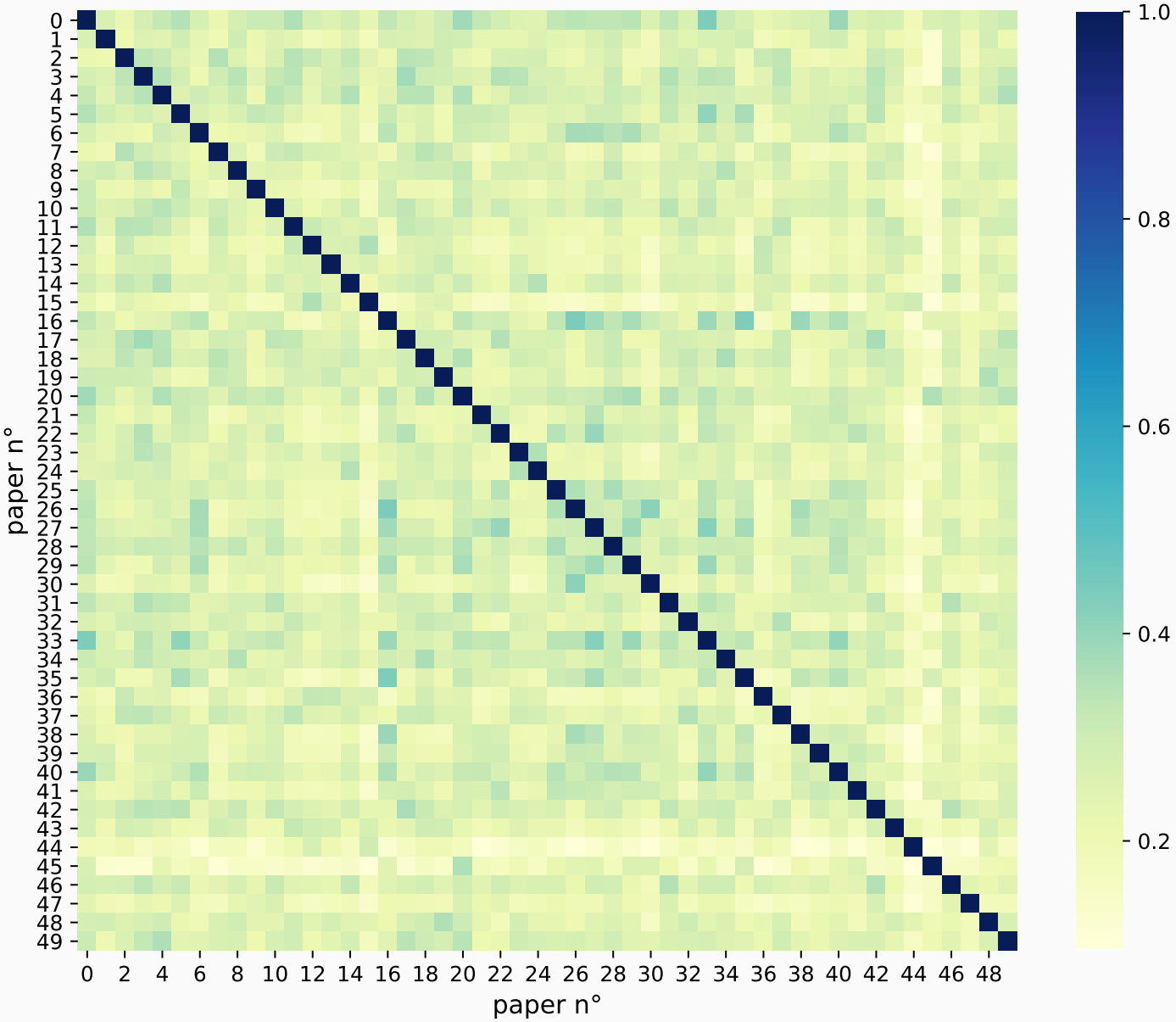
We would like to measure the similarity between the titles $\rightarrow$ **Cosine similarity**:

- Exploit the embedding computed before
- Compute the cosine similarity between embedding of titles[1]

```
def cosine_similarity(vec1, vec2):

    norm_vec1 = np.linalg.norm(vec1)   # norm of 1st embedded title
    norm_vec2 =  np.linalg.norm(vec2)  # norm of 2st embedded title

    m = min(len(vec1), len(vec2)) # min of the length

    result = np.vdot(vec1[:m], vec2[:m])/(norm_vec1*norm_vec2)
    return result
```

[1] Note that in order to do this we must have vectors of same length: thus we keep the minimum length between the titles.

Results of cosine similarity



Conclusions

Introduction

Cluster set-up

Analysis of Covid-19 papers

Conclusions

Conclusions and comments

In our work we exploited distributed processing in order to perform an analysis of some papers:

- We managed to build a distributed processing environment using CloudVeneto resources
- We studied the behaviour of the computation time for different Cluster parameters

In particular we saw that the optimal computation time for most of the tasks, has been reached with an intermediate number of partition and an high number of workers

Thanks for your attention!