# Benchmarking in Spark

Database Administration
Lab Guide 6

2022/2023

Deploy and benchmark Spark jobs in a cloud environment.

**Steps**

1. Deploy a cloud virtual machine by following the steps in the appendix.
2. Using SQL and the `title.basics` dataset, implement a function that counts the number of titles per decade, sorted by most to least.
3. Using SQL and the `title.basics` dataset, implement a function that computes the average title *runtime*, given a title type and the lower and upper limit years.
4. Measure the time of executing the `run` function in the provided script.[1] Experiment providing a DataFrame read from Parquet and another read from TSV.
5. Measure the `run` function time with the Parquet DataFrame using a single partition, by executing the function `coalesce(1)`.[2]
6. Measure the `run` function time with the Parquet DataFrame using a single worker (without `coalesce`), by scaling down the cluster.[3]
7. Export and read the Parquet DataFrame with different column partitions to improve the `averageRuntime` query (3 workers, without `coalesce`). Test the optimal partitioning found with the `titlesByDecade` query.

**Questions**

1. What is the performance difference between Parquet and TSV? How does their internal storage organization impact data locality in the executed queries?
2. What is the performance impact of executing with just one partition? Explain.
3. What is the optimal column partitioning for the `averageRuntime` query? Explain the performance impact of this partitioning on the `titlesByDecade` function. Is any partitioning worth it in the context of the `run` workload?

**Learning Outcomes** Benchmark different options in Spark and choose the optimal ones. Get familiarized with cloud environments.

[1] Suggestion: use Python's `time.time()` function.
[2] E.g., `run(titlesP.coalesce(1))`.
[3] To scale-down/scale-up a cluster, simply re-execute the `docker-compose up` command with the new number of workers desired: `docker-compose -p spark up -d --scale spark-worker=`$N$.

# Google Cloud virtual machine instructions

**IMPORTANT: provisioning a virtual machine in Google Cloud has an associated hourly cost. Be sure to delete the instance at the end as each student has a limited amount of credits.**

1. Download the new supplementary files;
2. Prepare a zip file with the needed material (docker-compose.yml, run.sh, install.sh, app/main.py);
3. Go to `https://console.cloud.google.com`, open the navigation menu on the top-left side, and access *Compute Engine*;
4. On the left menu, open the *Metadata* setting; Click *Edit* and add your public SSH key;
5. Back in *Compute Engine*, select *Create Instance* and create the virtual machine as follows:
   (a) *Machine Type* – Custom, 4 vCPU, 8 GB RAM;
   (b) *Boot disk* – Ubuntu, Ubuntu 22.04 LTS x86/64, SSD persistent disk, 100 GB;
   (c) *Advanced Options - Management - Availability policies - VM provisioning model* – Spot (machines created with this option can be randomly turn off, but are available at a lower price: `https://cloud.google.com/compute/docs/instances/spot`);
6. Move the auxiliary zip file to the cloud instance. E.g.: `scp` $files.zip$ $user@ip$`:` (the public IP can be found in the *Compute Engine* page);
7. Access the virtual machine through SSH, prepare the Spark cluster, and execute the job:

```
# access the virtual machine and extract the files
ssh user@ip
sudo apt install unzip -y
unzip files.zip
cd files

# installs docker and downloads the title.basics dataset (TSV and parquet)
sudo chmod +x install.sh
./install.sh

# spark cluster with 3 workers
docker-compose -p spark up -d --scale spark-worker=3

# submit the main.py job
docker exec spark_spark_1 python3 main.py
```

8. After completing the guide, delete the virtual machine by accessing the *Compute Engine* page, clicking the three dots on the respective instance, and selecting *Delete*.