General Information

1. Installation

Both Dask and Spark can be installed with conda

```
conda install dask
conda install pyspark
```

For Spark, installing Java and setting JAVA_HOME are required

```
sudo apt install openjdk-8-jdk
sudo nano /etc/environment
JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
```

2. Characters

Spark: written in Scala, compatible with Java, Python and R. In general Spark has more ready-to-use tools, e.g. MLlib for machine learing and GraphX for graph processing. It integrates well with many other Apache projects. It is fundamentally an extension of the Map-Shuffle-Reduce paradigm.

Dask: written in Python. It works well with common python libraries like NumPy, Pandas. It is lack of high-level optimization (comparing to Spark), but it is more fexible. Therefore it can build more complex bespoke systems. It is fundamentally based on generic task scheduling.

3. The way of working

- Spark: Spark separate the input parameters into several blocks, and apply parallel computation of all blocks. Within each block the computaions are performed sequentially.
    1. Set up SparkContext
       ```
       sc = SparkContext(master="local[{}]".format(nr_worker))
       ```
    2. Parallelize input parameters in to blocks
       ```
       task = sc.parallelize(in_out_file_pairs_spark)
       ```
    3. Map the function to each block and collect results
       ```
       task.map(export_ndvi).collect()
       ```
    4. Stop SparkContext
       ```
       SparkContext.stop(sc)
       ```
- Daks: Dask registers the operations in a list (“futures” in the script), and distribute these tasks into the available resources defined in the Client.
    1. Set up the client:
       ```
       cluster = LocalCluster(processes=True, n_workers=nr_worker,
       threads_per_worker=1, local_directory='./dask-worker-space')
       ```
    2. submit the function to a list of futures
       ```
       future = [client.submit(export_ndvi, f) for f in in_out_file_pairs_dask]
       ```
    3. gather futures (computaion)
       ```
       results = client.gather(futures)
       ```
    4. shutdown client
       ```
       address = client.scheduler.address  # get adress
       client.close()
       Client(address).shutdown()
       ```

Test case: NDVI computation with Sentinel-2 data

1. Description of the test case

In this test case we attempt to compute the Normalized Difference Vegetation Index (NDVI) from Sntinel-2 images. We run the same NDVI computaion through both Spark and Dask, with the same input images, and assess the wall time of computaion.

The purpose is to 1) have a local test run and get farmiliar with both frameworks. 2) investigate if Spark and Dask have major performance difference.

The code of this test can be found in: https://github.com/phenology/dask_spark_comparision_s2

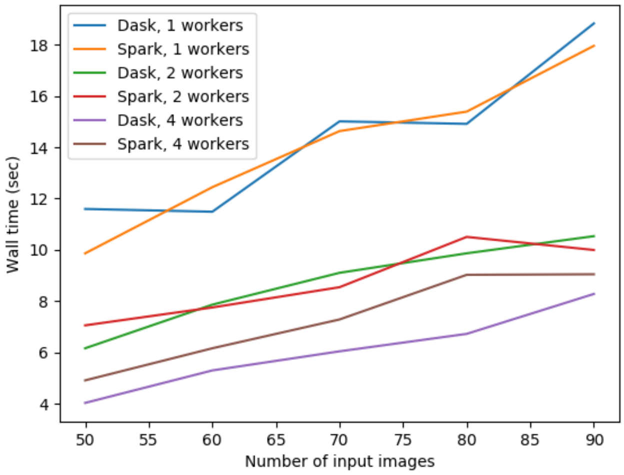
2. Performance comparision

Below are the three comparision runs between Dask and Spark:

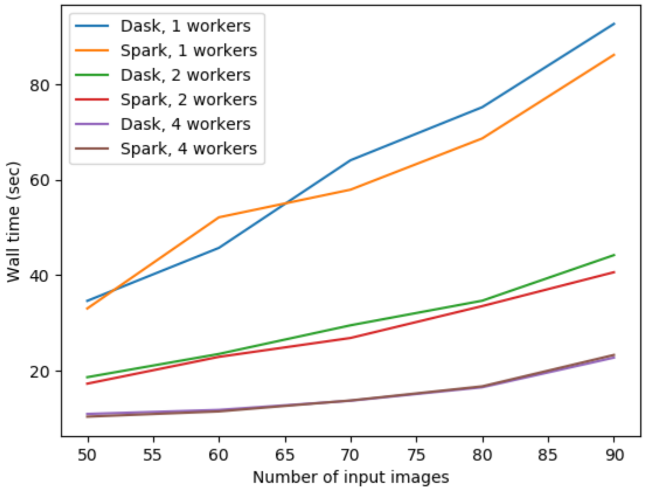
- Run 1: processing 50-100 images, no image exporting
- Run 2: processing 50-100 images, with image exporting
- Run 3: processing 50-500 images, no image exporting

For all three wrong we excute with 1, 2 and 4 workers in both system.

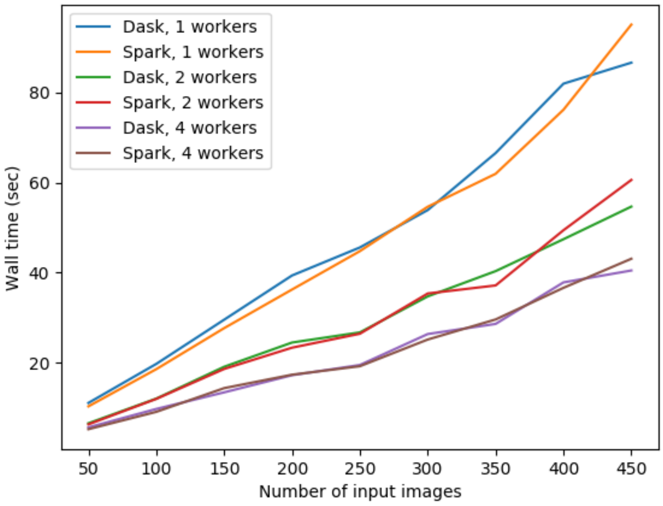
In the results we only track the wall time of actual processing (Step 3 as mentioned in "General information, step 3"). Therefore we ignore the time of setting up and closing the scheduler.



Wall time 50-100 images, no exporting



Wall time 50-100 images, with exporting



Wall time 50-500 images, no exporting

3. Taking out from the test case

- The performance of Spark and Dask do not show major difference in this specific case.
- The speed of processing are roughly linear related with the number of images and the number of workers, which match the expectation.

Recommendations

Below we assume a Python development case, so we do not consider Spark's advantage in Scala or SQL language. These conclusions are drew mainly from the documentaion of the two frameworks.

1. When to Spark

- When the use case is relatively simple, i.e. cleanly fits the Map-Shuffle-Reduce paradigm.

- When one can find a quick solution with the high-level tools provided by Spark.

2. **When to use Dask**

   - When the case is complex and more customization is required.
   - When one wants to perform parallization to the existing legacy code

Existing documentaions on comparision

- From Dask website: https://docs.dask.org/en/latest/spark.html
- Dask vs Spark comparision case on neuroimaging pipelines (concluding the performance is similar): https://arxiv.org/pdf/1907.13030.pdf