Multi-GPU K-Means with Dask

In this notebook you will use GPU-accelerated K-means to identify population clusters in a multi-node, multi-GPU scalable way with Dask.

Objectives

By the time you complete this notebook you will be able to:

- Use distributed, GPU-accelerated K-means with Dask

Imports

First we import the needed modules to create a Dask cuDF cluster.

In [1]:
```
import subprocess
import logging

from dask.distributed import Client, wait, progress
from dask_cuda import LocalCUDACluster
```

After that, we create the cluster.

In [2]:
```
cmd = "hostname --all-ip-addresses"
process = subprocess.Popen(cmd.split(), stdout=subprocess.PIPE)
output, error = process.communicate()
IPADDR = str(output.decode()).split()[0]

cluster = LocalCUDACluster(ip=IPADDR, silence_logs=logging.ERROR)
client = Client(cluster)
```

```
distributed.preloading - INFO - Import preload module: dask_cuda.initialize
distributed.preloading - INFO - Import preload module: dask_cuda.initialize
distributed.preloading - INFO - Import preload module: dask_cuda.initialize
distributed.preloading - INFO - Import preload module: dask_cuda.initialize
```

Finally, as we did before, we import CUDA context creators after setting up the cluster so they don't lock to a single device.

In [3]:
```
import cudf
import dask_cudf

import cuml
from cuml.dask.cluster import KMeans
```

Load and Persist Data

We will begin by loading the data, The data set has the two grid coordinate columns, `easting` and `northing`, derived from the main population data set we have prepared.

In [4]:
```
ddf = dask_cudf.read_csv('./data/pop5x_2-07.csv', dtype=['float32', 'float32'])
```

Training the K-means model is very similar to both the scikit-learn version and the cuML single-GPU version--by setting up the client and importing from the `cuml.dask.cluster` module, the algorithm will automatically use the local Dask cluster we have set up.

Note that calling `.fit` triggers Dask computation.

In [5]:
```
%%time
dkm = KMeans(n_clusters=20)
dkm.fit(ddf)
```

```
CPU times: user 5.71 s, sys: 4.03 s, total: 9.73 s
Wall time: 3min 42s
```
Out[5]:
```
<cuml.dask.cluster.kmeans.KMeans at 0x7ff653279250>
```

Once we have the fit model, we extract the cluster centers and rename the columns from their generic '0' and '1' to reflect the data on which they were trained.

In [6]:
```
cluster_centers = dkm.cluster_centers_
cluster_centers.columns = ddf.columns
cluster_centers.dtypes
```

Out[6]:
```
northing    float32
easting     float32
dtype: object
```

Exercise: Count Members of the Southernmost Cluster

Using the `cluster_centers`, identify which cluster is the southernmost (has the lowest `northing` value) with the `nsmallest` method, then use `dkm.predict` to get labels for the data, and finally filter the labels to determine how many individuals the model estimated were in that cluster.

In [7]:
```
# %load solutions/southernmost_cluster
south_idx = cluster_centers.nsmallest(1, 'northing').index[0]
labels_predicted = dkm.predict(ddf)
labels_predicted[labels_predicted==south_idx].compute().shape[0]
```

Please Restart the Kernel

In [8]:
```
import IPython
app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

Out[8]:
```
{'status': 'ok', 'restart': True}
```

Next

In the next notebook, you will calculate infection risk again, this time using the powerful XGBoost algorithm.