Running dask on a cluster

Dask greatly simplifies the work on a HPC cluster where different CPUs do not belong to the *same* machine like on a large station or a Google Cloud/AWS/Azure/SWITCHengine cloud computer.

In particular the dask-jobqueue module helps dealing with various queing systems typically used on such systems. For example at Unibe, the Ubelix cluster uses the SLURM system. In normal usage, one has to write submission requests to execute jobs, make sure they properly exploit resources if meant to work in parallel etc. As we'll show here Dask massively simplifies the procedure.

In [1]:
```
from dask_jobqueue import SLURMCluster
```

First we create a "cluster on the cluster" and use the `SLURMCluster` in this particualar case. We can specify here all parameteres that one can commonly specify on SLURM. Here we only say how many CPUs and hown much RAM per CPU we need:

In [2]:
```
cluster = SLURMCluster(
    cores=1,
    memory="5 GB"
)
```

With this command, Dask has created (but not submitted) the request to slurm. We can use the `jon_script()` method to see how that request looks. It's a standard SBATCH script:

In [ ]:
```
print(cluster.job_script())
```

At the top we see specifications for the cluster (including e.g. our RAM request) and on the bottom we se the command executed on the cores so that we can use them with Dask. Note that **this is all done automatically for you**.

Then we proceed as usual and create a client that we connect to the cluster. Unfortunately, it's not yet possible to use the dask dashboard on the cluster.

In [4]:
```
from dask.distributed import Client
client = Client(cluster)
```

However we can adjust the size of our cluster which for the moment has 0 workers and thus 0 CPUs. Any time we scale up, new jobs are sent to the cluster. If we scale down, the jobs are stopped. If we monitor our resource usaage on the cluster, we we'll see jobs appearing and disappearing.

In [16]:
```
client.cluster
```

We can also use the simple `scale()` command:

In [14]:
```
cluster.scale(jobs=10)
```

Finally we can do what we came to do: calculations !

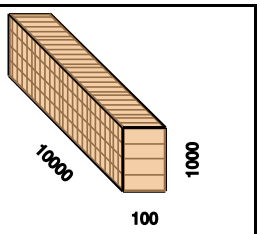
In [8]:
```
import dask.array as da
```

In [9]:
```
myarray = da.random.randint(0,100,(10000,1000,100))
```

In [10]:
```
myarray
```

Out[10]:

	Array	Chunk
Bytes	8.00 GB	80.00 MB
Shape	(10000, 1000, 100)	(400, 250, 100)
Count	100 Tasks	100 Chunks
Type	int64	numpy.ndarray



In [13]:
```
%%time
myarray.mean().compute()
```

```
CPU times: user 302 ms, sys: 32.4 ms, total: 335 ms
Wall time: 7.14 s
```

Out[13]:
```
49.498726549
```

In [15]:
```
%%time
myarray.mean().compute()
```

```
CPU times: user 462 ms, sys: 52.9 ms, total: 515 ms
Wall time: 2.83 s
```

Out[15]:
```
49.498726549
```