

Building an Infrastructure as a Service on the Amazon Web Service Cloud

ilante

June 25, 2021

1 Creating an HTCondor Cluster for Training Support Vector Machines for Secondary Structure Prediction

The aim of this project is to demonstrate how to build an Infrastructure as a Service (IaaS) on the [Amazon Web Services](#) Platform. We chose [libsvm](#) [1], a library for Support Vector Machines as application to run the test jobs, because training a model for classification is a computationally intensive task. In a real scenario it is necessary to perform a grid search, for finding the best hyperparameters for training the model [2]. The IaaS we introduce in this project is an HTCondor cluster consisting of only three nodes; one [main](#) node and two worker nodes. The infrastructure can be expanded by simply replicating the worker node instance according to the needs of the application. The main node was not set up as worker node to avoid the risk of overloading. A shared storage that is directly attached to the main node is available to all worker nodes. This shared storage is implemented by the Network File System (NFS) protocol. All scripts can be found in the projects [GitHub repository](#).

1.1 Initiating the Instances on the Amazon Web Services Cloud

For this project we chose to use the cloud service provider Amazon Web Services ([AWS](#)) as free credit was provided by the course, where we instantiated three Nodes: worker nodes and the main node were all run on CentOS Linux 8. For the main node, we chose the `t2.micro` instance type with a 10 Gb SSD as root storage. For the worker nodes, we chose the `t2.small` instance type with a 16 Gb SSD as root storage.

| <input type="checkbox"/> | Name | Size | Volume Type | Availability Zone | State | Attachment Information | Volume Status |
|--------------------------|------------|--------|-------------|-------------------|---|------------------------|---|
| <input type="checkbox"/> | shared_vol | 30 GiB | gp2 | us-east-1b | ● in-use | i-00a5ead5840cdb01... | ✔ Okay |
| <input type="checkbox"/> | WN_1 | 16 GiB | gp2 | us-east-1b | ● in-use | i-0863cc33834b6c3e... | ✔ Okay |
| <input type="checkbox"/> | WN_2 | 16 GiB | gp2 | us-east-1b | ● in-use | i-0b7176125f178249... | ✔ Okay |
| <input type="checkbox"/> | main | 10 GiB | gp2 | us-east-1b | ● in-use | i-00a5ead5840cdb01... | ✔ Okay |

To ensure that the machines are able to communicate through private IPv4 addresses, all nodes were instantiated in the same availability zone (`us-east-1b`) and the same security group. The security group is set to allow Secure Shell Protocol (*SSH*) connection, port 22, from everywhere, provided the user has the matching key. We did not open it exclusively to our local public IP, because throughout the project we are accessing the internet from different geographic locations. The IP address of my private laptop will change whenever I'm accessing from a different WiFi network, but also when the router is restarted. All Transmission Control Protocol (*TCP*), was allowed for machines of the *same* security group, thus allowing for file transfer [3]. Further, HTCondor daemons use a dynamically assigned port. All ICMP was allowed between machines sharing the same security group, to enable *ping* e.g. for checking if a certain host is online and other testing purposes.

EC2 > Security Groups > sg-003a78e68d04f193a - BDP1_project_Immanuela_Englander

sg-003a78e68d04f193a - BDP1_project_Immanuela_Englander

Actions

Details

Security group name

BDP1_project_Immanuela_Englander

Security group ID

sg-003a78e68d04f193a

Description

Allows SSH, TCP, UDP and ICMP (for ping)

VPC ID

vpc-1284076f

Owner

295029965479

Inbound rules count

3 Permission entries

Outbound rules count

1 Permission entry

Inbound rules

Outbound rules

Tags

Inbound rules (3)

Edit inbound rules

| Type | Protocol | Port range | Source | Description - optional |
|-----------------|----------|------------|---|---|
| All TCP | TCP | 0 - 65535 | sg-003a78e68d04f193a / BDP1_project_Immanuela_Englander | - |
| SSH | TCP | 22 | 0.0.0.0/0 | - |
| All ICMP - IPv4 | ICMP | All | sg-003a78e68d04f193a / BDP1_project_Immanuela_Englander | Allow ping inside the Virtual Private Network |

1.2 Configuring the Main Node

We changed the PS1 prompt of the main node ensuring any user logging in, can recognize it right away. This was done for reducing the risk of modifying any configuration on the wrong machine.

```
[centos@main:~$ echo $PS1
\[\033[01;32m\]\u@main\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]$
```

First, we installed dependencies for HTCondor [4] on both main and worker nodes with the commands shown below:

```
1 sudo su -
2 # SELinux policies for container runtimes, dependency for HTCondor
3 wget mirror.centos.org/centos/7/extras/x86_64/Packages/container-selinux-2.107-3.el7.noarch.
  rpm
4
5 yum localinstall container-selinux-2.107-3.el7.noarch.rpm
6
7 yum clean all
```

Then we created a new yum repo for HTCondor and copied the stable HTCondor into the new repo:

```
1 # creating new yum HTCondor repo
2 wget http://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo
3 #copy to my new yum repo
4 cp htcondor-stable-rhel7.repo /etc/yum.repos.d/
```

First we obtained the key used for authentication of the repo, and told the system to trust the key. Thereafter we installed all packages of HTCondor as follows:

```
1 # obtain key
2 wget http://htcondor.org/yum/RPM-GPG-KEY-HTCondor
3 # import and trust
4 rpm --import RPM-GPG-KEY-HTCondor
5 # install HTCondor
6 yum install condor-all
```

Then we enabled and started HTCondor with the following commands:

```
1 systemctl enable condor
2 systemctl start condor
```

To ensure that the installation was successful, we checked the status of the software we used the command showing in the screenshot on the next page:

```
[centos@main:~$ systemctl status condor
• condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/usr/lib/systemd/system/condor.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2021-05-31 14:47:44 UTC; 9min ago
     Main PID: 960 (condor_master)
        Status: "All daemons are responding"
          Tasks: 3 (limit: 4194303)
         Memory: 10.4M
        CGroup: /system.slice/condor.service
                └─ 960 /usr/sbin/condor_master -f
                  1195 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 990
                  1205 condor_shared_port -f

May 31 14:47:44 ip-172-31-1-180.ec2.internal systemd[1]: Started Condor Distributed High-Throughput-Computing.
May 31 14:47:44 ip-172-31-1-180.ec2.internal htcondor[1000]: Changing TCP_LISTEN_QUEUE (/proc/sys/net/core/somaxconn) from 128 to 1024
May 31 14:47:44 ip-172-31-1-180.ec2.internal htcondor[1013]: Not changing ROOT_MAXKEYS (/proc/sys/kernel/keys/root_maxkeys): new value (1000000) <= old value (15
May 31 14:47:44 ip-172-31-1-180.ec2.internal htcondor[1022]: Not changing ROOT_MAXKEYS_BYTES (/proc/sys/kernel/keys/root_maxbytes): new value (25000000) <= old 2
May 31 14:47:44 ip-172-31-1-180.ec2.internal htcondor[1037]: Changing FS_CACHE_DIRTY_BYTES (/proc/sys/vm/dirty_bytes) from 0 to 100000000
May 31 14:47:44 ip-172-31-1-180.ec2.internal htcondor[1044]: Changing MAX_RECEIVE_BUFFER (/proc/sys/net/core/rmem_max) from 212992 to 10485760
```

For basic configuration, we appended the following lines to the main nodes HTCondor configuration file, found in the path: /etc/condor/condor_config:

```
1 # Main Node IP
2 CONDOR_HOST = <Main_Node_private_IP>
3
4 # Main Node config
5 DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD
6
7 HOSTALLOW_READ = *
8 HOSTALLOW_WRITE = *
9 HOSTALLOW_ADMINISTRATOR = *
```

To make sure that the changed settings take effect, we restarted HTCondor and checked the status again with the following command:

```
1 sudo systemctl restart condor
2 condor_status
```

1.3 Setting up Network File System on the MASTER Node

To set up a Network File System (NFS) we first had to create a new disk ("Volume" in AWS lingo). From the Amazon Elastic Compute Cloud (EC2) side bar menu under 'Elastic Block Storage' we selected 'Volumes' and picked the default EBS General Purpose SSD (gp2) with 30 GiB capacity and attached it to the main node. This was done by checking the volume and subsequently opening the 'Actions' dropdown and selecting 'Attach Volume'.

Then via the console we added the new volume to the partition list via the `fdisk` utility:

`fdisk /dev/<my_new_volume>` was added to the partition table using the default parameters. The changes were saved using the `w` (write) command. Then we created the file system on the partition we just created by issuing the following command:

```
1 # Formatting my new partition
2 mkfs.ext4 /dev/<my_partition>
```

By modifying the `/etc/fstab` file (see snippet below) we ensured the main node will mount the volume automatically upon boot onto the directory `/data`

```
1 /dev/<my_newly_formatted_partition>          /data    ext4    defaults    0 0
```

I installed the NFS by issuing the commands below:

```
1 yum install nfs-utils rpcbind
2 systemctl enable nfs-server
3 systemctl enable rpcbind      # starts the service upon boot
4 systemctl start rpcbind
5 systemctl start nfs-server
6 # checking if install, enable start worked fine
7 systemctl status nfs-server
```

We created a new directory in `/data` which was then used as mount point for the shared FS. We appended the following line to the NFS configuration file in `/etc/exports` to expose `/data` to all the machines in my virtual private network, or virtual private cloud (VPC) in AWS lingo. This allows any computer in the VPC to connect to the NFS server and access the shared FS. This poses little security risk as only we can create new machines belonging to the same VPC, and it is a secure solution as long as only HTCondor Main and worker nodes are hosted in this VPC. If I, as admin wanted to launch any other application we would have to host it on a different VPC.

```
1 /data 172.31.0.0/16(rw,sync,no_wdelay)
```

The range of IP's of my virtual private network can be found by checking one of the instances and clicking onto the VPC link marked in pink (see screenshots below). From there you can find the IP range in the column "IPv4 CIDR" marked in pink.

| Name | Instance ID | Instance state | Instance type | Status check | Alarm status |
|---|---------------------|----------------|---------------|--------------|--------------|
| <input checked="" type="checkbox"/> main_centos | i-00a5ead5840cdb01e | Stopped | t2.micro | - | No alarms |
| <input type="checkbox"/> wn_1 | i-0863cc33834b6c3ee | Stopped | t2.micro | - | No alarms |
| <input type="checkbox"/> wn_2 | i-0b7176125f178249e | Stopped | t2.micro | - | No alarms |

| | | |
|---|----------------------|------------------------|
| Instance: i-00a5ead5840cdb01e (main_centos) | | |
| Details | Security | Networking |
| ▼ Instance summary Info | | |
| Instance ID | Public IPv4 address | Private IPv4 addresses |
| i-00a5ead5840cdb01e (main_centos) | - | - |
| Instance state | Public IPv4 DNS | Private IPv4 DNS |
| Stopped | - | - |
| Instance type | Elastic IP addresses | VPC ID |
| t2.micro | - | vpc-1284076f |

I changed the rights to the shared directory to grant read, write and execute access for all users.

```
1 sudo chmod 777 /data
```

Your VPCs (1/1) Info 🔄 Actions Create VPC

Filter VPCs < 1 > ⚙️

VPC ID: vpc-1284076f ✕ Clear filters

| <input checked="" type="checkbox"/> | Name | VPC ID | State | IPv4 CIDR | IPv6 CIDR (Network border group) |
|-------------------------------------|------|--------------|-----------|---------------|----------------------------------|
| <input checked="" type="checkbox"/> | - | vpc-1284076f | Available | 172.31.0.0/16 | - |

vpc-1284076f 📄 📄 📄

Details CIDRs Flow logs Tags

Details

| | | | |
|---|-----------------------------------|----------------------------------|---------------------------------------|
| VPC ID vpc-1284076f | State Available | DNS hostnames Enabled | DNS resolution Enabled |
| Tenancy Default | DHCP options set dopt-fa567a80 | Main route table rtb-0d2b2573 | Main network ACL acl-7ef31902 |
| Default VPC Yes | IPv4 CIDR 172.31.0.0/16 | IPv6 pool - | IPv6 CIDR (Network border group) - |
| Route 53 Resolver DNS Firewall rule groups | - | - | - |

1.4 Configuring the Worker Node

For configuring the worker node we first instantiated a t2.micro machine with CentOS Linux 8. Again, for better orientation, we changed the prompt to:

```
[centos@ wn_1 :~$ echo $PS1
\[\033[01;32m\]\u@ wn_1 \[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]$
```

For the installation of HTCondor on this worker node the same procedure was followed as described in section 1.2. The `/etc/condor/condor.config` file however, had to be configured differently for the worker node by appending the following lines to it:

```
1 # Main Node IP
2 CONDOR_HOST = <Main_Node_private_IP>
3
4 # Worker Node config
5 DAEMON_LIST = MASTER, STARTD
6
7 HOSTALLOW_READ = *
8 HOSTALLOW_WRITE = *
9 HOSTALLOW_ADMINISTRATOR = *
```

To be able to mount the shared volume on the worker node the NFS client library was installed issuing the commands below:

```
1 yum install nfs-utils
```

To ensure `/data` was mounted upon boot the following line was added to the worker nodes `/etc/fstab`:

```
1 <private_IP_of_MAIN_node>:/data /data nfs defaults 0 0
```

And then we mounted it with:

```
1 mount -a
```

Even though the commands were successful, we ran `ll /data` from the worker node. We also created a file from the worker to verify that the permissions were set correctly. Then we installed the `libsvm` application on the worker node via [1]:

```
1 sudo yum install libsvm
```

2 Installing Docker on Main and Worker Nodes and Creating an Image on my Local Machine

Docker is an open-source software project that is used for packing and shipping applications in containers and automating the deployment thereof [5, 6]. First we created a Dockerfile to create an image on my local machine:

```
1 FROM centos:8
2 RUN yum install -y epel-release
3 RUN yum install -y libsvm
```

We built and tagged the image which can also be found online on [Docker Hub](#) [6]:

```
1 ila@spacebar docker % docker tag centos8_libsvm:1.0.0 ilante/centos8_libsvm:1.0.0
2 ila@spacebar docker % docker image ls
3 REPOSITORY          TAG          IMAGE ID      CREATED        SIZE
4 ilante/centos8_libsvm 1.0.0        dc2ce4cbb529  13 minutes ago 286MB
5 centos8_libsvm       1.0.0        dc2ce4cbb529  13 minutes ago 286MB
```

Lastly we pushed the image to Docker Hub such that the worker in my IaaS can later pull it.

```
1 ila@spacebar docker % docker push ilante/centos8_libsvm:1.0.0
2 The push refers to repository [docker.io/ilante/centos8_libsvm]
3 d86f07b0b8d9: Pushed
4 6a016089002c: Pushed
5 2653d992f4ef: Pushed
6 1.0.0: digest: sha256:c92ae898ac81f930e60f095e9c258d9f86388517b3f2c76701db16d28d2fc107 size:
   953
```

From the worker node we ensured functionality by issuing both the `svm-predict` and the `svm-train` commands, as shown in the following snippet copied and pasted from the terminal:

```
1 centos@ wn_1 :~$ sudo su -
2 Last login: Fri Jun 01 20:27:40 UTC 2021 on pts/0
3 [root@ip-172-31-9-58 ~]# docker run -it ilante/centos8_libsvm:1.0.0
4 [root@4be262e7ed64 /]# svm-predict
5 Usage: svm-predict [options] test_file model_file output_file
6 options:
7 -b probability_estimates: whether to predict probability estimates, 0 or 1 (default 0); for
   one-class SVM only 0 is supported
8 -q : quiet mode (no outputs)
9 [root@4be262e7ed64 /]# svm-train
10 Usage: svm-train [options] training_set_file [model_file]
11 options:
12 -s svm_type : set type of SVM (default 0)
13   0 -- C-SVC      (multi-class classification)
14   1 -- nu-SVC     (multi-class classification)
15   2 -- one-class  SVM
16   3 -- epsilon-SVR (regression)
17   4 -- nu-SVR     (regression)
18 -t kernel_type : set type of kernel function (default 2)
19   0 -- linear: u'*v
20   1 -- polynomial: (gamma*u'*v + coef0)^degree
21   2 -- radial basis function: exp(-gamma*|u-v|^2)
22   3 -- sigmoid: tanh(gamma*u'*v + coef0)
23   4 -- precomputed kernel (kernel values in training_set_file)
24 -d degree : set degree in kernel function (default 3)
25 -g gamma : set gamma in kernel function (default 1/num_features)
26 -r coef0 : set coef0 in kernel function (default 0)
27 -c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
28 -n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
29 -p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
30 -m cachesize : set cache memory size in MB (default 100)
31 -e epsilon : set tolerance of termination criterion (default 0.001)
32 -h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
33 -b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or
   1 (default 0)
34 -wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
35 -v n: n-fold cross validation mode
36 -q : quiet mode (no outputs)
```

2.1 Configuring Condor to use Docker

To allow HTCondor to run Dockerized jobs we changed the configuration of the worker nodes by adding the condor user to the Docker group:

```
1 sudo usermod -aG docker condor
```

Thereafter we updated the condor configuration on the worker nodes to allow the shared volume to be mounted on the containers by appending the following lines to `/etc/condor/condor.config`:

```
1 ##-----  
2 ## Docker Volume Config  
3 ##-----  
4 # Defining only one docker volume  
5 DOCKER_VOLUMES = SHARED_DATA  
6 # Assigning the /data directory to docker volume  
7 DOCKER_VOLUME_DIR_SHARED_DATA = /data  
8 # Mount SHARED_DATA volume in all containers handled by HTcondor  
9 DOCKER_MOUNT_VOLUMES = SHARED_DATA
```

To update the status of the cluster we ran a command for concluding the reconfiguration.

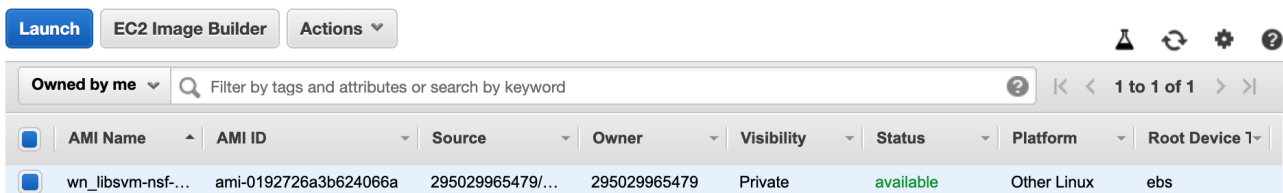
```
centos@main:~/BDP-project-aws-main$ condor_reconfig  
Sent "Reconfig" command to local master
```

For verifying that the reconfiguration was successful we checked the cluster for Docker enabled nodes:

```
[root@ip-172-31-1-180 ~]# condor_status -l | grep -i docker  
DockerVersion = "Docker version 20.10.3, build 48d30b5"  
HasDocker = true  
HasDockerVolumeSHARED_DATA = true  
StarterAbilityList = "HasDocker,HasJICLocalConfig,HasSelfCheckpointTransfers,HasFileTransferPluginMethods,HasJICLocalStdin,HasTransferInputRemaps,HasPerFileEncryption,HasFileTransfer,HasVM,HasJobDeferral,HasTDP,HasReconnect,HasMPI"  
DockerVersion = "Docker version 20.10.3, build 48d30b5"  
HasDocker = true  
HasDockerVolumeSHARED_DATA = true  
StarterAbilityList = "HasDocker,HasJICLocalConfig,HasSelfCheckpointTransfers,HasFileTransferPluginMethods,HasJICLocalStdin,HasTransferInputRemaps,HasPerFileEncryption,HasFileTransfer,HasVM,HasJobDeferral,HasTDP,HasReconnect,HasMPI"  
----- 172.31.1.180 74 █
```

2.2 Expanding the Cluster

Given that workers are stateless, while the master is stateful, the number of workers can be changed without taking the cluster offline. To be able to expand my cluster with minimal effort we created an image of my fully configured worker node. From this image the administrators of the cluster can scale the cluster horizontally according to their needs (and financial resources). For the scope of this project we contented myself with second worker node that did not require any manual configuration, created from the image you can see below.



| AMI Name | AMI ID | Source | Owner | Visibility | Status | Platform | Root Device 1 |
|------------------|-----------------------|------------------|--------------|------------|-----------|-------------|---------------|
| wn_libsvm-nsf... | ami-0192726a3b624066a | 295029965479/... | 295029965479 | Private | available | Other Linux | ebs |

3 Submitting an Support Vector Machine Test Job to my HTCondor Cluster

Support Vector Machines (SVM) are a supervised machine learning algorithm first published as *Support Vector Networks* by Corinna Cortes and Vladimir Vapnik in 1995 [7]. The library that was implemented in this project is called `libsvm` [1]. The SVM algorithm achieves classification by finding the optimal hyperplane that separates two classes while maximizing the *margin* defined as the distance between the support vectors [2]. The maximum margin hyperplane is used as a decision boundary for classifying new data points. Inspired by the extremely long training times of SVM used during another project we decided to choose this application. For this algorithm it is paramount to find the best values for the hyperparameters C and γ by performing a grid search. The use case of the grid search shall be explored for a use case when predicting the expected costs.

3.1 Choosing Input Executable and Job Files

For the scope of this project we chose six input files of increasing size to train a SVM model which was subsequently used to predict protein secondary structure from a single testing file. These files were generated in a previous [project](#) in the following fashion: The `libsvm` programs (predict and train) take input data in the format of numerical vectors for both training and prediction. Thus we used a [script](#) that transforms the data from a profile windows, into a 20 (amino acids) by 17 (window size) vector. These 340 feature vectors were subsequently matched to the class of the central residue Helix, Strand or Coil (H, E or C). The secondary structure was then encoded as 1 for H, 2 for E and 3 for C. No additional larger testing files for prediction were used, because bulk time of the application is spent on testing, while in a real use case the file to be predicted would not have a significant influence.

The test job consisted in training six SVM models from the files numbered 0 – 5 shown below with the `svm-train` function of `libsvm`. Each of the six models was subsequently used to predict the same input file `test_data.svm`. The sizes of the minimal input files were as follows:

```
1 centos@main:~/BDP-project-aws-main/in_train_tiny$ ll -h *.svm
2 -rw-rw-r--. 1 centos centos 16M Jun 17 20:52 test_data.svm
3 -rw-rw-r--. 1 centos centos 5.1M Jun 16 16:44 train0.svm
4 -rw-rw-r--. 1 centos centos 11M Jun 16 16:47 train1.svm
5 -rw-rw-r--. 1 centos centos 16M Jun 16 16:50 train2.svm
6 -rw-rw-r--. 1 centos centos 21M Jun 16 16:52 train3.svm
7 -rw-rw-r--. 1 centos centos 26M Jun 16 16:53 train4.svm
8 -rw-rw-r--. 1 centos centos 31M Jun 16 16:54 train5.svm
```

The simple `bash` script that serves as the executable for `condor` is shown below:

```
1 #!/usr/bin/bash
2 #####
3 # Run svm-train then svm-predict
4 #####
5
6 #####
7 # 1 arg: <path_to_input_parameter>/train0.svm
8 # 2 arg: <path_to_output_parameter>/svm_outputs/train/train0.model
9 # 2 arg: serves also as input to svm-predict
10 # 3 arg: <path_to_input_parameter>/in_predict_tiny/test0.svm
11 # 4 arg: <path_to_output_parameter>/svm_outputs/predict/prediction0.out
12 #####
13 # Measure time of each jobinstance
14 SECONDS=0
15 svm-train -c 2 -g 0.5 -m 1024 $1 $2 && \
16 svm-predict $3 $2 $4
17 echo "Duration" $SECONDS
```

The job file (`svm_test_predict.job`) for HTCondor was set to request 1024 MB which is identical to the memory (`-m 1024`) requested in line 15 of the `svm_test_predict.job` file (above). We requested 1 CPU, as `libsvm` is not a multi-threaded application which means We cannot take advantage of more than one CPU.

```
1 #####
2 ##### svm-train #####
3 ##### svm-predict #####
4 #####
5
6 ##### The program to be executed #####
7
8 Executable = svm_test_predict_exec.sh
9 #####
10 # Deciding how much Resources must be allocated
11 #####
12 #
13 request_memory = 1024
14 request_cpus = 1
15
16 ##### Input Sandbox #####
17
18 Input      = /home/centos/BDP-project-aws-main/in_train_tiny/train$(Process).svm
19 # Can contain standard input
20
21 transfer_input_files = /home/centos/BDP-project-aws-main/in_train_tiny/test_data.svm
22 # Because I have more than 1 input file I need transfer_input_files!
23 # using sb for input and shared vol for .model file
24
25 #####
26 # Need to rename /data/svm_outputs/train$number for
27 # consecutive runs
28 #
29 Arguments = "train$(Process).svm /data/svm_outputs/train/train$(Process).model test_data.svm
              prediction$(Process).out"
```



```

30 # train$(Process).svm & test$(Process).svm are passed
31 # via the sandbox
32 #
33 #/data/svm_outputs/train/train$(Process).model
34 # are passed via the shared volume to share the model
35 # with members of the team publicly
36 ##### Output Sandbox #####
37
38 Log          = condor_out/train$(Process).log
39 # will contain condor log
40
41 Output       = condor_out/train$(Process).out
42 # will contain the standard output
43
44 Error        = condor_out/train$(Process).error
45 # will contain the standard error
46 transfer_output_remaps = "prediction$(Process).out=out_predict_tiny/prediction$(Process).out"
47
48 ##### condor control variables #####
49
50 should_transfer_files = YES
51 when_to_transfer_output = ON_EXIT
52
53 Universe = vanilla
54
55 #####
56
57 Queue 6

```

I ran six instances of the test job on the cluster. Given the settings described previously, the jobs ran in pairs, one on each worker node (see screenshot below).

```

centos@main:~/BDP-project-aws-main$ condor_submit svm_test_predict.job
Submitting job(s).....
6 job(s) submitted to cluster 22.
centos@main:~/BDP-project-aws-main$ condor_q

```

```

-- Schedd: ip-172-31-1-180.ec2.internal : <172.31.1.180:9618?... @ 06/17/21 21:35:59
OWNER BATCH_NAME   SUBMITTED   DONE    RUN    IDLE  TOTAL JOB_IDS
centos ID: 22      6/17 21:35    _       2      4       6 22.0-5

```

```

Total for query: 6 jobs; 0 completed, 0 removed, 4 idle, 2 running, 0 held, 0 suspended
Total for centos: 6 jobs; 0 completed, 0 removed, 4 idle, 2 running, 0 held, 0 suspended
Total for all users: 6 jobs; 0 completed, 0 removed, 4 idle, 2 running, 0 held, 0 suspended

```

The cluster completed the task successfully producing 6 SVM models and 6 predictions:

```

centos@main:~$ ll /data/svm_outputs/train/train?.model
-rw-r--r--. 1 nobody nobody 5230358 Jun 18 21:23 /data/svm_outputs/train/train0.model
-rw-r--r--. 1 nobody nobody 5990309 Jun 17 21:37 /data/svm_outputs/train/train1.model
-rw-r--r--. 1 nobody nobody 6453166 Jun 17 21:39 /data/svm_outputs/train/train2.model
-rw-r--r--. 1 nobody nobody 6850698 Jun 17 21:41 /data/svm_outputs/train/train3.model
-rw-r--r--. 1 nobody nobody 7256416 Jun 17 21:44 /data/svm_outputs/train/train4.model
-rw-r--r--. 1 nobody nobody 7611740 Jun 17 21:47 /data/svm_outputs/train/train5.model

```

The duration of each task ranged from 100 to 383 seconds, with an average of 227.7 seconds.

```

centos@main:~/BDP-project-aws-main/condor_out$ grep "Duration" *
train0.out:Duration 100
train1.out:Duration 154
train2.out:Duration 204
train3.out:Duration 268
train4.out:Duration 336
train5.out:Duration 383

```

4 Running an Identical Job in Docker Containers

To run an identical job in Docker Containers, we modified the job file for HTCondor, in order to ensure that none of the previously generated data would be overwritten. This was done in part by creating a new output directory reserved for the Docker job output (logs, errors and output). Note that the universe (line 60) has to be set to docker. As the model was saved in the shared volume, here the name of the output was changed to make it distinguishable from the other models:


```

1 #####
2 ##### Docker #####
3 ##### svm-train #####
4 ##### svm-predict #####
5 #####
6
7 ##### The program to be executed #####
8
9 Executable = svm_test_predict_exec.sh
10
11 #####
12 ##### Name of Docker image #####
13 #####
14 docker_image = ilante/centos8_libsvm:1.0.0
15
16 #####
17 # Deciding how many Resources must be allocated
18 #####
19 #
20 request_memory = 1024
21 request_cpus = 1
22
23 ##### Input Sandbox #####
24
25 Input      = /home/centos/BDP-project-aws-main/in_train_tiny/train$(Process).svm
26 # Can contain standard input
27
28 transfer_input_files = /home/centos/BDP-project-aws-main/in_train_tiny/test_data.svm
29 # Because I have more than 1 input file I need transfer_input_files!
30 # using sb for input and shared vol for .model file
31
32 #####
33 # Need to rename /data/svm_outputs/train$number for
34 # consecutive runs
35 #
36 Arguments = "train$(Process).svm /data/svm_outputs/train/train_docker$(Process).model
37             test_data.svm prediction$(Process).out"
38 # train$(Process).svm & test$(Process).svm are passed
39 # via the sandbox
40 #
41 #/data/svm_outputs/train/train$(Process).model
42 # are passed via the shared volume to share the model
43 # with members of the team publicly
44 ##### Output Sandbox #####
45
46 Log      = condor_out_docker/train$(Process).log
47 # will contain condor log
48
49 Output    = condor_out_docker/train$(Process).out
50 # will contain the standard output
51
52 Error      = condor_out_docker/train$(Process).error
53 # will contain the standard error
54 transfer_output_remaps = "prediction$(Process).out=out_predict_tiny/prediction$(Process).out"
55
56 ##### condor control variables #####
57
58 should_transfer_files = YES
59 when_to_transfer_output = ON_EXIT
60
61 Universe = docker
62
63 #####
64 Queue 6

```

Each of the job instances ran successfully. The duration of each of the six job was slightly longer than in the non-containerized version (see section 2.2); ranging from 105 to 397 seconds. Given that the cluster is composed of virtual machines, running Docker adds another layer of abstraction and isolation which may have led to this small performance loss.

Depending on the needs of the job Docker can provide higher flexibility and reproducibility in case of highly customized binaries. As previously mentioned, by the time of writing, `libsvm` does not provide a multi-threaded implementation for any major linux distribution. Chang and Lin do however provide [instructions](#) on how to compile such version in house [1]. Such implementation may then be put into a container and deployed to the cluster for convenience of its users. Using this hypothetical multi-threaded implementation of `libsvm` could greatly reduce the time for training the models.

```
centos@main:~/BDP-project-aws-main/condor_out_docker$ grep "Duration" *
train0.out:Duration 105
train1.out:Duration 157
train2.out:Duration 214
train3.out:Duration 277
train4.out:Duration 351
train5.out:Duration 397
```

4.1 Data Management Model

The data model is based on the following assumptions: The input data is sensitive data and therefore cannot be shared with other users of the infrastructure. The logs, outputs and errors are not shared for the scope of not polluting the shared volume with this kind of volatile data. The models produced by the training, however, cannot be traced back to the sensitive input, and are to be shared with the other users for classification purposes. Given these assumptions and the infrastructure, described in section 1, the data model is the following: The executable, the input files for training the model and the test file for validating the model `test_data.svm` are passed via the sand box. This choice is justified by the size of the data and the number of jobs. Each of the generated models are saved to the shared volume as they must be available to other users. The standard output, condor log and condor error files are retrieved via the output sandbox to be available to the job submitter for inspection and debugging reasons.

5 Evaluating the Execution Time

As previously mentioned, the input chosen was of increasing size. From this data the RAM utilization and execution time by input size was plotted using several python libraries ([GitHub](#)). The plots show that the jobs scale linearly which is congruent with the findings of Bottou et al. who found that running an SVM, space and time complexity are linear with respect to the number of support vectors [8].

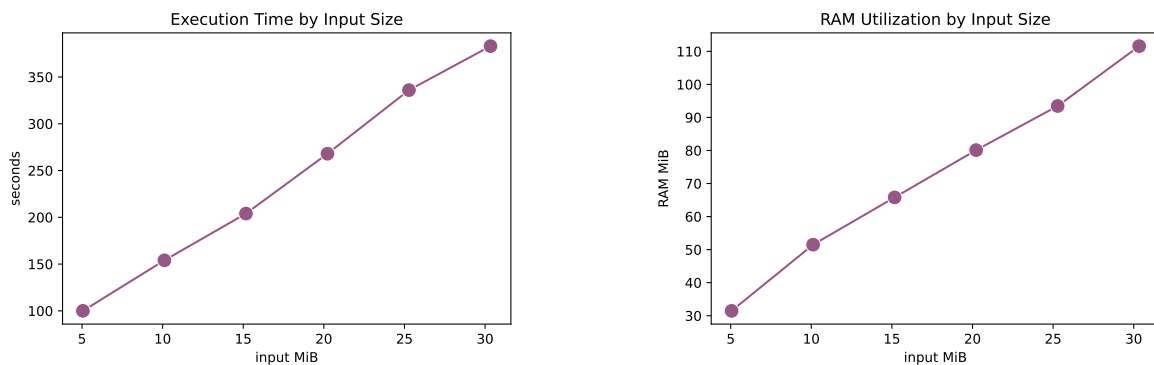


Figure 1: Time (left) and RAM (right) used for each of the six input files

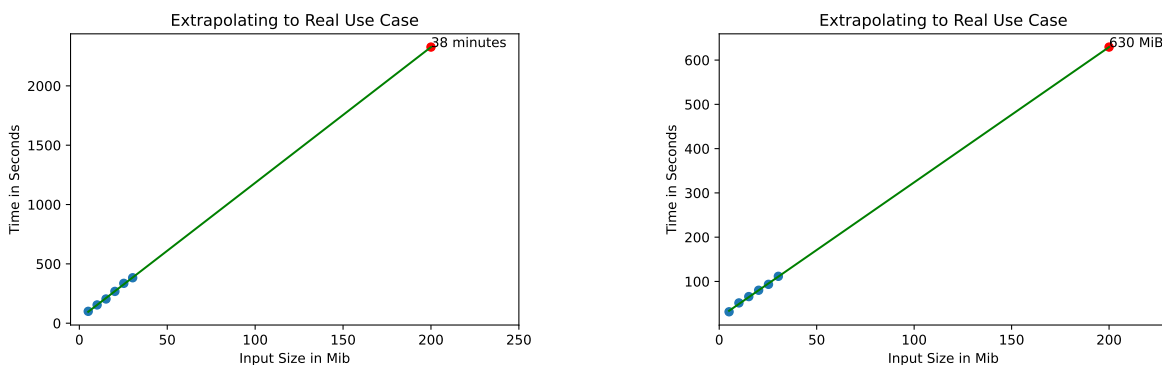


Figure 2: In a real use case with a training input of 200 MiB the expected runtime is 38 minutes and the expected RAM usage is 630 MiB

5.1 Extrapolating to a Realistic use case

Training a model with an input file of 200 MiB, which would correspond to roughly 1200 sequences. This number was chosen, based on the number of sequences selected by the developers of JPred4, a protein secondary structure prediction server, as it is critical to have training data that are representative of the protein space in order to be able to generate a model that is capable of generalizing to unseen data. [9].

5.2 Expected Costs of a Non-Trivial Application

The execution of a grid search for determining the best hyperparameters is a plausible non-trivial use case that could be addressed by this HTCondor cluster that we have developed. To execute an effective grid search entails to train SVM models using 10 combinations of the hyperparameters C and γ . For each of the combinations a five-fold cross validation should be carried out based splitting the training set that holds 1200 sequences. This approach would require the training 50 different models.

One of the advantages of the IaaS is that horizontal up-scaling can be done easily as described in section 2.2. This would allow us to replicate the worker nodes to achieve a cluster of 25 worker nodes with 2 vCPUs per node. Each of the workers would run 2 jobs in parallel amounting to an up-time of 38 minutes for running the entire task of training 50 models.

These 25 worker nodes would be orchestrated by a main node of the same make and size, thus the cluster would be comprised of 26 identical nodes. This hypothetical set up of such a large number of small nodes is preferred over just two large worker nodes given that the [AWS pricing](#) shows that machines endowed with high numbers of cores are significantly more expensive than many small machines amounting to the same number of cores (see figure below) [10]. Further, in case of job failure, we can take advantage of the pay-per-use model in the AWS cloud: The number of nodes needed to complete the task would stay up and running, while nodes that have produced a model successfully may be shut down and not payed for.

The proposed set-up of 26 t4g.medium nodes would require merely \$ 0.553 for completing the task in 38 minutes, provided it ran successfully on the first attempt. The calculation of the expected cost is outlined in the [supplementary materials on GitHub](#). It is important to note that the main bottle neck of the proposed infrastructure is represented by the data movement from the main node to the worker nodes 4. Here the theoretical run-time of 38 minutes does not consider how moving the larger input and output data to such a high number of nodes would affect the actual run-time.

The storage space required for the generated models does not exceed 7650 MB (7.12 GiB) which is well within the capacity of the volume already set up for the NFS (30 GiB). The storage on general-purpose SSD EBS devices costs \$ 0.1 per Gb per month but the expected cost for moving data out of the AWS ecosystem have not been considered. Another thing that might increase cost is the local GDPR regulations might call for the cluster being in the availability zone of eu-south-1 in the region of Milan instead of US-east-1b in N. Virginia.

| Instance name ▲ | On-Demand hourly rate ▼ | vCPU ▼ | Memory ▼ | Storage ▼ | Network performance ▼ |
|-----------------|-------------------------|--------|----------|-----------|-----------------------|
| t4g.medium | \$0.0336 | 2 | 4 GiB | EBS Only | Up to 5 Gigabit |
| m6g.12xlarge | \$1.848 | 48 | 192 GiB | EBS Only | 12 Gigabit |

Figure 3: The screenshot taken from the AWS pricing site shows that it is more cost effective to instantiate many nodes with 2 CPUs, rather than one large machine with many.

6 Applying Concepts from the BDP2 Course to the Non-Trivial Application

The architecture described may be improved by implementing auto-scaling the number of worker nodes in the cluster. Further, it would be possible to make use of Docker Hub and GitHub for configuring autobuilds of Docker images which allow automatized updating of an image when ever modifications are made to an existing container via a simple commit. Additional modification could be done by using Docker Swarm or Kubernetes cluster, to orchestrate a multitude of containers. Another possible solution is to use a serverless computing system like AWS Lambda or OpenFaaS (possibly on a self-scaling Kubernetes cluster). The management of the latter requires however specialized knowledge and additional tools such as `kubect1` CLI, while it is more rich in functionality than Docker Swarm.

Considering the file system, it is worth noting that by configuring Docker to include a scratch directory in the host machine where the job is initiated, by bind mounting a volume permanently, we have tied the containers to the host. We could avoid this by using AWS S3 bucket, which allows to obtain unique links to buckets of files. The S3 bucket has the advantage of employing a pay per use pricing model, as opposed to the EBS volumes where we pay for the entire capacity, regardless of space used.

References

1. Chang, C.-C. & Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* **2**, 27:1–27:27 (2011).
2. Gholami, R. & Fakhari, N. en. in *Handbook of Neural Computation* (eds Samui, P., Sekhar, S. & Balas, V. E.) 515–535 (Academic Press, Jan. 2017). ISBN: 978-0-12-811318-9. <http://www.sciencedirect.com/science/article/pii/B9780128113189000272> (2021).
3. Cerf, V. & Kahn, R. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications* **22**. Conference Name: IEEE Transactions on Communications, 637–648. ISSN: 1558-0857 (May 1974).

4. Thain, D., Tannenbaum, T. & Livny, M. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience* **17**, 323–356 (2005).
5. Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* **2014**, 2 (2014).
6. *Docker Hub* <https://hub.docker.com/> (2021).
7. Cortes, C. & Vapnik, V. Support-vector networks. en. *Machine Learning* **20**, 273–297. ISSN: 1573-0565. <https://doi.org/10.1007/BF00994018> (2021) (Sept. 1995).
8. Bottou, L., Chapelle, O., DeCoste, D. & Weston, J. *Support Vector Machine Solvers* in (2007).
9. Drozdetskiy, A., Cole, C., Procter, J. & Barton, G. J. JPred4: a protein secondary structure prediction server. *Nucleic Acids Research* **43**, W389–W394. ISSN: 0305-1048. <https://doi.org/10.1093/nar/gkv332> (2020) (July 2015).
10. *EC2 On-Demand Instance Pricing – Amazon Web Services* <https://aws.amazon.com/ec2/pricing/on-demand/> (2021).

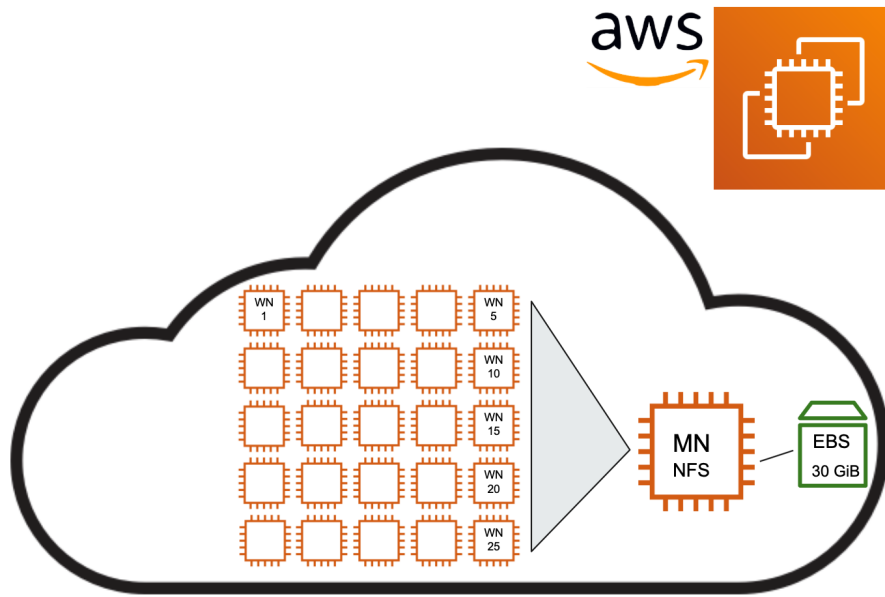


Figure 4: Proposed infrastructure of cluster: 25 worker nodes (WN) and 1 main node (MN) all of the type t4g.medium with 2 vCPU and 4 GiB of RAM. The main bottle neck of the infrastructure lies in the data transfer between main node and worker nodes and may lead to a higher run-time than the one proposed we modelled with the data available to us.

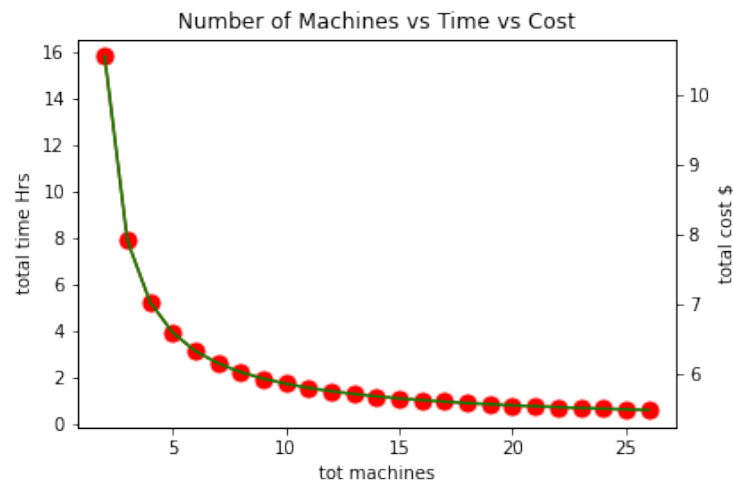


Figure 5: Note that the price and run-time remain roughly the same once the cluster reaches a size of 20 nodes.