Supercomputer tools

This describes some information about the super computer tools that are available at IU as well as information on how to use these tools.

- Containers on IU supercomputers
- Supercomputer info
- Notes on Screen
- AMD optimized crfsuite

Containers on IU supercomputers

Karst, Carbonate and Big Red II have the singularity package available which allows users to run docker or singularity images on the supercomputers. This can be a great way to run programs that are not installed on the supercomputers.

To use singularity, load the module

```
module add singularity
```

Then pull down an image. You can pull an image from dockerhub or singularity's hub.

```
singularity pull docker://julia:latest
```

Unlike docker, singularity creates a file that contains the image specification. To run the container use the image file generated by your pull command.

```
# singularity exec <local image> <cmd>
singularity exec julia-latest.simg julia
```

For more information, please see the singularity documentation.

Supercomputer info

Carbonate node: 710.223 GFLOPS on intel mkl linpack

If you need to get 32GB of VRAM on Carbonate-dl

```
From a previous help ticket: "dl[11,12] are in fact v100-16GB parts. If the user needs v100-32GB they’ll need to add a “-w <node>“, where <node> is dl1 or dl2."
```

Notes on Screen

It is a good idea to use screen sessions

On super computers like those at IU, it is essential to use a screen session for submitting interactive jobs. Screen basically emulates a user sitting at the computer screen. It accepts output and can give input. However, you can detach from screen sessions and log out from the computer without causing running jobs to terminate.

Here are some notes provided by IU's Knowledge Base

When you can't re-attach to your screen session after a lost connection

In some cases, your previous screen session may not have detached properly when you lost your connection. If this happens, you can detach your session manually.

To see your existing screen sessions, enter:

```
screen -list
```

This will display a list of your current screen sessions. For instance, if you had one attached and one dead screen, you would see:

```
 There are screens on: 25542.pts-28.hostname (Dead ???) 1636.pts-21.hostname (Attached)
Remove dead screens with 'screen -wipe'. 2 Sockets in /tmp/screens/S-username.
```

To detach an attached screen, enter:

```
screen -D
```

If you have more than one attached screen, you can specify a particular screen to detach. For example, to detach the screen in the above example, you would enter:

```
screen -D 1636.pts-21.hostname
```

Once you've done this, you can resume the screen by entering the screen -r command.

(In the above example, the dead screen isn't causing problems, but you should probably enter the screen -wipe command to get rid of it.)

AMD optimized crfsuite

Problem

The bundled binary from python-crfsuite and sklearn-crfsuite performs rather badly on amd processors.

For example, in a trial run with training a small pos tagger on an AMD Epyc 7601, each iteration in the hyperparameter search took about 1 minute 15 seconds. This is only with a small training set of about 400 sentences. With the full 6,000 sentences available it takes upwards of 4 days to finish a single run. Rough.

However, on a dual intel E5-2680 system (16 cores at 3.2 Ghz all core turbo), the performance is much faster. On that same small dataset of 400 sentences, each iteration takes about 30 seconds, the entire hyperparameter search over 50 combinations takes 2 minutes.

This appears to be due to the fact that the binaries that ship with sklearn-crfsuite were compiled on an intel platform.

Solution

To fix this, I created a fork of python-crfsuite that uses the avx2 instructions available on amd's zen processors (this should also help with newer intel processors that have avx extensions).

This fork is available at https://github.com/ksteimel/python-crfsuite.git

To use this, clone the repo

```
git clone --recurse-submodules https://github.com/ksteimel/python-crfsuite.git
```

Then change into the new directory.

```
cd python-crfsuite
```

it's a good idea to create a virtual environment so if you decide you don't want to use this version, you don't have to.

```
virtualenv -p <your python location> <path to virtualenv>source <path to
virtualenv>/bin/activate
```

Then, we need to build the package.

```
python setup.py build
python setup.py install
```

Now you should have an optimized build of python-crfsuite You can then install sklearn-crfsuite .

```
pip install sklearn-crfsuite
pip install scikit-learn
```

To get out of your virtual environment run,

```
deactivate
```

How did we do?

The whole point of this was to speed up performance on amd processors with crfsuite. If we go back to our small training dataset, we see a substantial boost in performance.

We've now gone from one minute 15 seconds per iteration to only 30ish seconds per iteration.

It may seem somewhat shocking that the amd processor is only about as fast per iteration as the intel processor. However, the amd processor has double the number of cores (32 instead of 16) with a lower clock speed. (2.2 Ghz for the epyc processor compared to 3.2 for the intel procesor). If we look at the entire grid search across 50 parameters, we see the core advantage of the epyc

processor emerge: the grid search finished in only 1.1 minutes on the AMD processor while it took 2.0 minutes on the pair of intel processors.

Not too shabby for 2 minutes of work.