VQA

Information Guide

Version: 0.0.1

March 24, 2021

Table of Contents

Version History

S. No.	Version Number	Author(s)	Date
1	0.0.1	Atsushi Kajita, G R Ramdas Pillai	03/24/2021

General Information

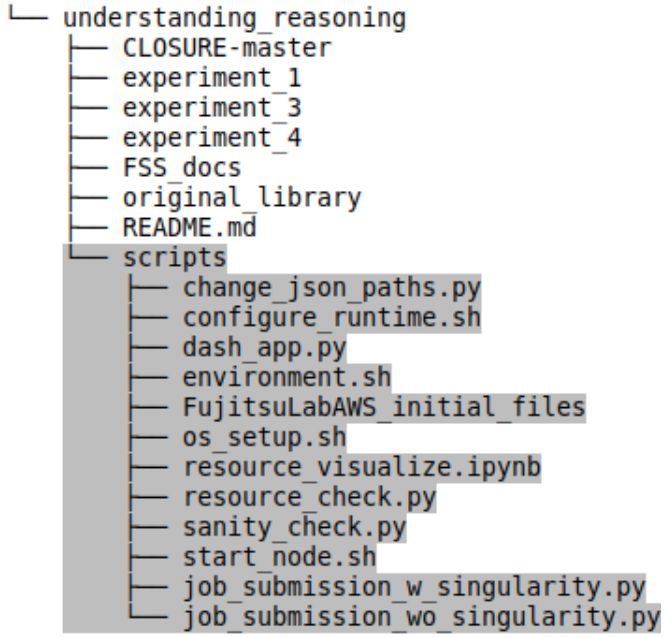
The VQA codebase consists of utilities targeted towards adding high level parallelization. There are different methods that can be used to achieve this parallelization and are discussed under the 'Execution Methodologies' section.

The code information is as stated below:
Gitlab Link: https://gitlab.com/tomotake/understanding_reasoning.git
Branch: **fixstars-dev**

The processed results are stored at:
Path: `/om5/user/gpillai/shared_understanding_reasoning/results/`
Backup Path: `/om2/user/gpillai/shared_understanding_reasoning/results/`

Common Scripts Overview

All the scripts used in this project are stored at:



The common scripts applicable to any methodology are:

- **change_json_paths.py**: The train.json copied from any other location (e.g. OpenMind) will have different paths for output and datasets. This simple script changes the paths for below. It is recommended to have the datasets in below paths to avoid access issues on OpenMind.
    - Output_path to `${PWD}/results/train_<index>`
    - Dataset path to `${PWD}/data_generation/datasets/dataset_<id>`
- **dash_app.py**: This is the DASH application script responsible for showing the progress table. The default port used is **42126**. Change it according to port availability.
- **resource_check.py:** Each set of experiments may have potentially different sets of system resource requirements. For example, case_2 experiments need 11 GB of main memory, case_3 and case_4 need 27 GB of main memory. It is important to know this information beforehand to appropriately use the available computational resources. This script iterates through the train.json and understands the compute resources necessary for each of the experiment indexes.
- **resource_visualize.ipynb**: This jupyter notebook visualizes the results of the resource_check.py file. This should be executed once the above script has finished execution.
- **sanity_check.py:** A simple script to verify if the models have really completed all the expected iterations before marking flag_completed. Discrepancies happen only when there are issues with copying results from multiple nodes. (For example, we met the issue with AWS. The issue occurred due to a mis sync between head node and one of the node's results).

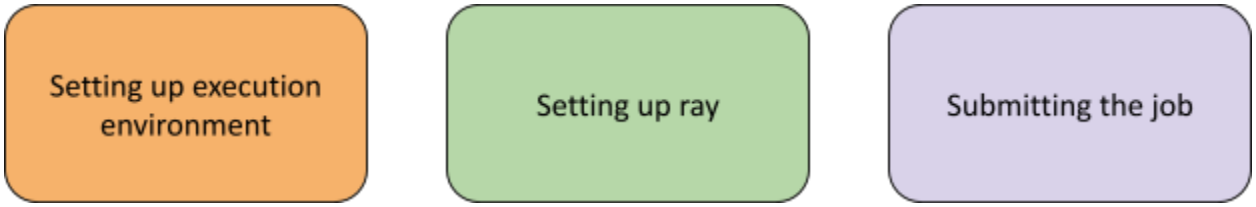
The other scripts are specific to each methodology to run the experiments and will be covered in below topics.

Execution Methodologies

As mentioned earlier, there are mainly three execution methodologies that we can consider to run the VQA experiments efficiently.

- With SLURM and singularity (OpenMind)
- Barebones setup(DGX, AWS)

Each execution methodology is divided into 3 subparts:



Let's look at how these change for each of the methods.

Note: All below methods are tested on x86 platform and are not fully supported for the ppc64le architecture. See Appendix for more information on ppc64le systems (Satori).

With SLURM and singularity

This method is only applicable in cases where the clustering is based on SLURM. An example of such a cluster is OpenMind. It is highly recommended to use a terminal emulator like tmux for uninterrupted execution.

Setting up execution environment:

With SLURM, we make use of the acceleration codebase to set up the ray. The underlying python environment is created using Anaconda. The packages to install are:

```
conda create -n acceleration python=3.8
pip install PTable --progress-bar off
pip install opencv-python --progress-bar off
pip install -U ray==1.1.0 --progress-bar off
pip install nvgpu --progress-bar off
pip install reprint --progress-bar off
pip install psutil --progress-bar off
pip install plotly dash dash-core-components dash_html_components dash_table
--progress-bar off
conda install -y -q scikit-image
conda install -y -q -c anaconda cython
conda install -y -q jupyter
conda install -y -q numba
conda install -y -q tqdm
conda install -y -q h5py
```

Note that we use ray version 1.1.0. Ray is an actively developing project and has poor backward compatibility with respect to interface. However, the errors are generally self explanatory and have a good support community.

Once the conda environment is ready, we can clone the acceleration repository.

Gitlab link: https://gitlab.com/tomotake/acceleration.git
Branch: fixstars-dev-vqa

Refer to this User Manual for detailed instructions of the acceleration codebase.

We will go through the commands necessary to get a working setup without getting into the details.

Steps:

1. Clone the repository:
   `git clone -b fixstars-dev-vqa https://gitlab.com/tomotake/acceleration.git`
2. Change the 'configure_runtime.sh' as per requirements. Specifically, the conda environment name and all ports. Note that if more than one user is running ray on the system, the subsequent users will need to change the ports.
3. Change the OpenMind slurm time on '`om_start_head.sh`' and '`om_start_worker.sh`' accordingly.
4. Install the source. This installs the scripts to `~/.lifelong`
   `source install.sh`
5. Start the cluster. This will start the ray on the head and all the workers.
   `${OM_SCRIPTS_DIR}/start_cluster.sh -l -i <head_address> -n <number of worker nodes>`
   For example, on polestar, to start 10 worker nodes:
   `${OM_SCRIPTS_DIR}/start_cluster.sh -l -i 172.16.20.230 -n 10`

   Note: Do keep an eye on timeout errors. If there is a timeout error due to unavailability of nodes on SLURM, it is recommended to stop the cluster (see point 7) and restart the cluster with a lower number of worker nodes.
   The ray dashboard can be accessed on ray port (default: 8265).
6. Now you are ready to submit jobs.
7. To stop the cluster:
   `${OM_SCRIPTS_DIR}/stop_cluster.sh`

   Note: If there are timeout issues due to non-availability of nodes (mostly due to preemption with a higher priority node), it is better to cancel the slurm jobs. Use: `squeue -u <username>` to see the '`ray_worker`' nodes pending in slurm and cancel them.

Setting up Ray:

In this method, the ray is already set up in the previous stage. You can start the job submission.

Submitting a job:

The acceleration code provides a more complicated and general job submission script. This can be accessed by: `${OM_SCRIPTS_DIR}/job_submission.py`

For the VQA project, we provide another job_submission script which is more specific to this use case and enables the progress reporting specific to VQA.

1. To submit a job, clone the VQA repository.

   Gitlab link: https://gitlab.com/tomotake/understanding_reasoning.git
   Branch name: fixstars-dev

2. Prepare the datasets and store them at:
   `understanding_reasoning/experiment_X/data_generation/datasets/`
   where x is the case number.

3. Prepare the train.json and store it at:
   `understanding_reasoning/experiment_X/results`
   You can use the `change_json_paths.py` script mentioned in the previous section to change the dataset and output path in the train.json.

4. Submit the job:
   For this methodology, we will use the script: `job_submission_w_singularity.py`

   Go to the experiment path:
   ```
   cd understanding_reasoning/experiment_X/
   ```
   Copy the script to the experiment path:
   ```
   cp -f ../scripts/job_submission_w_singularity.py ./job_submission.py
   ```
   Job submission:
   ```
   python job_submission.py --work_path <path of the experiment>
   --experiment_index <indexes> -- <command to execute>
   ```

   Here,
   <path of the experiment> : is the directory where the experiment intended to be executed is located.
   <indexes>: are the experiment indexes to be executed. This can be comma separated or given in a range or a combination of both.
   <command to execute>: is the python command to train without experiment index parameter.
   For example, if case_3 experiments have to be executed with indexes 0-35 and 45-80. The case_3 is located at:
   ```
   /om5/user/gpillai/vqa/understanding_reasoning/experiment_3
   ```

   Then the command will be:
   ```
   python job_submission.py --work_path
   /om5/user/gpillai/vqa/understanding_reasoning/experiment_3 --experiment_index
   0-35,45-80 -- python main.py --host_filesystem om --run train
   ```

5. When the job submission script is executed successfully, it opens a DASH interface on its port (default: 42126). You can ssh to access the progress information.

Barebones Setup

This is one of the simpler methods to quickly enable parallelization on custom setups with smaller (or manageable) number of worker nodes. Note that it can also be used for single head node systems, that is, 0 worker nodes.

Setting up execution environment:

We will install the below packages to have a working execution environment.

The packages to install are:

```
conda create -n acceleration python=3.8
pip install PTable --progress-bar off
pip install opencv-python --progress-bar off
pip install -U ray==1.1.0 --progress-bar off
pip install nvgpu --progress-bar off
pip install reprint --progress-bar off
pip install psutil --progress-bar off
pip install plotly dash dash-core-components dash_html_components dash_table
--progress-bar off
conda install -y -q scikit-image
conda install -y -q -c anaconda cython
conda install -y -q jupyter
conda install -y -q numba
conda install -y -q tqdm
conda install -y -q h5py
```

If not using singularity for execution, install pytorch as well. We use this method for DGX and AWS.

```
conda install -y -q pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
```

If using the singularity image, then the image already has the necessary packages required to run the experiment.

Setting up Ray:

1. Once the conda environment is ready, we can proceed to clone the vqa repository.

   Gitlab link: https://gitlab.com/tomotake/understanding_reasoning.git
   Branch name: fixstars-dev

2. Start ray on the head node:
   `(head node) bash understanding_reasoning/scripts/start_node.sh -n head`

   This should also start the ray dashboard on ray port (default: 8265). Note the head IP address returned here.

3. Start ray on worker nodes:
   `bash understanding_reasoning/scripts/start_node.sh -n worker -a <head ip address>`

4. You can (should) verify that the ray is up on all systems by checking the ray dashboard mentioned in point 2.

Submitting a job:

We provide a job_submission script to quickly enable submission of multiple experiments to be executed on the ray cluster.

1. Prepare the datasets and store them at:
   `understanding_reasoning/experiment_X/data_generation/datasets/`
   where x is the case number.

2. Prepare the train.json and store it at:

   `understanding_reasoning/experiment_X/results`

   You can use the change_json_paths.py script mentioned in the previous section to change the dataset and output path in the train.json.

3. Submit the job:

   If you are using singularity, use the script:

   `job_submission_w_singularity.py`

   If you are using your own singularity image, make the appropriate changes in the script file.

   If you are using the base environment without singularity, use the script:

   `job_submission_wo_singularity.py`

   Go to the experiment path:

   ```
   cd understanding_reasoning/experiment_X/
   ```

   Copy the script to the experiment path:

   ```
   cp -f ../scripts/job_submission_wo_singularity.py ./job_submission.py
   ```

   Job submission:

   ```
   python job_submission.py --work_path <path of the experiment>
   --experiment_index <indexes> -- <command to execute>
   ```

   Here,

   <path of the experiment> : is the directory where the experiment intended to be executed is located.

   <indexes>: are the experiment indexes to be executed. This can be comma separated or given in a range or a combination of both.

   <command to execute>: is the python command to train without experiment index parameter.

For example, if case_3 experiments have to be executed with indexes 0-35 and 45-80. The case_3 is located at:
`/om5/user/gpillai/vqa/understanding_reasoning/experiment_3`

Then the command will be:

```
python job_submission.py --work_path
/om5/user/gpillai/vqa/understanding_reasoning/experiment_3 --experiment_index
0-35,45-80 -- python main.py --host_filesystem om --run train
```

When the job submission script is executed successfully, it opens a DASH interface on its port (default: 42126). You can ssh to access the progress information.

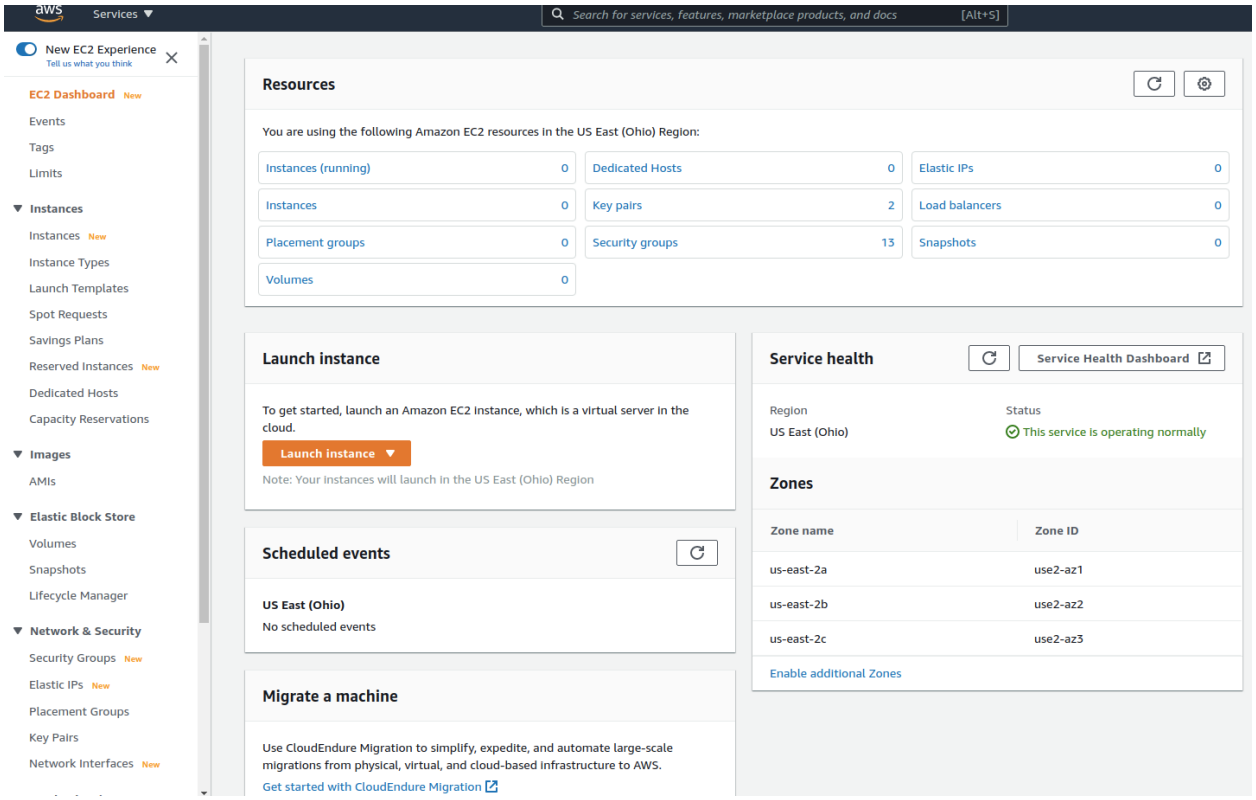
We use this barebones method without singularity on DGX and AWS.

Appendix

AWS Setup

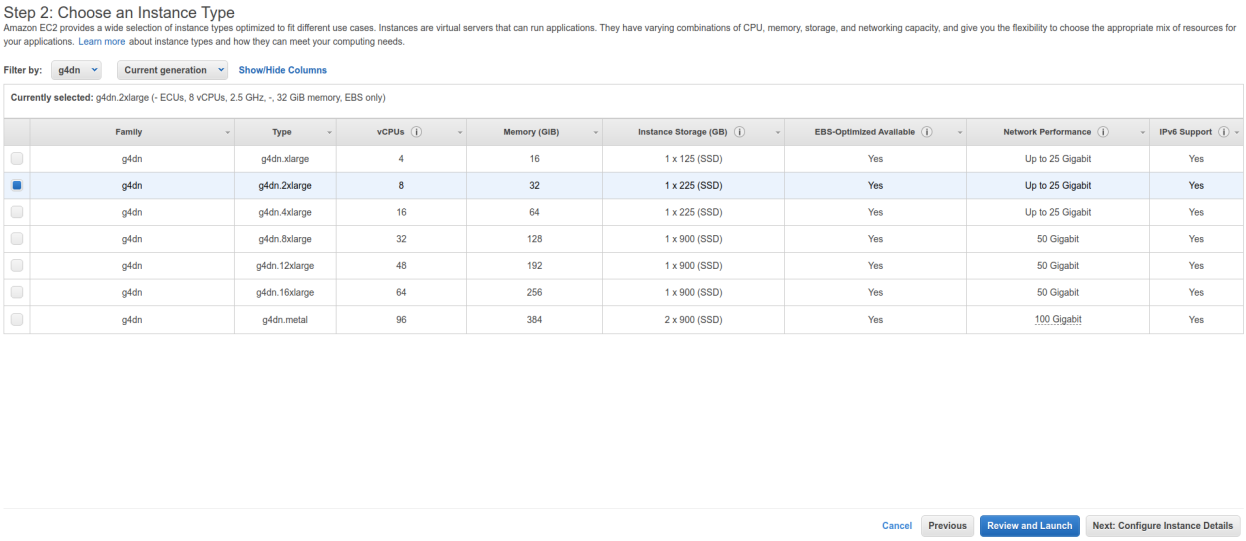
We will briefly walk through the process of creating an AWS instance.

Below is the screenshot of the EC2 Dashboard:

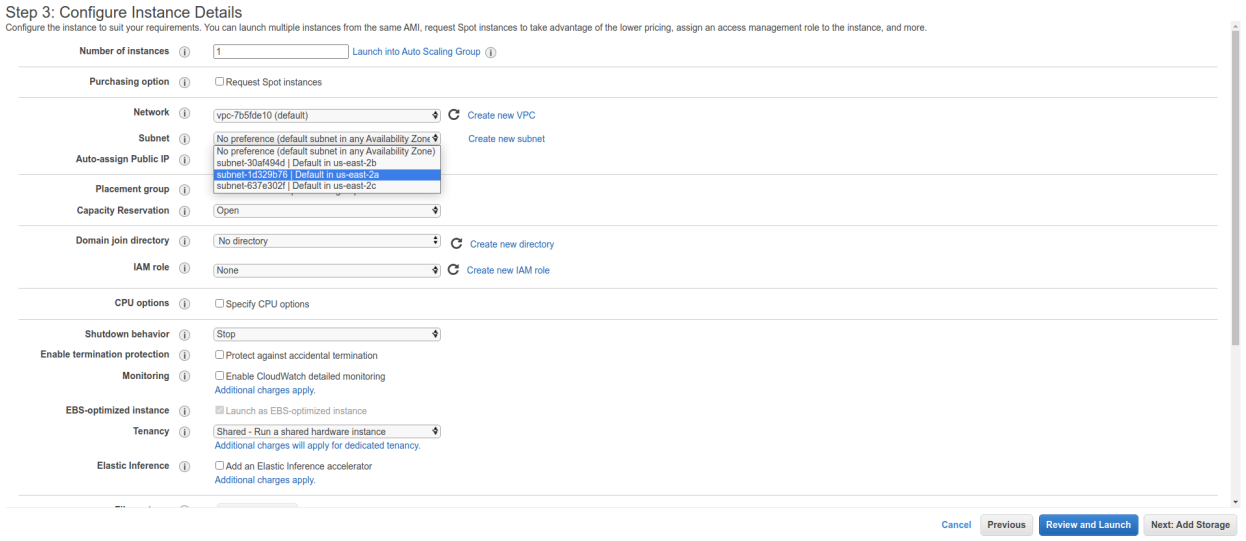


To create an instance, click on `Launch Instance`.
We will create an Ubuntu instance:

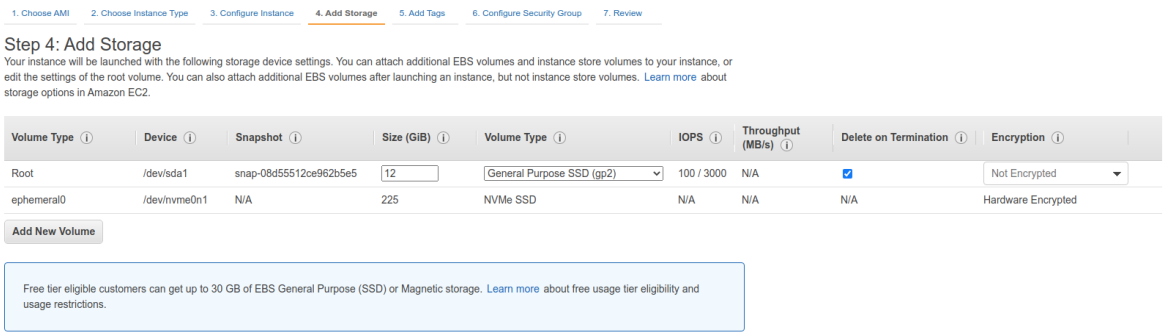
For the purpose of the VQA project, the best fitting instances are the `g4dn.2xlarge`. We select the same in Step 2



In Step 3, we configure the number of instances to be created and also it's important to have the same subnet if we need to have the possibility of using EBS Multi-Attach feature.



In Step 4: We need to make sure the root drive has at least 12 GB memory for NVIDIA driver installation.



You can now launch the instances after this. We can get a list of all instances in the EC2 dashboard with the relevant login IP information. The username would be 'ubuntu'

To set up all the base packages necessary to have a working environment, follow below steps:

It's preferable to paste below on a script and easily execute.

Script 1

```
$ sudo mkdir /mnt/nvme
$ sudo mkfs -q -t ext4 /dev/nvme1n1
$ sudo mount /dev/nvme1n1 /mnt/nvme/
$ sudo chmod -R 777 /mnt/nvme/
$ cd /mnt/nvme
$ mkdir Downloads; cd Downloads/
$ wget
https://developer.download.nvidia.com/compute/cuda/11.2.1/local_installers/cuda_11.2.1_460.32.03_linux.run
$ wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-x86_64.sh
$ sudo apt update
$ sudo apt install -y build-essential
$ sudo bash cuda_11.2.1_460.32.03_linux.run --silent --driver --toolkit
--toolkitpath=/mnt/nvme/CUDA11 --samples --samplespath=/mnt/nvme/CUDA11/Samples
--librarypath=/mnt/nvme/CUDA11 --installpath=/mnt/nvme/CUDA11 --no-man-page
$ sudo apt install -y git
$ sudo apt install -y nvtop
$ sudo apt install -y expect
```

```
$ sudo apt install -y htop
$ sudo apt install -y net-tools
$ sudo apt install -y openssh-server
$ sudo timedatectl set-timezone America/New_York
$ sudo apt install -y svtools moreutils
$ bash Anaconda3-2020.11-Linux-x86_64.sh -bf -p /mnt/nvme/anaconda3
$ echo 'export PS1="\[\033[32m\]\h: \[\033[36m\]\W \[\033[35m\]\\$ \[$(tput sgr0)\]"'
>> ~/.bashrc
$ echo "export LD_LIBRARY_PATH=/mnt/nvme/CUDA11/lib64:${LD_LIBRARY_PATH}" >>
~/.bashrc
$ echo "export PATH=/mnt/nvme/CUDA11/bin:${PATH}" >> ~/.bashrc
$ echo "export PATH=/mnt/nvme/anaconda3/bin:${PATH}" >> ~/.bashrc
$ cd ~
```

After script execution,

```
$ source ~/.bashrc
```

Script 2

```
$ conda init bash
$ conda update -y -q conda
$ conda create -n vqa -y -q python=3.8
$ echo "conda activate vqa" >> ~/.bashrc
$ echo "cd /mnt/nvme" >> ~/.bashrc
```

After script execution

```
$ source ~/.bashrc
```

Script 3

```
$ pip install PTable --progress-bar off
$ pip install opencv-python --progress-bar off
$ conda install -y -q pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
$ pip install -U ray==1.1.0 --progress-bar off
$ pip install nvgpu --progress-bar off
$ pip install reprint --progress-bar off
$ pip install psutil --progress-bar off
$ pip install plotly dash dash-core-components dash_html_components $ dash_table
--progress-bar off
$ conda install -y -q scikit-image
$ conda install -y -q -c anaconda cython
$ conda install -y -q jupyter
$ conda install -y -q numba
$ conda install -y -q tqdm
$ conda install -y -q h5py
$ pip install selenium --progress-bar off
```

You can simply follow the barebones methodology discussed in the above sections to set up ray on 1 node as head and the rest of the nodes as workers.

Satori Setup

Satori is a Power architecture high performance cluster. Currently the version of ray (0.8) available in powerai channel on conda has some network issues. So we are not able to use the job submission scripts. However, we can create a working environment and use traditional SLURM commands to run the trainings.

Below are the steps to create a working VQA environment:

```
$ cd /nobackup/users/<username>
$ wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-ppc64le.sh
$ bash Anaconda3-2020.11-Linux-ppc64le.sh -bf -p /nobackup/users/<username>/anaconda3
```

We add IBM conda channel to install pytorch 1.3 (as of Mar 24, 2021)

```
$ conda config --prepend channels
https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda/
$ conda create -n vqa python=3.6
$ conda activate vqa
$ conda install pytorch
$ conda install -c plotly plotly
$ conda install -c conda-forge ptable
$ conda install -c anaconda psutil
$ conda install -c conda-forge dash dash-core-components
$ conda install -y -q scikit-image
$ conda install -y -q -c anaconda cython
$ conda install -y -q jupyter
$ conda install -y -q numba
$ conda install -y -q tqdm
$ conda install -y -q h5py
```

While requesting SLURM nodes, we must ask for appropriate memory required using the '`--mem`' option.