Testing GPU/ML Framework Compatibility

Justin Hiemstra
HTCondor Week
May 24, 2022
Introduction

- Every ML job that runs on a GPU in CHTC requires three things to run correctly:

Source: https://docs.nvidia.com/cuda/cuda-runtime-api/index.html
Introduction

- Every ML job that runs on a GPU in CHTC requires three things to run correctly:
  a. A deep learning framework – e.g. PyTorch, TensorFlow

Source: https://docs.nvidia.com/cuda/cuda-runtime-api/index.html
Introduction

- Every ML job that runs on a GPU in CHTC requires three things to run correctly:
  a. A deep learning framework – e.g. PyTorch, TensorFlow
  b. GPU selection – in HTCondor, this is defined using a CUDA Compute capability

Source: https://docs.nvidia.com/cuda/cuda-runtime-api/index.html
Introduction

- Every ML job that runs on a GPU in CHTC requires three things to run correctly:
  a. A deep learning framework – e.g. PyTorch, TensorFlow
  b. GPU selection – in HTCondor, this is defined using a CUDA Compute capability
  c. An instance of some CUDA runtime library – this handles communication between the GPU and the computer by providing drivers.

Source: https://docs.nvidia.com/cuda/cuda-runtime-api/index.html
So why is this a difficult problem?
● Lack of documentation

Source: https://www.explainxkcd.com/wiki/index.php/2224:_Software_Updates
- Lack of documentation
- Dropped support for obsolete frameworks/compute capabilities
  - e.g. support for compute capability 2.x (Fermi architecture) dropped starting at CUDA runtime version 9.0

Source: https://www.explainxkcd.com/wiki/index.php/2224:_Software_Updates
- Lack of documentation
- Dropped support for obsolete frameworks/compute capabilities
  - e.g. support for compute capability 2.x (Fermi architecture) dropped starting at CUDA runtime version 9.0
- Non-heterogeneous server configuration

Source: https://www.explainxkcd.com/wiki/index.php/2224:_Software_Updates
Testing a single tuple of versions
Testing a single tuple of versions

Conda Environment
(Framework and CUDA toolkit)
Testing a single tuple of versions

Conda Environment
(Framework and CUDA toolkit)

HTCondor Submit File
(Compute Capability and Timeout)
Testing a single tuple of versions

- Conda Environment
  - (Framework and CUDA toolkit)
- Submit the Job
- HTCondor Submit File
  - (Compute Capability and Timeout)
Testing a single tuple of versions

Analysis:
- Did it match?
- Did the framework find and use the GPU?
- Did the environment resolve?
Scaling up

- Brute-force analysis of the entire version space poses challenges:
  - Very large search space will gobble up resources
  - Need some type of automation for collecting valid test parameters
  - How to manage all the jobs
Scaling up

Prune the version space

Dynamically generate run files

Job management

Routine testing & result aggregation

Analysis
Version space pruning

- Only certain compute capabilities may be available
  - Query the system:

    ```bash
    condor_status -compact -constraint TotalGpus > 0 -af CUDACapability
    ```
Version space pruning

- Only certain compute capabilities may be available
  - Query the system:

```bash
condor_status -compact -constraint TotalGpus > 0 -af CUDACapability
```

- Focus attention on framework and CUDA runtime versions available through Conda
  - Generate these using RegEx:

```bash
conda search tensorflow-gpu -c conda-forge | grep -E -o \' [0-9]+.[0-9]+.[0-9]+ \' | cut -d. -f 1-2 | awk '($1=$1;print)' | uniq

conda search cudatoolkit -c conda-forge | grep -E -o \' [0-9]+.[0-9]+.[0-9]+ \' | cut -d. -f 1-2 | awk '($1=$1;print)' | uniq
```
Version space pruning

- Only certain compute capabilities may be available
  - Query the system:
    ```
    condor_status -compact -constraint TotalGpus > 0 -af CUDACapability
    ```

- Focus attention on framework and CUDA runtime versions available through Conda
  - Generate these using RegEx:
    ```
    conda search tensorflow-gpu -c conda-forge | grep -E -o ' [0-9]+.[0-9]+.[0-9]+ ' | cut -d. -f 1-2 | awk '{$1=$1;print}' | uniq
    ```
    ```
    conda search cudatoolkit -c conda-forge | grep -E -o ' [0-9]+.[0-9]+.[0-9]+ ' | cut -d. -f 1-2 | awk '{$1=$1;print}' | uniq
    ```

- Consider testing only the most recent versions of each framework/CUDA runtime
Scaling up

- Prune the version space
- Dynamically generate run files
- Job management
- Analysis
- Routine testing & result aggregation
Dynamic file generation

- Each job needs several files:
  - Portable Conda installer
  - Conda environment file (ML framework version and CUDA runtime version)
  - HTCondor submit file (Compute capability and timeout)
  - Script to configure environment on execute node
Dynamic file generation

- Each job needs several files:
  - Portable Conda installer
  - Conda environment file (ML framework version and CUDA runtime version)
  - HTCondor submit file (Compute capability and timeout)
  - Script to configure environment on execute node

- For each job, use Python String format() method to build files

```python
env_yml = """"channels:
- conda-forge
- defaults
dependencies:
- tensorflow-gpu={}
-cudatoolkit={}"""
.format(tf_version,cuda_lib_version)
```
Scaling up

- Prune the version space
- Dynamically generate run files
- Job management
- Routine testing & result aggregation
- Analysis
Job management

- Once all files are generated, they need to be submitted and managed:
  - Each job as a timeout – if no match has occurred within this period, count as a failure
  - Wait until global timeout before processing output files
Job management

- Once all files are generated, they need to be submitted and managed
  - Each job as a timeout – if no match has occurred within this period, count as a failure
  - Wait until global timeout before processing output files

How can we automatically manage job submissions without leaving a script running on the submit node?
Directed Acyclic Graph Workflows (DAGs)

- HTCondor supports DAGman, a tool for DAG workflows
Directed Acyclic Graph Workflows (DAGs)

- HTCondor supports DAGman, a tool for DAG workflows
- Testing procedure using DAGman:
  - DAGman spawns a parent process
  - Parent process runs PRE Script to handle pruning and file generation
  - Parent process submits all jobs with a 24 hour timeout. Most will match but some won't
  - After 24 hours, parent process runs a POST script to interpret output of each job
Scaling up

- Prune the version space
- Dynamically generate run files
- Job management
- Analysis
- Routine testing & result aggregation
Analysis

- **Conditions for "success:"**
  - The job matched
  - The environment resolved
  - ML framework can see GPU and run a basic operation

- **For each job:**
  - Scan the output and error file
  - Look for keywords that indicate status of each condition

- **A failure in any step constitutes a total failure**
  - Record which step failed to better understand the system
### Valid Combinations

<table>
<thead>
<tr>
<th>Framework</th>
<th>Framework Version</th>
<th>Compute Capability</th>
<th>CUDA Runtime Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF</td>
<td>2.6</td>
<td>7.5</td>
<td>11.1-11.5</td>
</tr>
<tr>
<td>TF</td>
<td>2.7</td>
<td>7.5</td>
<td>11.1-11.5</td>
</tr>
<tr>
<td>TF</td>
<td>2.6</td>
<td>8.0</td>
<td>11.1-11.5</td>
</tr>
<tr>
<td>TF</td>
<td>2.7</td>
<td>8.0</td>
<td>11.1-11.5</td>
</tr>
<tr>
<td>PT</td>
<td>1.10</td>
<td>7.5</td>
<td>11.1-11.5</td>
</tr>
<tr>
<td>PT</td>
<td>1.11</td>
<td>7.5</td>
<td>11.1-11.5</td>
</tr>
<tr>
<td>PT</td>
<td>1.10</td>
<td>8.0</td>
<td>11.1-11.5</td>
</tr>
<tr>
<td>PT</td>
<td>1.11</td>
<td>8.0</td>
<td>11.1-11.5</td>
</tr>
</tbody>
</table>

### Invalid Combinations

<table>
<thead>
<tr>
<th>Framework</th>
<th>Framework Version</th>
<th>Compute Capability</th>
<th>CUDA Runtime Version</th>
<th>Failure Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF</td>
<td>2.4</td>
<td>8.0</td>
<td>11.4</td>
<td>Environment Resolution Failure</td>
</tr>
<tr>
<td>TF</td>
<td>2.6</td>
<td>8.0</td>
<td>11.6</td>
<td>Failure to Match</td>
</tr>
<tr>
<td>PT</td>
<td>1.11</td>
<td>8.0</td>
<td>11.1</td>
<td>Environment Resolution Failure</td>
</tr>
<tr>
<td>PT</td>
<td>1.11</td>
<td>8.0</td>
<td>11.6</td>
<td>Failure to Match</td>
</tr>
</tbody>
</table>
Scaling up

- Prune the version space
- Job management
- Routine testing & result aggregation
- Dynamically generate run files
- Analysis
Routine testing and result aggregation

- Run quarterly, view tables, look for anything unexpected
- Use the stored valid and invalid combinations to help during facilitation meetings, quickly diagnose reasons for failure
- Researcher could use this code and DAG to explore how these version combinations affect runtime or deep learning model performance
Questions?

GitHub Repo:
https://github.com/CHTC/gpu-compatibility-testing

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATH Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

Additional support provided by the UW-Madison GPU Lab UW2020 award funded by the Office of the Vice Chancellor for Research and Graduate Education and the Wisconsin Alumni Research Foundation.