11falsenonelisttrue

MPI Parallel Jobs

MPI jobs are scheduled largely like any other job. To facilitate communication between parallel jobs run both on the same host and on different hosts, you'll need to enable host networking and IPC modes with the `LSF_DOCKER_NETWORK` and `LSF_DOCKER_IPC` environment variables, as well as request job slots and resources sufficient to run the job. Below is a hypothetical example of how to run an mpi_hello_world MPI application with 2 processes forced to run on separate hosts:

export LSF_DOCKER_NETWORK=host > export LSF_DOCKER_IPC=host > bsub -n 2 -R 'affinity[core(1)] span[ptile=1]' -I -q general-interactive \ -a 'docker(joe.user/ubuntu16.04_openmpi_lsf:1.1)' /usr/local/mpi/bin/mpirun /usr/local/bin/mpi_hello_world Job <2042> is submitted to queue . <> <> 1.1: Pulling from joe.user/ubuntu16.04_openmpi_lsf Digest: sha256:5661e21c7f20ea1b3b14537d17078cdc288c4d615e82d96bf9f2057366a0fb8f Status: Image is up to date for joe.user/ubuntu16.04_openmpi_lsf:1.1 docker.io/joe.user/ubuntu16.04_openmpi_lsf:1.1 Hello world from processor compute1-exec-3.ris.wustl.edu, rank 0 out of 2 processors Hello world from processor compute1-exec-9.ris.wustl.edu, rank 1 out of 2 processors]]>

GPU Parallel Jobs

There are execution nodes in the cluster with GPUs. These are "resources" in the parlance of the cluster manager. Simply add the `-gpu` flag to make them visible.

We provide services for the entire WashU campus and as such, we suggest a best practices of not over allocating how many resources a single user utilizes. To that point, we suggest that a single user take up no more than 10 GPUs at a time.

Below is a hypothetical example using Tensorflow 2:

Here we save the above code as a file named tf2.py and submit to the cluster:

If you are using other options that are part of the `-R` option, you will need to list `gpuhost` first.

If you are using `select` for another option, you will need to place `gpuhost` within that and first.

```
-R select[gpuhost && port8888=1]
```

bsub -Is -q general-interactive -R 'gpuhost' -gpu "num=4:gmodel=TeslaV100_SXM2_32GB" -a 'docker(tensorflow/tensorflow:latest-gpu)' /bin/bash -c "CUDA_VISIBLE_DEVICES=0,1,2,3; python tf2.py; ... lots of output ... 2019-11-01 22:07:25.455050: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1 2019-11-01 22:07:25.607212: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties: name: Tesla V100-SXM2-32GB major: 7 minor: 0 memoryClockRate(GHz): 1.53 pciBusID: 0000:18:00.0 2019-11-01 22:07:25.608657: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 1 with properties: name: Tesla V100-SXM2-32GB major: 7 minor: 0 memoryClockRate(GHz): 1.53 pciBusID: 0000:3b:00.0 2019-11-01 22:07:25.610081: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 2 with properties: name: Tesla V100-SXM2-32GB major: 7 minor: 0 memoryClockRate(GHz): 1.53 pciBusID: 0000:86:00.0 2019-11-01 22:07:25.611511: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 3 with properties: name: Tesla V100-SXM2-32GB major: 7 minor: 0 memoryClockRate(GHz): 1.53 ...]]>

In this particular example, the Tensorflow 2 container expects the use of the CUDA_VISIBLE_DEVICES environment variable. This behavior will likely vary with different software containers.