Using IBM Spectrum LSF with NVIDIA DGX systems

This guide assumes that you are using IBM Spectrum LSF 10.1 Service Pack 8 and are familiar with LSF administration. If you are using an earlier version, additional configuration steps are required.

For more information on IBM Spectrum LSF GPU support and GPU best practices, refer to the following links: IBM Knowledge Center, IBM DeveloperWorks, or IBM Spectrum LSF User Group.

Note that the output of the some of the examples are simplified for clarity in this document. Format the output of the query command by using the "-o *fieldlist*" and "-json" options to present just the information that you are interested in as plain text or JSON formatted.

While this guide is intended for NVIDIA DGX systems, the same functionality is available on all x86 and Power systems supporting NVIDIA GPUs with the same software stack.

Configuration

IBM Spectrum LSF (LSF) automatically detects and configures NVIDIA GPU support. This means that you can take advantage of GPUs as soon as LSF is installed. GPU related resources are automatically created and treated as built-in resources.

If you upgraded from an earlier version, you must define LSF_GPU_AUTOCONFIG=Y in the lsf.conf file.

Optional configuration items

While the majority of GPU configuration is automatic, there are some additional optional features that you can enable:

- Access to GPU's can be strictly enforced by defining LSF_RESOURCE_ENFORCE="gpu" in the lsf.conf file. This will place the allocated GPUs in the same cgroup as the job, ensuring that the job can only use the GPUs that are allocated to it. It also ensures that jobs that do not request GPUs will not be able to access them.

- If you are using the NVIDIA Data Center GPU Manager (DCGM), define LSF_DCGM_PORT=5555 in the lsf.conf file to enable LSF to communicate with DCGM. If you do not enable DCGM support, the number of GPU health metrics collected is considerably reduced. When enabled, LSF periodically evaluates the health of the GPU. You may also define alarms and actions in LSF RTM that are triggered on certain events (such as an ECC error).

- If your GPUs are used infrequently, you can obtain significant power savings by powering down the GPUs when not in use. To enable this feature, define LSB_GPU_POWEROFF_DURATION=*time* in the lsf.conf file where *time* is the number of seconds that the GPU is idle before forcing it to the lowest power state.

- By default, LSF's fairshare scheduling algorithms are based on the CPU consumption and elapsed wall clock time. If your applications are heavily based on GPUs, this is no longer "fair" because the algorithms do not account for GPU time. To account for GPU time, you can enable current and historical GPU time in the fairshare calculations (ENABLE_GPU_HIST_RUN_TIME & GPU_RUN_TIME_FACTOR parameters in the lsb.params file).

- LSF frequently uses pre-emption to allow a higher priority job to start by suspending a lower priority job. However, GPUs do not currently support the concept of suspension. This means that if you enable pre-emption for GPU workloads, they are killed and requeued rather than suspended.

Displaying GPU Information

LSF collects a wealth of information about the system and its GPUs. View the static GPU configuration configuration with the “lshosts -gpu” command:

```
> lshosts -gpu
HOST_NAME     gpu_id  gpu_model         gpu_driver    gpu_factor     numa_id
ibm-dgx01          0  TeslaV100_SXM2_   418.67        7.0            0
                   1  TeslaV100_SXM2_   418.67        7.0            0
                   2  TeslaV100_SXM2_   418.67        7.0            0
                   3  TeslaV100_SXM2_   418.67        7.0            0
                   4  TeslaV100_SXM2_   418.67        7.0            1
                   5  TeslaV100_SXM2_   418.67        7.0            1
                   6  TeslaV100_SXM2_   418.67        7.0            1
                   7  TeslaV100_SXM2_   418.67        7.0            1
```

Obtain a summary of the load and health of the GPUs with the “lsload -gpu” command, and obtain detailed health information about each GPU with the “lsload -gpuload” command. If DCGM is not enabled, no information is displayed for some of these fields.

```
> lsload -gpu
HOST_NAME status  ngpus gpu_shared_avg_ut ngpus_shared ngpus_excl_t ngpus_excl_p ngpus_prohibited ngpus_physical
ibm-dgx01     ok      8              30%            4            2            2                0             8

> lsload -gpuload
HOST_NAME gpuid gpu_model   mode temp ecc gpu_ut gpu_mut gpu_mtotal gpu_mused gpu_pstate gpu_status gpu_error
ibm-dgx01     0 TeslaV100_S  0.0  48C 0.0   100%      7%      15.7G      629M          0          ok         -
              1 TeslaV100_S  0.0  38C 0.0     0%      0%      15.7G       11M          0          ok         -
              2 TeslaV100_S  3.0  41C 0.0     0%      0%      15.7G        0M          0          ok         -
              3 TeslaV100_S  3.0  38C 0.0     0%      0%      15.7G        0M          0          ok         -
              4 TeslaV100_S  3.0  33C 0.0     0%      0%      15.7G        0M          0          ok         -
              5 TeslaV100_S  3.0  35C 0.0     0%      0%      15.7G        0M          0          ok         -
              6 TeslaV100_S  3.0  35C 0.0     0%      0%      15.7G        0M          0          ok         -
              7 TeslaV100_S  0.0  32C 0.0     0%      0%      15.7G        0M          0          ok         -
```

Note that the mode, temp and ecc columns have had “gpu_” removed from the header to aid formatting on the page.

Submitting GPU jobs

LSF has supported NVIDIA GPU workloads for over 10 years. In that time, GPU capabilities have evolved, and so has the command syntax used to submit workloads. These examples use the syntax in LSF 10.1 Service Pack 8 (LSF_GPU_NEW_SYNTAX=extend in lsf.conf).

- **bsub -gpu** { - | "[num=num_gpus] [:mode= {shared | exclusive_process}] [:mps= {yes | no | per_socket | per_gpu}] [:j_exclusive={yes | no}] [:gmodel=model_name[-mem_size]] [:gtile={tile_num|'!'}] [:gmem=mem_value] [:nvlink=yes][:aff=yes | no]"}

Examples

1: Request a single GPU

```
> bsub -gpu - ./gpu_app
```

2: Request 2 GPUs with EXCLUSIVE_PROCESS mode. LSF will switch an available GPU to the requested mode

```
> bsub -gpu "num=2:mode=exclusive_process" ./gpu_app
```

3: Request 4 GPUs with NVLink connections

```
> bsub -gpu "num=4:nvlink=yes" ./gpu_app
```

4: Request 2 TeslaV100 GPUs with NVLink connections between the allocated GPUs

```
> bsub -gpu "num=2:nvlink=yes: gmodel=TeslaV100" ./gpu_app
```

5: Request 4 GPUs with 2 GPUs on each socket

```
> bsub -gpu "num=4:gtile=2" ./gpu_app
```

6: Guarantee CPU-GPU affinity for the job

```
> bsub -n2 -R "affinity[core(2)]" -gpu "num=2" ./gpu_app
```

7: Reserve 100MB GPU memory for each allocated GPU

```
> bsub -gpu "num=2:gmem=100M" ./gpu_app
```

8: Request 4 GPUs and start an MPS instance for each GPU

```
> bsub -gpu "num=4:mps=per_gpu" ./gpu_app
```

9: Request 1 GPU on a shared GPU. If more than one GPU is in shared mode, LSF remaps CUDA_VISIBILE_DEVICES to ensure that not all jobs use the same GPU0.

```
> bsub -gpu "num=1:mode=shared" ./gpu_app
```

The “bsub -gpu” syntax provides a simple but powerful interface for submitting typical workloads. However, there are workloads that may have more complex requirements:

- Request 1 host with 2 V100 GPUs with NVLink connections or 2 hosts with 4 K80 GPU on each host
- Request 2 V100 or 4 K80 GPUs on the same socket
- Request 8 cores each with a GPU, plus to 2 additional cores with no gpu’s allocated.

The simple “-gpu” syntax cannot handle these requirements However, you can also specify GPU requirements with LSF’s existing “-R resource_requirements” syntax, which supports both alternate and compound resources. Refer to the LSF documentation for a more detailed description of this functionality.

Viewing Job Information

To view how GPUs are being used by jobs, add the “-gpu” option to the bjobs command:

```
> bsub -gpu "num=1:mode=exclusive_process" ./gpu_burn 120
Job <540> is submitted to default queue <normal>.

> bjobs -l -gpu 540
Job <540>, User <test>, Project <default>, Status <DONE>, Queue <normal>, Com
                     mand <./gpu_burn 120>, Share group charged </test>
……
Mon Jul 15 11:16:08: Done successfully. The CPU time used is 153.0 seconds.
                     HOST: ibm-dgx01; CPU_TIME: 153 seconds
                        GPU ID: 0
                            Total Execution Time: 122 seconds
                            Energy Consumed: 25733 Joules
                            SM Utilization (%): Avg 99, Max 100, Min 64
                            Memory Utilization (%): Avg 28, Max 39, Min 9
                            Max GPU Memory Used: 30714888192 bytes
                         GPU Energy Consumed: 25733.000000 Joules
 MEMORY USAGE:
 MAX MEM: 219 Mbytes;  AVG MEM: 208 Mbytes
```

For jobs that are using CPU affinity or NVLink affinity, use the “-aff” option to see the details:

```
> bsub -n2 -R "affinity[core(2)]" -gpu "num=2" sleep 1h
Job <559> is submitted to default queue <normal>.

> bjobs -l -gpu –aff 559
Job <549>, User <test>, Project <default>, Status <DONE>, Queue <normal>, Com
                     mand <sleep 1h>, Share group charged </test>
……

RESOURCE REQUIREMENT DETAILS:
 Combined: select[(ngpus>0) && (type == local)] order[gpu_maxfactor] rusage[ngp
                     us_physical=2.00] affinity[core(2)*1]
 Effective: select[((ngpus>0)) && (type == local)] order[gpu_maxfactor] rusage[
                     ngpus_physical=2.00] affinity[core(2)*1]
 AFFINITY:
                     CPU BINDING                             MEMORY BINDING
                     ------------------------                ------------------
 HOST                TYPE   LEVEL  EXCL   IDS                POL   NUMA SIZE
 ibm-dgx01           core   -      -      /0/0/0             -     -    -
                                          /0/0/1
 ibm-dgx01           core   -      -      /0/0/2             -     -    -
                                          /0/0/3

 GPU REQUIREMENT DETAILS:
 Combined: num=2:mode=shared:mps=no:j_exclusive=no
 Effective: num=2:mode=shared:mps=no:j_exclusive=no
 GPU_ALLOCATION:
 HOST          TASK   ID   MODEL          MTOTAL  FACTOR MRSV    SOCKET NVLINK
 ibm-dgx01        0    0   TeslaV100_S     15.7G     7.0   0M         0    -/N
                  1    1   TeslaV100_S     15.7G     7.0   0M         0    N/-
```

Host based view of how GPUs are being used

Use the bhosts command to have a host-based, rather than job-based, view of how GPUs are allocated.

A simple summary is available with the “bhosts -gpu” command:

```
> bhosts -gpu ibm-dgx01
HOST_NAME               ID           MODEL      MUSED       MRSV  NJOBS     RUN    SUSP     RSV
ibm-dgx01                0      TeslaV100_SXM2_    0M         0M      0       0       0       0
                         1      TeslaV100_SXM2_    0M         0M      0       0       0       0
                         2      TeslaV100_SXM2_    0M         0M      0       0       0       0
                         3      TeslaV100_SXM2_    0M         0M      0       0       0       0
                         4      TeslaV100_SXM2_    0M         0M      0       0       0       0
                         5      TeslaV100_SXM2_    0M         0M      0       0       0       0
                         6      TeslaV100_SXM2_    0M         0M      0       0       0       0
                         7      TeslaV100_SXM2_    0M         0M      0       0       0       0
```

And a detailed view is available with the “bhosts -l -gpu” command:

```
> bhosts -l -gpu ibm-dgx01
HOST: ibm-dgx01
NGPUS NGPUS_SHARED_AVAIL NGPUS_EXCLUSIVE_AVAIL
    8                  8                    8

STATIC ATTRIBUTES
ID     MODEL               MTOTAL         FACTOR   SOCKET NVLINK
0      TeslaV100_SXM2_16GB  15.7G            7.0        0  -/Y/Y/Y/Y/N/N/N
1      TeslaV100_SXM2_16GB  15.7G            7.0        0  Y/-/Y/Y/N/Y/N/N
2      TeslaV100_SXM2_16GB  15.7G            7.0        0  Y/Y/-/Y/N/N/Y/N
3      TeslaV100_SXM2_16GB  15.7G            7.0        0  Y/Y/Y/-/N/N/N/Y
4      TeslaV100_SXM2_16GB  15.7G            7.0        1  Y/N/N/N/-/Y/Y/Y
5      TeslaV100_SXM2_16GB  15.7G            7.0        1  N/Y/N/N/Y/-/Y/Y
6      TeslaV100_SXM2_16GB  15.7G            7.0        1  N/N/Y/N/Y/Y/-/Y
7      TeslaV100_SXM2_16GB  15.7G            7.0        1  N/N/N/Y/Y/Y/Y/-

DYNAMIC ATTRIBUTES
ID     MODE                MUSED      MRSV       TEMP    ECC     UT      MUT     PSTATE STATUS   ERROR
0      EXCLUSIVE_PROCESS     0M       0M          48C     0      100%     7%          0 ok       -
1      EXCLUSIVE_PROCESS     0M       0M          39C     0        0%     0%          0 ok       -
2      EXCLUSIVE_PROCESS     0M       0M          50C     0      100%     0%          0 ok       -
3      EXCLUSIVE_PROCESS   320M       0M          46C     0      100%    25%          0 ok       -
4      EXCLUSIVE_PROCESS   320M       0M          47C     0      100%    25%          0 ok       -
5      EXCLUSIVE_PROCESS     0M       0M          36C     0        0%     0%          0 ok       -
6      EXCLUSIVE_PROCESS   320M       0M          48C     0      100%    25%          0 ok       -
7      SHARED                0M       0M          32C     0        0%     0%          0 ok       -

GPU JOB INFORMATION
ID     JEXCL  RUNJOBIDS          SUSPJOBIDS         RSVJOBIDS
0      Y      550                 -                  -
1      -      -                   -                  -
2      -      -                   -                  -
3      -      -                   -                  -
4      -      -                   -                  -
5      Y      550                 -                  -
6      -      -                   -                  -
7      -      -                   -                  -
```

Docker

LSF provides integrated support for Docker and the NVIDIA Container Runtime for Docker.

LSF takes responsibility for all the privileged operations that need to occur during the container start up time. This means the user **does not** need to be in the Docker user group, which removes the risk of the user ever gaining root access in the container and the host.

LSF's integration with Docker leverages LSF's Application Profile functionality. This allows the Administrator to control who can start a container, where the container is coming from, and how the container is started. For example:

```
Begin Application
NAME = tensorflow
CONTAINER=nvidia-docker[image(nvcr.io/nvidia/tensorflow:19.06-py3) options(--rm --net=host --ipc=host)]
EXEC_DRIVER=context[user(root)] \
   starter[/opt/ibm/lsfsuite/lsf/10.1/linux3.10-glibc2.17-x86_64/etc/docker-starter.py] \
controller[/opt/ibm/lsfsuite/lsf/10.1/linux3.10-glibc2.17-x86_64/etc/docker-control.py] \
   monitor[/opt/ibm/lsfsuite/lsf/10.1/linux3.10-glibc2.17-x86_64/etc/docker-monitor.py]
DESCRIPTION = NVIDIA Deep Learning Tensorflow
End Application
```

For more information on the `nvcr.io/nvidia/tensorflow:19.06-py3` container, refer to the following link: NVIDIA Deep Learning Frameworks.

An example Tensorflow Benchmark job

To view an example Tensorflow Benchmark requesting 4 GPUs in interactive mode and run inside a NVIDIA Deep Learning Tensorflow docker container:

```
> bsub -gpu "num=4:mode=exclusive_process" -app tensorflow -I python tf_cnn_benchmarks.py --num_gpus=4 --
model resnet50 --batch_size 128 --num_batches=100
Job <560> is submitted to default queue <interactive>.
<<Waiting for dispatch ...>>
<<Starting on ibm-dgx01>>

================
== TensorFlow ==
================

NVIDIA Release 19.06 (build 6800111)
TensorFlow Version 1.13.1

Container image Copyright (c) 2019, NVIDIA CORPORATION.  All rights reserved.
Copyright 2017-2019 The TensorFlow Authors.  All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying project or file.

…
Running warm up
Done warm up
Step    Img/sec total_loss
1       images/sec: ****.** +/- 0.0 (jitter = 0.0) *****
10      images/sec: ****.** +/- 0.1 (jitter = 0.3) *****
20      images/sec: ****.** +/- 0.1 (jitter = 0.3) *****
30      images/sec: ****.** +/- 0.0 (jitter = 0.2) *****
40      images/sec: ****.** +/- 0.0 (jitter = 0.3) *****
50      images/sec: ****.** +/- 0.0 (jitter = 0.3) *****
60      images/sec: ****.** +/- 0.0 (jitter = 0.3) *****
70      images/sec: ****.** +/- 0.0 (jitter = 0.3) *****
80      images/sec: ****.** +/- 0.0 (jitter = 0.3) *****
90      images/sec: ****.** +/- 0.0 (jitter = 0.3) *****
100     images/sec: ****.** +/- 0.0 (jitter = 0.3) *****
----------------------------------------------------------------
total images/sec: ****.**
----------------------------------------------------------------
```

For more information on Tensorflow Benchmark, refer to the following link: High performance benchmarks (tf_cnn_benchmarks)..

Copyright and Trademark Information