Using IBM Spectrum LSF with NVIDIA DGX systems

Version 1.2
June 2022

Table of Contents

INTRODUCTION 3
CONFIGURATION 3
OPTIONAL CONFIGURATION ITEMS 3
DISPLAYING GPU INFORMATION 4
SUBMITTING GPU JOBS 5
VIEWING JOB INFORMATION 6
HOST BASED VIEW OF GPU ALLOCATIONS 7
MULTI-INSTANCE GPU (MIG) 8
STATIC MIG CONFIGURATION 8
DYNAMIC MIG CONFIGURATION 11
CONTAINERIZED WORKLOADS 14
EXAMPLE TENSORFLOW BENCHMARK JOBS 14

Introduction

This guide assumes that you are using IBM Spectrum LSF 10.1 Fix Pack 11 (or above), and have familiarity with LSF administration. Earlier versions of LSF may not support all of the capabilities outlined in this document.

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

Note that the output of some of the LSF commands are simplified for clarity in this document. To format the output of LSF query commands, the `-o fieldlist` and `-json` options may be used to present just the information that you are interested in as plain text or JSON formatted.

While this guide is intended for NVIDIA DGX, NVIDIA DGX A100 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 LSF resources.

This is enabled with the parameter `LSF_GPU_AUTOCONFIG=Y` in the `$LSF_ENVDIR/lsf.conf` file. Note that this parameter is set by default in LSF 10.1 Fix Pack 6 and above.

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=<port>` (default `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 IBM Spectrum 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, the fairshare scheduling algorithms of LSF 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 with the parameters `ENABLE_GPU_HIST_RUN_TIME=Y` and `GPU_RUN_TIME_FACTOR=Y` (`lsb.params` or `lsb.queues`).

- 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, these lower priority jobs could be terminated 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 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 first introduced in LSF 10.1 Fix Pack 8 (`LSF_GPU_NEW_SYNTAX=extend` in `lsf.conf`).

```
bsub -gpu { - | "[num=num_gpus] [:mig={gpuinstance}/{computeinstance}] [: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

- Request a single GPU
  - `bsub -gpu - ./gpu_app`
- 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`
- Request 4 GPUs with NVLink connections
  - `bsub -gpu "num=4:nvlink=yes" ./gpu_app`
- Request 2 TeslaV100 GPUs with NVLink connections between the allocated GPUs
  - `bsub -gpu "num=2:nvlink=yes: gmodel=TeslaV100" ./gpu_app`
- Request 4 GPUs with 2 GPUs on each socket
  - `bsub -gpu "num=4:gtile=2" ./gpu_app`
- Guarantee CPU-GPU affinity for the job
  - `bsub -n2 -R "affinity[core(2)]" -gpu "num=2" ./gpu_app`
- Reserve 100MB GPU memory for each allocated GPU
  - `bsub -gpu "num=2:gmem=100M" ./gpu_app`
- Request 4 GPUs and start an MPS instance for each GPU
  - `bsub -gpu "num=4:mps=per_gpu" ./gpu_app`
- 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 the LSF "`-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 GPU allocations

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      -      -                  -                  -
```

Multi-Instance GPU (MIG)

The Nvidia Multi-Instance GPU capability allows the Nvidia A100 (Ampere-family) GPUs to be subdivided into up to 7 GPU instances, each a fully functional GPU. LSF supports scheduling to statically defined MIG instances and can be configured to dynamically configure MIG instances based upon workload demand. By right sizing the GPU partitions, LSF can drive better utilization and throughput of GPU workloads. The Nvidia A100 can be subdivided as illustrated in Figure 1.

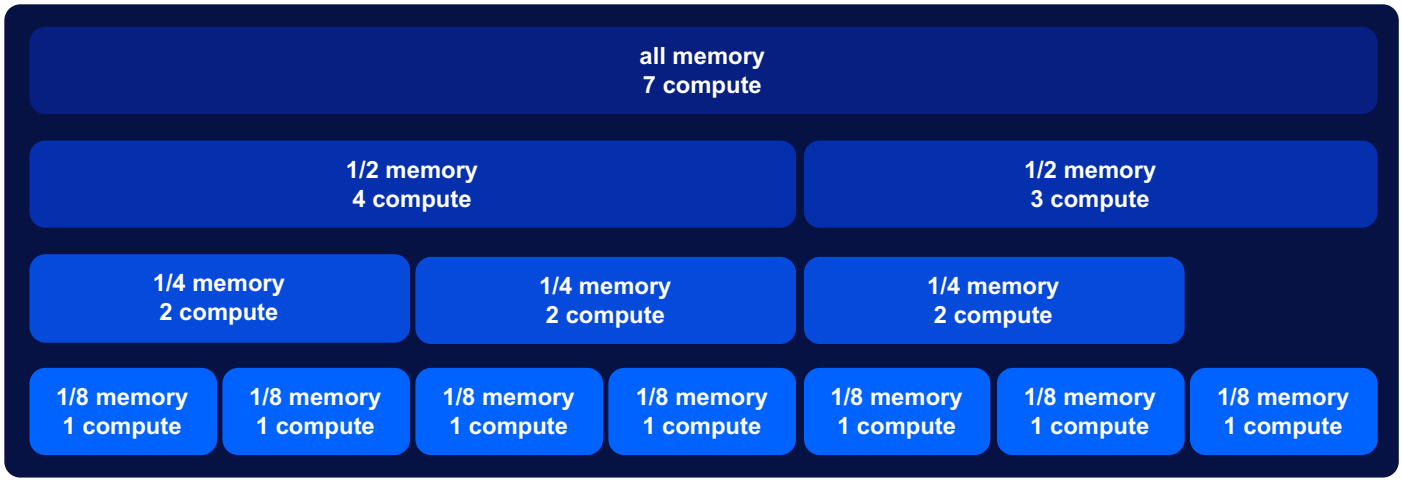

Figure 1 How the Nvidia A100 can be subdivided is fixed.

Static MIG configuration

The following example on NVIDIA DGX A100 considers the case where the administrator has statically configured MIG instances. LSF will schedule jobs to the predefined MIG instances.

From the `nvidia-smi` output, we can see that the administrator has manually configured 2 MIG instances on GPU0, one each on GPU's 1, 2, 3 and 4.

```
$ nvidia-smi

+-------------------------------------------------------------------------------+
| MIG devices:                                                                  |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0   11   0   0  |      3MiB /  4864MiB | 14      0 |  1   0    0    0    0 |
+------------------+----------------------+-----------+-----------------------+
|  0   13   0   1  |      3MiB /  4864MiB | 14      0 |  1   0    0    0    0 |
+------------------+----------------------+-----------+-----------------------+
|  1    3   0   0  |      7MiB /  9984MiB | 28      0 |  2   0    1    0    0 |
+------------------+----------------------+-----------+-----------------------+
|  2    2   0   0  |     11MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+
|  3    1   0   0  |     14MiB / 20096MiB | 56      0 |  4   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+
```

Using the LSF `lshosts` command we can see the hardware configuration, and the "`-gpu`" flag shows the details of the GPU's that have been detected:

```
$ lshosts -gpu
HOST_NAME   gpu_id       gpu_model   gpu_driver   gpu_factor      numa_id       vendor          mig
Dgxa      8      0 TeslaA100_SXM4_    450.51.06          8.0            3       Nvidia            Y
                 1 TeslaA100_SXM4_    450.51.06          8.0            3       Nvidia            Y
                 2 TeslaA100_SXM4_    450.51.06          8.0            1       Nvidia            Y
                 3 TeslaA100_SXM4_    450.51.06          8.0            1       Nvidia            Y
                 4 TeslaA100_SXM4_    450.51.06          8.0            7       Nvidia            Y
                 5 TeslaA100_SXM4_    450.51.06          8.0            7       Nvidia            Y
                 6 TeslaA100_SXM4_    450.51.06          8.0            5       Nvidia            Y
                 7 TeslaA100_SXM4_    450.51.06          8.0            5       Nvidia            Y
```

As expected, there are 8 physical A100 GPUs in the system, and MIG has been enabled for GPUs 0-4. The MIG configuration can be viewed with the "`-mig`" flag for `lshosts`.

```
$ lshosts -gpu -mig
HOST_NAME gpu_id  gpu_model        gpu_driver  gpu_factor  numa_id  vendor  devid  gid  cid  inst_name
dgxa      0       TeslaA100_SXM4_  450.51.06   8.0         3        Nvidia  0      11   0    1g.5gb
          0       TeslaA100_SXM4_  450.51.06   8.0         3        Nvidia  0      13   0    1g.5gb
          1       TeslaA100_SXM4_  450.51.06   8.0         3        Nvidia  0      3    0    2g.10gb
          2       TeslaA100_SXM4_  450.51.06   8.0         1        Nvidia  0      2    0    3g.20gb
          3       TeslaA100_SXM4_  450.51.06   8.0         1        Nvidia  0      1    0    4g.20gb
          4       TeslaA100_SXM4_  450.51.06   8.0         7        Nvidia  0      0    0    7g.40gb
          5       TeslaA100_SXM4_  450.51.06   8.0         7        Nvidia  -      -    -    -
          6       TeslaA100_SXM4_  450.51.06   8.0         5        Nvidia  -      -    -    -
          7       TeslaA100_SXM4_  450.51.06   8.0         5        Nvidia  -      -    -    -
```

This matches the `nvidia-smi` output, and we can also clearly see that GPU's 5, 6 and 7 have no MIG instances defined.

We can submit jobs and request the amount of GPU memory required, or the specific MIG configuration/slices desired. First, let's submit a job requesting a single MIG slice. This is done using the `bsub` "`-gpu`" flag with options `num={number of GPUs}:mig={gpuinstance}/{computeinstance}`.

```
$ bsub -gpu "num=1:mig=1/1" ./e06-gpu
Job <416> is submitted to default queue <normal>.

$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
416     rladmin RUN   normal     dgxa        dgxa        e06-gpu    Nov 25 06:19
```

We can view the job detail using the bjobs command with the "-gpu" and "-l" options to see what GPU instances been allocated.

```
$ bjobs -l -gpu 416
Job <416>, User <rladmin>, Project <default>, Status <RUN>, Queue <normal>, Command <./e06-gpu>, Share group
charged </rladmin>
Wed Nov 25 06:19:01: Submitted from host <dgxa-c18-u19-enp226s0>, CWD <$HOME>
                     Requested GPU <num=1:mig=1/1>;
Wed Nov 25 06:19:01: Started 1 Task(s) on Host(s) <dgxa>, Allocated 1 Slot(s)
                     on Host(s) <dgxa>,
                     Execution Home </home/rladmin>,
                     Execution CWD </home/rladmin>;

SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
loadSched   -     -     -     -       -     -    -     -     -      -      -
loadStop    -     -     -     -       -     -    -     -     -      -      -
RESOURCE REQUIREMENT DETAILS:
Combined: select[(ngpus>0) && (type == local)] order[r15s:pg] rusage[ngpus_physical=1.00:mig=1/1]
Effective: select[((ngpus>0)) && (type == local)] order[r15s:pg] rusage[ngpus_physical=1.00:mig=1/1]

GPU REQUIREMENT DETAILS:
Combined: num=1:mode=shared:mps=no:j_exclusive=yes:gvendor=nvidia:mig=1/1
Effective: num=1:mode=shared:mps=no:j_exclusive=yes:gvendor=nvidia:mig=1/1

GPU_ALLOCATION:
HOST TASK GPU_ID  GI_ID/SIZE    CI_ID/SIZE    MODEL        MTOTAL  FACTOR MRSV    SOCKET NVLINK/XGMI
dgxa 0    0       4/1           4/1           TeslaA100_SX 39.5G   8.0    0M      3      -
```

Additionally, the LSF `bhosts` command can be used to get detailed information about how jobs have been allocated to GPUs.

```
$ bhosts -gpu -l
HOST: dgxa
NGPUS NGPUS_SHARED_AVAIL NGPUS_EXCLUSIVE_AVAIL
8     8                  8

STATIC ATTRIBUTES
GPU_ID MODEL                MTOTAL    FACTOR   SOCKET VENDOR   MIG    NVLINK/XGMI
0      TeslaA100_SXM4_40GB  39.5G     8.0      3      Nvidia   Y      -/N/N/N/N/N/N/N
1      TeslaA100_SXM4_40GB  39.5G     8.0      3      Nvidia   Y      N/-/N/N/N/N/N/N
2      TeslaA100_SXM4_40GB  39.5G     8.0      1      Nvidia   Y      N/N/-/N/N/N/N/N
3      TeslaA100_SXM4_40GB  39.5G     8.0      1      Nvidia   Y      N/N/N/-/N/N/N/N
4      TeslaA100_SXM4_40GB  39.5G     8.0      7      Nvidia   Y      N/N/N/N/-/N/N/N
5      TeslaA100_SXM4_40GB  39.5G     8.0      7      Nvidia   Y      N/N/N/N/N/-/N/N
6      TeslaA100_SXM4_40GB  39.5G     8.0      5      Nvidia   Y      N/N/N/N/N/N/-/N
7      TeslaA100_SXM4_40GB  39.5G     8.0      5      Nvidia   Y      N/N/N/N/N/N/N/-

DYNAMIC ATTRIBUTES
GPU_ID MODE               MUSED     MRSV      TEMP   ECC    UT     MUT    PSTATE STATUS   ERROR
0      SHARED             87M       0M        31C    0      0%      0%   0      ok       -
1      SHARED             7M        0M        27C    0      0%      0%   0      ok       -
2      SHARED             11M       0M        28C    0      0%      0%   0      ok       -
3      SHARED             14M       0M        28C    0      0%      0%   0      ok       -
4      SHARED             0M        0M        31C    0      0%      0%   0      ok       -
5      SHARED             0M        0M        30C    0      0%      0%   0      ok       -
6      SHARED             0M        0M        30C    0      0%      0%   0      ok       -
7      SHARED             0M        0M        30C    0      0%      0%   0      ok       -

GPU JOB INFORMATION
GPU_ID JEXCL  RUNJOBIDS          SUSPJOBIDS         RSVJOBIDS          GI_ID/SIZE   CI_ID/SIZE
0      Y      416                -                  -                  4/1          4/1
1      -      -                  -                  -                  -            -
2      -      -                  -                  -                  -            -
3      -      -                  -                  -                  -            -
4      -      -                  -                  -                  -            -
5      -      -                  -                  -                  -            -
6      -      -                  -                  -                  -            -
7      -      -                  -                  -                  -            -
```

Now let's submit two more jobs, specifying the desired GPU memory rather than slices:

```
$ bsub -gpu "num=1:gmem=30G" ./e06-gpu
Job <417> is submitted to default queue <normal>.

$ bsub -gpu "num=1:gmem=30G" ./e06-gpu
Job <418> is submitted to default queue <normal>.

$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
417     rladmin RUN   normal     dgxa        dgxa        e06-gpu    Nov 25 06:25
418     rladmin PEND  normal     dgxa                    e06-gpu    Nov 25 06:25
```

Job 417 is dispatched, but job 418 cannot be dispatched due to insufficient GPU resources.

```
$ bjobs -l 418
Job <418>, User <rladmin>, Project <default>, Status <PEND>, Queue <normal>, Command <./e06-gpu>
Wed Nov 25 06:25:15: Submitted from host <dgxa>, CWD <$HOME>, Requested GPU <num=1:gmem=30000.00>;

 PENDING REASONS:
 Host's available GPU resources cannot meet the job's requirements: dgxa;
```

Explanation Job 418 is in a pending state because there are no other pre-defined MIG instances that can accommodate a 30GB job. Job 418 will remain pending until 417 is complete or if the administrator reconfigures the MIG instances.

Dynamic MIG configuration

LSF supports the ability to dynamically reconfigure MIG based on the workload requirements. This capability allows the GPU configuration to be automatically right sized to match the requirements of the workload, delivering a greater GPU ROI.

To enable the dynamic management of MIG instances in LSF, the `LSF_MANAGE_MIG=Y` parameter must be set in `lsf.conf` and a reconfiguration of LSF is required.

Note This change should be made when no applications are running on the GPUs.

The following example is on a DGX A100 system. Assuming `LSF_MANAGE_MIG=Y` has been configured, consider the following workload that has been submitted to the DGX A100 (equipped with 40GB GPUs) to an LSF queue configured as first-come first-served (FCFS) scheduling:

1. 8 jobs each requiring 32GB GPU memory (most of GPU memory)
2. 8 jobs each requiring 16GB GPU memory
3. 8 jobs each requiring 8GB GPU memory
4. 56 jobs each requiring 4GB
5. 8 jobs each requiring 36GB GPU memory (most of GPU memory)

```
$ repeat  8 bsub -gpu "num=1:gmem=32G" -J "large32"  ./e06-gpu
$ repeat  8 bsub -gpu "num=1:gmem=16G" -J "medium16" ./e06-gpu
$ repeat  8 bsub -gpu "num=1:gmem=8G"  -J "medium8"  ./e06-gpu
$ repeat 56 bsub -gpu "num=1:gmem=4G"  -J "small4"   ./e06-gpu
$ repeat  8 bsub -gpu "num=1:gmem=36G" -J "large36"  ./e06-gpu
```

- The first set of jobs, all require 32GB of GPU memory. LSF reconfigures all 8 GPUs in the system to have a single MIG partition each to run the jobs (see Figure 2). We see in the output of the `bjobs` command, the 8 "large32" jobs are running. Furthermore, the `bhosts` output shows a single job per GPU as the jobs are using the full memory size of the GPU.

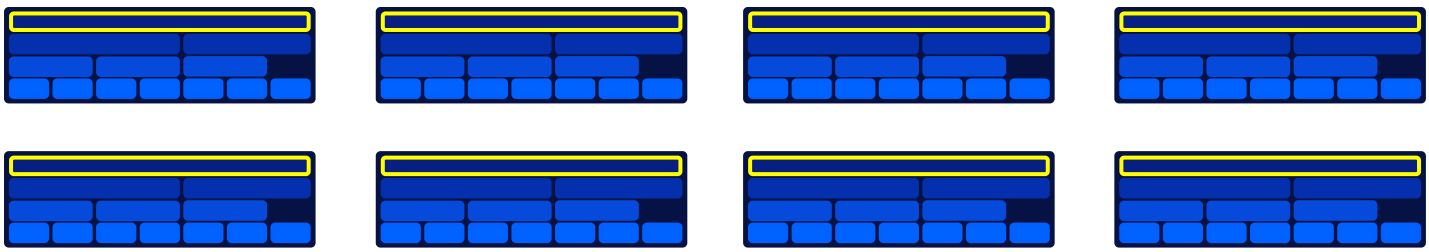

Figure 2 A single partition is created per GPU

```
$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
2001    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2002    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2003    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2004    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2005    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2006    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2007    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2008    rladmin RUN   normal     dgxa        dgxa        large32    Mar 10 09:50
2009    rladmin PEND  normal     dgxa        -           medium16   Mar 10 09:50
. . .
$ bhosts -gpu
HOST_NAME      ID           MODEL     MUSED      MRSV  NJOBS    RUN   SUSP    RSV
dgxa            0 TeslaA100_SXM4_      106M       28G      1      1      0      0
                1 TeslaA100_SXM4_      106M       28G      1      1      0      0
                2 TeslaA100_SXM4_      106M       28G      1      1      0      0
                3 TeslaA100_SXM4_      106M       28G      1      1      0      0
                4 TeslaA100_SXM4_      106M       28G      1      1      0      0
                5 TeslaA100_SXM4_      106M       28G      1      1      0      0
                6 TeslaA100_SXM4_      106M       28G      1      1      0      0
                7 TeslaA100_SXM4_      106M       28G      1      1      0      0
```

- As the initial set of “large32” jobs complete, LSF reconfigures the MIG partitions on all GPUs to meet the requirements of the next jobs pending in the queue. By intelligently partitioning the GPU (see Figure 3), LSF is able to “fit” 2 jobs per GPU. The `bjobs` output shows 8 “medium16” jobs as well as 8 “medium8” jobs running. This means that there are 2 jobs running on each GPU: 1 “medium16” and 1 “medium8”.

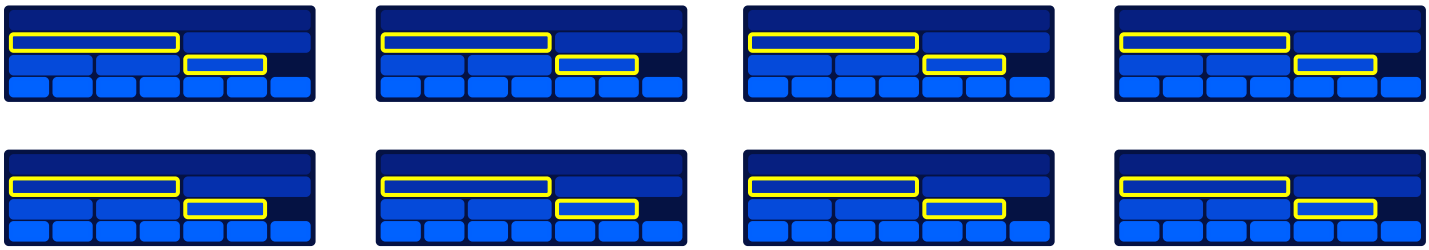

Figure 3 LSF creates 2 MIG partitions to accommodate a “medium16” and “medium8” job per physical GPU

```
$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
2009    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2010    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2011    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2012    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2013    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2014    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2015    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2016    rladmin RUN   normal     dgxa        dgxa        medium16   Mar 10 09:50
2017    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2018    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2019    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2020    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2021    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2022    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2023    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2024    rladmin RUN   normal     dgxa        dgxa        medium8    Mar 10 09:50
2025    rladmin PEND  normal     dgxa                    small4     Mar 10 09:50
```

- As the “medium16” and “medium8” jobs complete, LSF reconfigures the MIG partitions on all GPUs to meet the requirements of the “small4” jobs. By partitioning the GPUs as shown in Figure 4, LSF can dispatch a total of 7 jobs to each GPU. The LSF bjobs output shows the “small4” jobs running, and the `bhosts` output shows a total of 7 jobs per GPU.

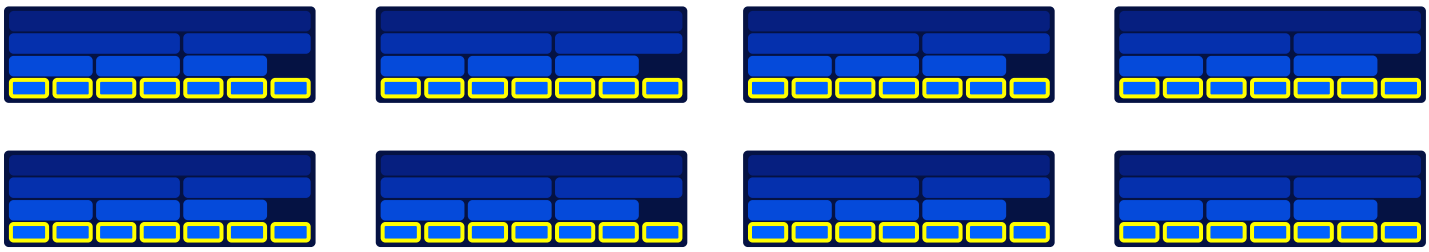

Figure 4 LSF creates 7 MIG partitions per GPU for the “small4” jobs

```
$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
2025    rladmin RUN   normal     dgxa        dgxa        small4     Mar 10 09:50
2026    rladmin RUN   normal     dgxa        dgxa        small4     Mar 10 09:50
2027    rladmin RUN   normal     dgxa        dgxa        small4     Mar 10 09:50
...

$ bhosts -gpu
HOST_NAME      ID           MODEL     MUSED      MRSV  NJOBS    RUN   SUSP    RSV
dgxa            0 TeslaA100_SXM4_      106M       28G      7      7      0      0
                1 TeslaA100_SXM4_      106M       28G      7      7      0      0
                2 TeslaA100_SXM4_      106M       28G      7      7      0      0
                3 TeslaA100_SXM4_      106M       28G      7      7      0      0
                4 TeslaA100_SXM4_      106M       28G      7      7      0      0
                5 TeslaA100_SXM4_      106M       28G      7      7      0      0
                6 TeslaA100_SXM4_      106M       28G      7      7      0      0
                7 TeslaA100_SXM4_      106M       28G      7      7      0      0
```

- As the “small4” jobs complete, LSF reconfigures the MIG partitions on the GPUs for the “large36” jobs. These jobs each require 36GB of GPU memory. Therefore, LSF creates a single MIG partition on each GPU to run the work. The GPUs are partitioned according to Figure 5.

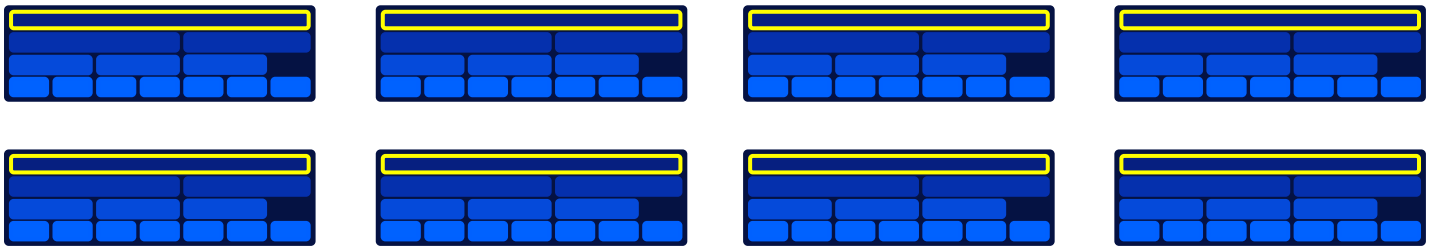

Figure 5 LSF creates a single, large MIG partition per GPU for the “large36” jobs

```
$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
2079    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
2080    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
2081    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
2082    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
2083    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
2084    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
2085    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
2086    rladmin RUN   normal     dgxa        dgxa        large36    Mar 10 09:50
$ bhosts -gpu
HOST_NAME      ID           MODEL      MUSED       MRSV  NJOBS    RUN    SUSP     RSV
dgxa            0 TeslaA100_SXM4_       106M        28G      1      1       0       0
                1 TeslaA100_SXM4_       106M        28G      1      1       0       0
                2 TeslaA100_SXM4_       106M        28G      1      1       0       0
                3 TeslaA100_SXM4_       106M        28G      1      1       0       0
                4 TeslaA100_SXM4_       106M        28G      1      1       0       0
                5 TeslaA100_SXM4_       106M        28G      1      1       0       0
                6 TeslaA100_SXM4_       106M        28G      1      1       0       0
                7 TeslaA100_SXM4_       106M        28G      1      1       0       0
```

This example demonstrated how LSF can dynamically reconfigure MIG partitions on DGX A100. LSF supports dynamic reconfiguration of MIG partitions on NVIDIA Ampere-class GPUs. This capability helps to enable greater utilization and throughput.

Containerized workloads

LSF provides integrated support for running containerized applications with support for Docker, Nvidia Docker, Shifter, Singularity, Podman, and Enroot. All container start-up and filesystem mounting is performed by LSF, which ensures that users never gain elevated privileges. Access to containerized workloads is transparent to users, with container details abstracted using LSF application profiles (`lsb.applications`). This allows the administrator to control who can start a container, the container origin, and how the container is started. When GPUs are requested, GPU reservations are passed into the container by LSF. Below, an example application profile has been defined for a TensorFlow container from the NGC container registry.

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

Example Tensorflow Benchmark jobs

1. In this example, a TensorFlow benchmark is submitted requesting 4 GPUs in interactive mode, running inside an 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: ****.**
----------------------------------------------------------------
```

2. In this example, a TensorFlow benchmark is submitted requesting 1 GPU with 32GB GPU memory in interactive mode, running inside an Nvidia Deep Learning TensorFlow Docker container. Assuming that dynamic management of MIG `LSF_MANAGE_MIG=Y` (`lsf.conf`) has been configured, LSF dynamically creates a single large MIG partition on a single A100 GPU for the job (Figure 6).

Figure 6 LSF creates a single large MIG partition for the 36GB job

```
$ bsub -gpu "num=1:gmem=32" -app tensorflow -I python tf_cnn_benchmarks.py --num_gpus=1 --model resnet50 --
batch_size 128 --num_batches=100
Job <561> 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 running the Tensorflow Benchmark, refer to the following link [tf_cnn_benchmarks: High performance benchmarks](#).