多媒体计算与通讯实验室

GPU 集群 Torque 排队系统使用手册

袁平波 2016.5.20

本实验室新购进 24 块 K80 tesla GPU。为了充分利用 GPU 计算资源，我们利用 Torque 来管理同学们的计算任务队列。头结点的 IP 是 192.168.17.240。下面说明使用本 GPU 集群的几个步骤。

1. 申请帐号.

本集群有一个头结点和多个服务结点构成，因此提交计算作业需要在头结点上拥有帐号，需要使用集群的学生需要给我发一个申请邮件，同时 cc 给自己的导师，在导师批准后相应的帐号会被建立。

2. 建立 job 脚本文件

Torque 管理系统不能直接提交二进制可执行文件，需要编写一个文本的脚本文件，来描述相关参数情况。一个示例脚本文件 myjob1.pbs 如下：

```
#PBS    -N  myjob1
#PBS    -o  /home/username/myjob1.out
#PBS    -e  /home/username/myjob1.err
#PBS    -l nodes=1:gpus=1:S
#PBS    -r y
cd $PBS_O_WORKDIR
echo Time is `date`
echo Directory is $PWD
echo This job runs on following nodes:
cat $PBS_NODEFILE
cat $PBS_GPUFILE
./my_proc
```

脚本文件中定义的参数默认是以#PBS 开头的。其中：

-N 定义的是 job 名称，可以随意。

-o 定义程序运行的标准输出文件，如程序中 printf 打印信息,相当于 stdout;

-e 定义程序运行时的错误输出文件,相当于 stderr。

-l 定义了申请的结点数和 gpus 数量。nodes=1 代表一个结点，一般申请一个结点，除非采用 mpi 并行作业;gpus=1 定义了申请的 GPU 数量,根据应用实际使用的 gpu 数量来确定,S 表示 job 类型，后面有详细描述。队列系统的默认 job 请求时间是一周，如果运行的 job 时间估计会超过，则可以使用下面的参数：

```
#PBS    -l nodes=1:gpus=1:S,walltime=300:00:00
```

表示请求 300 小时的 job 时间。

-r 表示 job 立即执行。

my_proc 是用户的可执行程序。需要通过 scp 或 winscp 复制到自己的 home 目录。如果程序运行过程中需要读取数据文件和生成数据文件，也需要在运行前后上传和下传。后面的 cat 和 echo 信息是打印出时间、路径、运行节点及 GPU 分配情况等信息，便于调试。

3. 提交作业 ：qsub

```
$qsub myjob1.pbs
```

myjob1.pbs 是前一步骤生成的脚本文件。相应可执行

文件和数据文件也必须就位。

4. 查看作业：qstat -n

```
[ypb@torqueServer ~]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
165.torqueServer           my_job1          ypb             00:00:00 C batch
166.torqueServer           my_job1          ypb                    0 R batch
```

上图中 165 是 jobid 运行状态有以下几种状态：

C - Job 已经运行结束
E - Job 运行结束后退出
H - Job 被挂起
Q - job 被排队，可被手动启动或路由
R - job 在运行中.
T - job 被移动
W - job 等待其执行时间到来（-a 选项设置 job 启动时间）

其中-n 参数可以列出运行 job 的结点。

其他常用命令：

1）挂起作业:qhold

Qhold 命令可以挂起作业，被挂起的作业将暂时停止执行，可以让其余的作业优先得到资源运行，被挂起的作业在 qstat 中显示为 H 状态，下面的命令将挂起 id 为 165 的 job。

$qhold 165

2）取消挂起：qrls

被挂起的作业可以重新被运行，如下面的命令将重新运行 id 为 165 的 job

$qrls 165

3）终止作业：qdel

如果用户想放弃一个作业的执行，可以使用 qdel 命令，

下面的命令将终止 id 为 165 的 job。

$qdel 165

4) 查看结点 pbsnodes

$pbsnodes

5) 查看空闲结点 pbsnodes -l free

$pbsnodes -l free

5. 关于集群环境的说明

应同学要求，集群的每一个结点都安装了 caffe 深度学习的环境。包括 Gcc4.8.5, cmake 3.1.3, python 2.7.5, blas3.4.2, numpy 1.9.1,opencv3.0.0。

头结点 torqueServer 没有编译环境，也没有 GPU 卡，如果需要测试自己的代码是否能在集群环境下运行，可以先写一个简单程序在第 7 结点上试运行：

$ssh Gpu107

$./myproc.sh #在这里运行你自己的测试程序。

另外，/opt/下面有下载好的 caffe-master.zip，可以复制到自己目录下：(以下步骤可以在 Gpu107 上完成)

$cp /opt/caffe-master.zip ~/.

a)解压:

$unzip caffe-master.zip

b)配置并修改 config 文件

$cp Makefile.config.example Makefile.config

$vi Makefile.config 修改如下参数

BLAS := atlas

BLAS_LIB := /usr/lib64/atlas

PYTHON_INCLUDE:=/usr/include/python2.7
/usr/lib/python2.7/dist-packages/numpy/core/include

PYTHON_LIB := /usr/lib64

c)编译

$make all -j12

$make test -j 12

$make runtest

d)获取数据

$ sh data/mnist/get_mnist.sh

e)重建 lmdb

$ sh examples/mnist/create_mnist.sh

f)训练数据

$ sh examples/mnist/train_lenet.sh

也可以直接复制已经解压编译好并下载了数据的文件夹（复制过程中有权限错误，忽略不会影响后序过程），这样可以免去 a-d)步骤的编译和数据下载，直接进行 e|f),如下：

$cp /opt/caffe-master ~/. -R

$ sh examples/mnist/create_mnist.sh #重建 lmdb

```
$ sh examples/mnist/train_lenet.sh    #训练
```

6. 关于 GPU 集群的存储问题

用户登录 pbs 头节点(192.168.16.240)后默认的路径是/home/$USER,但/home 下的总空间只有 1T，主要用于存放代码等重要文档，同学们在运行代码过程中用到的数据文件尽量不要放在/home 下，目前可以用于存放数据的 mount 点有/data、/data1、/data2、/data3、/data4、/data5、/data6、/data7,每个 mount 点约 1.5T 空间（使用 df -h 查看)。

同学们可以在/data$i($i=1..7)下建立自己的用户名为子目录，对于一些公共测试数据，可以不放在用户子目录下，而直接放在/datai 下,供大家使用,避免存放大量重复的数据。尤其是同一导师的学生，尽量减少重复下载和存储测试数据。

/data$i($i=1..7)是挂接在 Gpu10$i 结点上的存储,/data 挂接在头结点本地。因此如果有大量数据需要读写并且对 IO 速度有要求的应用,可以考虑把数据存放于某个 mount 点，比如/data3，然后提交 job 时使用参数

```
#PBS   -l nodes=Gpu103:gpus=1
```

则可以使 job 运行在 Gpu103 结点。这样数据和代码运行于同一节点，IO 会避开 nfs 网络操作。但指定节点操作削弱了 pbs 系统的排队功能，可能会导致任务失败。因此除非有特殊要求，一般不建议这么做。

7. 关于 GPU 集群多核心使用情况

使用 GPU 多核心进行运算，可以让 job 缩短完成时间，避免过长的等待时间,从而充分发挥 GPU 的集群优势。但经过测试表明，如果对显存没有特别大的要求，使用单核心的效率还是比多核心要高,因此大家应根据实际情况决定申请 job 类型。

另使用多核心需要注意以下几点：

1） 在申请脚本文件里加入资源申请参数以及打印相应分配的 gpu 情况,便于调试：

```
#PBS  -l nodes=1:gpus=2:D
echo This job runs on following nodes and gpus:
cat $PBS_NODEFILE
cat $PBS_GPUFILE
```

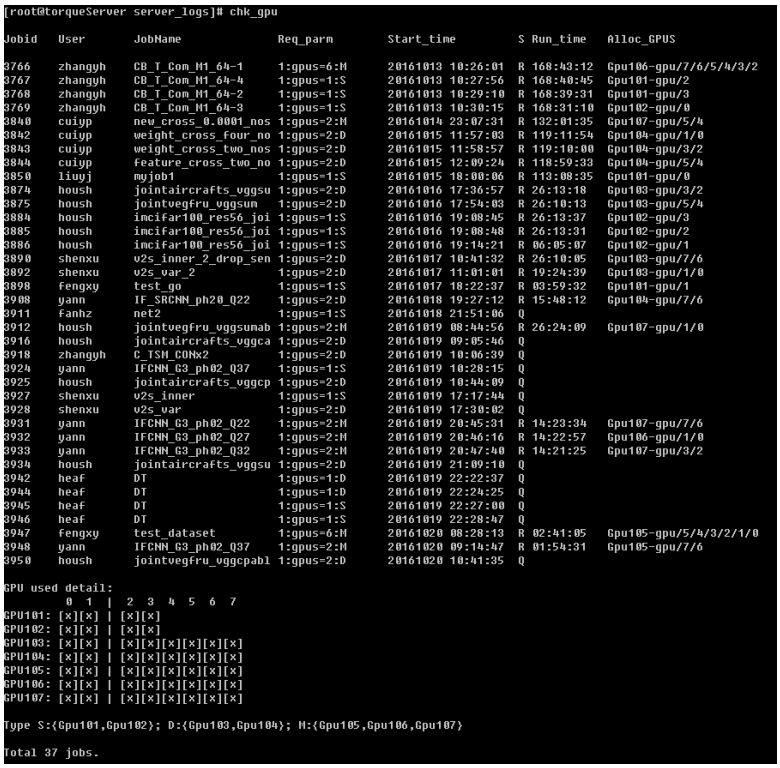
以上是申请一个结点 2 个 GPU 核心的参数。

2） 在运行的程序脚本中加入调用多 GPU 参数，例如 caffe 中需要加入-gpu=all 这样的参数：

```
$caffe train -solver solver.txt -gpu=all
```

如果不加此参数，可能会导致请求的 gpu 数和实际使用的 gpu 数不一致，影响排队系统的工作。

3） 提交 job 后可以使用 chk_gpu 查看占用的 GPU 情况，使用 chk_gpuused〈节点名〉查看结点 GPU 使用率和显存使用情况，节点名为 Gpu101（或 101）等。



```
[root@torqueServer server_logs]# chk_gpu

Jobid   User      JobName             Req_parm       Start_time        S Run_time   Alloc_GPUS

3766    zhangyh   CB_T_Com_M1_64-1    1:gpus=6:M     20161013 10:26:01  R 168:43:12  Gpu106-gpu/7/6/5/4/3/2
3767    zhangyh   CB_T_Com_M1_64-4    1:gpus=1:S     20161013 10:27:56  R 168:40:45  Gpu101-gpu/2
3768    zhangyh   CB_T_Com_M1_64-2    1:gpus=1:S     20161013 10:29:10  R 168:39:31  Gpu101-gpu/3
3769    zhangyh   CB_T_Com_M1_64-3    1:gpus=1:S     20161013 10:30:15  R 168:31:10  Gpu102-gpu/0
3840    cuiyp     new_cross_0.0001_nos 1:gpus=2:M    20161014 23:07:31  R 132:01:35  Gpu107-gpu/5/4
3842    cuiyp     weight_cross_four_no 1:gpus=2:D    20161015 11:57:03  R 119:11:54  Gpu104-gpu/1/0
3843    cuiyp     weight_cross_two_nos 1:gpus=2:D    20161015 11:58:57  R 119:10:00  Gpu104-gpu/3/2
3844    cuiyp     feature_cross_two_no 1:gpus=2:D    20161015 12:09:24  R 118:59:33  Gpu104-gpu/5/4
3850    liuyj     myjob1              1:gpus=1:S     20161015 18:08:06  R 113:08:35  Gpu101-gpu/0
3874    housh     jointaircrafts_vggsu 1:gpus=2:D    20161016 17:36:57  R 26:13:18   Gpu103-gpu/3/2
3875    housh     jointvegfru_vggsum  1:gpus=2:D     20161016 17:54:03  R 26:10:13   Gpu103-gpu/5/4
3884    housh     imcifar100_res56_joi 1:gpus=1:S    20161016 19:00:45  R 26:13:37   Gpu102-gpu/3
3885    housh     imcifar100_res56_joi 1:gpus=1:S    20161016 19:00:48  R 26:13:31   Gpu102-gpu/2
3886    housh     imcifar100_res56_joi 1:gpus=1:S    20161016 19:14:21  R 06:05:07   Gpu102-gpu/1
3890    shenxu    v2s_inner_2_drop_sen 1:gpus=2:D    20161017 10:41:32  R 26:10:05   Gpu103-gpu/7/6
3892    shenxu    v2s_var_2           1:gpus=2:D     20161017 11:01:01  R 19:24:39   Gpu103-gpu/1/0
3898    fengxy    test_go             1:gpus=1:S     20161017 18:22:37  R 03:59:32   Gpu101-gpu/1
3908    yann      IF_SRCNN_ph20_Q22   1:gpus=2:D     20161018 19:27:12  R 15:48:12   Gpu104-gpu/7/6
3911    fanhz     net2                1:gpus=1:S     20161018 21:51:06  Q
3912    housh     jointvegfru_vggsunab 1:gpus=2:M    20161019 08:44:56  R 26:24:09   Gpu107-gpu/1/0
3916    housh     jointaircrafts_vggca 1:gpus=2:D    20161019 09:05:46  Q
3918    zhangyh   C_TSM_CONx2         1:gpus=2:D     20161019 10:06:39  Q
3924    yann      IFCNN_G3_ph02_Q37   1:gpus=1:S     20161019 10:28:15  Q
3925    housh     jointaircrafts_vggcp 1:gpus=2:D    20161019 10:44:09  Q
3927    shenxu    v2s_inner           1:gpus=1:S     20161019 17:17:44  Q
3928    shenxu    v2s_var             1:gpus=2:D     20161019 17:30:02  Q
3931    yann      IFCNN_G3_ph02_Q22   1:gpus=2:M     20161019 20:45:31  R 14:23:34   Gpu107-gpu/7/6
3932    yann      IFCNN_G3_ph02_Q27   1:gpus=2:M     20161019 20:46:16  R 14:22:57   Gpu106-gpu/1/0
3933    yann      IFCNN_G3_ph02_Q32   1:gpus=2:M     20161019 20:47:40  R 14:21:25   Gpu107-gpu/3/2
3934    housh     jointaircrafts_vggsu 1:gpus=2:D    20161019 21:09:10  Q
3942    heaf      DT                  1:gpus=1:D     20161019 22:22:37  Q
3944    heaf      DT                  1:gpus=1:D     20161019 22:24:25  Q
3945    heaf      DT                  1:gpus=1:S     20161019 22:27:00  Q
3946    heaf      DT                  1:gpus=1:S     20161019 22:28:47  Q
3947    fengxy    test_dataset        1:gpus=6:M     20161020 08:28:13  R 02:41:05   Gpu105-gpu/5/4/3/2/1/0
3948    yann      IFCNN_G3_ph02_Q37   1:gpus=2:M     20161020 09:14:47  R 01:54:31   Gpu105-gpu/7/6
3950    housh     jointvegfru_vggcpab1 1:gpus=2:D    20161020 10:41:35  Q

GPU used detail:
        0  1  |  2  3  4  5  6  7
GPU101: [x][x] | [x][x]
GPU102: [x][x] | [x][x]
GPU103: [x][x] | [x][x][x][x][x][x]
GPU104: [x][x] | [x][x][x][x][x][x]
GPU105: [x][x] | [x][x][x][x][x][x]
GPU106: [x][x] | [x][x][x][x][x][x]
GPU107: [x][x] | [x][x][x][x][x][x]

Type S:{Gpu101,Gpu102}; D:{Gpu103,Gpu104}; M:{Gpu105,Gpu106,Gpu107}

Total 37 jobs.
```

4） 如果发现请求的 gpu 数和实际使用的 gpu 数不一致的 job 将被清除。

5） （本节内容自 2016.12.9 后已经调整，不再启用）

由于多个 GPU 核心是不同的 CPU 控制，因此在不同的 CPU 控制的 GPU 核心之间是 socket 连接的，不是 PCIe 桥接或 PCI 内部交换的，而 caffe 等软件目前可能不支持 socket 连接的多 GPU 核心运算。为了解决这个问题，目前做如下规定：

(a) 使用单核的 job 必须加参数 S，如：

```
#PBS  -l nodes=1:gpus=1:S
```

(b) 使用双核的 job 必须加参数 D，如：

```
#PBS  -l nodes=1:gpus=2:D
```

(c) 使用>=3 核的 job 必须加参数 M，如：

```
#PBS  -l nodes=1:gpus=6:M
```

可以采用命令 chk_gpu 来查看各个结点已经使用的 GPU 核心详细情况。

由于服务器两 CPU 管理的 GPU 并不平均，0-1 核心是一个 CPU 管理，2-7 核心是另一个 CPU 管理。因此，为了防止 job 跨不同 CPU 管理的核心，作以下要求：对于 S、D 类型的 job 可以随时提交队列进行排队；对于 M 类型的 job （仅 5，6，7 结点支持）不能轻易排队，必须要看 chk_gpu 显示的 detail 情况，来决定提交 2:M 还是 6:M 类型 job。

具体申请原则规定如下：

a)S、D 类型 job 可以随时提交排队；

b)对 M 类型节点，

如果有 0,1 位 GPU 空闲，则可提交 2:M 类型 job;

如果有 3~7 位 GPU 空闲，则可提交 6:M 类型 job;

如果 0~7 位 GPU 都空闲，则必须先提交 2:M 类型 job, 再提交 6:M 类型 job，次序不可颠倒。

b) 如果有 6:M 类型 job 排队，则 M 型节点 2~7 位 GPU 原则上是不允许提交 2:M 型 job 的，如果是暂时占用，那么当 6:M 所需要的资源满足时，暂时占用的 job 即使没有完成,也会被删除。

更新日志：

2016.12.9

增加了新的结点 Gpu108 和 Gpu109，每个结点 8 块 K80 卡，16 个 GPU 核心。

增加了存储空间/data8 和/data9。

结点类型做如下调整,取消 M 类型作业：

1）101-104 为 S 型单核，提交时使用参数 nodes=1:gpus=1:S

2）105-107 为 D 型双核，提交时使用参数 nodes=1:gpus=2:D

3）108 为 Q 型 4 核，提交时使用参数 nodes=1:gpus=4:Q

4）109 为 E 型 8 核，提交时使用参数 nodes=1:gpus=8:E

取消 M 型结点。

另外，提请大家注意，近来发现 NFS 服务消耗网络带宽非常大，因此大家尽量把计算数据存放到相应类型的结点，比如使用 1S job 类型的计算数据尽量存放在 S 结点 101-104 上。

2016.12.20

增加了新的结点 Gpu110，8 块 K80 卡，16 个 GPU 核心。

结点类型做如下调整：

1）101-105 为 S 型单核，提交时使用参数 nodes=1:gpus=1:S

2）106-108 为 D 型双核，提交时使用参数 nodes=1:gpus=2:D

3）109 为 Q 型 4 核，提交时使用参数 nodes=1:gpus=4:Q

4）110 为 E 型 8 核，提交时使用参数 nodes=1:gpus=8:E

2017.1.18

增加了新的结点 Gpu111、Gpu112，每个结点 8 块 K80 卡，16 个 GPU 核心。

考虑到 S 类型结点需求较大，故结点类型做如下调整：

1）101-107 为 S 型单核，提交时使用参数 nodes=1:gpus=1:S

2）108-110 为 D 型双核，提交时使用参数 nodes=1:gpus=2:D

3）111 为 Q 型 4 核，提交时使用参数 nodes=1:gpus=4:Q

4）112 为 E 型 8 核，提交时使用参数 nodes=1:gpus=8:E

2017.1.22

增加了新的结点 Gpu113，8 块 K80 卡，16 个 GPU 核心。

结点类型安排如下：

1）101-107 为 S 型单核，提交时使用参数 nodes=1:gpus=1:S

2）108-110 为 D 型双核，提交时使用参数 nodes=1:gpus=2:D

3）111 为 Q 型 4 核，提交时使用参数 nodes=1:gpus=4:Q

4）112-113 为 E 型 8 核，提交时使用参数 nodes=1:gpus=8:E

到目前为止，集群共拥有 72 块 K80，144 核心。

2017.3.7

扩展了 chk_gpu 命令，增加帐号名作为参数。列出用户使用 GPU 情况。

新增命令 whois <帐号名>，查询帐号对应的人员姓名。

为了避免 107 结点运行较大较长时间测试程序，影响正常作业的提交，新增了守护进程，限制 Gpu107 运行的测试程序不能超过 15 分钟。

2017.3.27

增加了存储/data10 /data11

安装了 torch 框架

提供网页查看 job 使用 GPU 情况:

查看个人 job:

http://202.38.69.241:38240/chk_gpu.php?userid=*your_userid*

查看全部 job：

http://202.38.69.241:38240/chk_gpu.php

2017.4.19

增加了存储 /data12 /data13

升级 cuda 从 7.5 到 8.0 ,同时升级了卡驱动

升级 tensorflow 从 0.10.0 到 1.0.1。

2017.4.24

增加了新的结点 Gpu114,4 块 Nvidia Tesla K80，8 核心。至此，集群共有 76 块 Nvidia tesla K80，152 核心。

2017.5.4

安装了 python3.4.5 环境,并安装了相应的支持 CPU 加速功能的兼容 SSE4.1、SSE4.2、AVX、AVX2 和 FMA 的 tensorflow-1.1.0。python 和 pip 两个版本均有对应命令 pip2/pip3 和 python2 /python3,请使用是注意版本。

解决了提交超出个人最大数量限制的 job 会假死的 bug。可以提交任意数量的 job 进行排队。

2017.5.6

为了解决 IO 频繁的应用因 nfs 带来的性能下降问题，集群支持应用使用每个结点的 tmpfs（内存虚拟的文件系统），使用方法是：在运算开始前，将数据文件从/datai 位置 copy 到/dev/shm。由于 tmpfs 是多用户竞争使用，故使用前需要检测 tmpfs 是否足够存放自己的数据文件,可以在脚本文件中加入如下代码：

```
avl_size=`df |grep "/dev/shm"|awk '{printf "%d",$4/1024}'`
[ $avl_size -lt 2000 ] && echo "not enough tmpfs! " && exit
//here copy your data to /dev/shm/.
//here run your main program.
```

上例是数据文件为 2G 的示例,实际使用时用数据文件的实际大小取代 2000,单位是 M。

由于 tmpfs 空间有限，Gpu101-Gpu102 为 32G，Gpu103-Gpu107，Gpu114 为 63G，Gpu108-Gpu113 为 252G。因此如果对 IO 没有特殊要求的 job 无需使用此空间做缓存。

2017.5.7

增加命令 chk_tmpfs [nodename]， 用于查询每个结点可用的 tmpfs 大小，单位 M。如：

$chk_tmpfs Gpu108

查询 Gpu108 结点的可用 tmpfs 空间，如不带参数则列出全部结点

增加命令 clean_tmpfs [nodename] 用于删除本用户在目标结点 tmpfs 中的全部文件。如：

$clean_tmpfs Gpu108

清除 Gpu108 结点中本用户在/dev/shm 下的全部文件。如果不带参数，则清除全部结点的 tmpfs 文件。本命令在有 job 运行阶段慎用，尤其是不带参数的命令，有可能会删除正在用的数据。一般在 qdel 删除 job 后可以用本命令清除运行结点的 tmpfs 缓存数据。

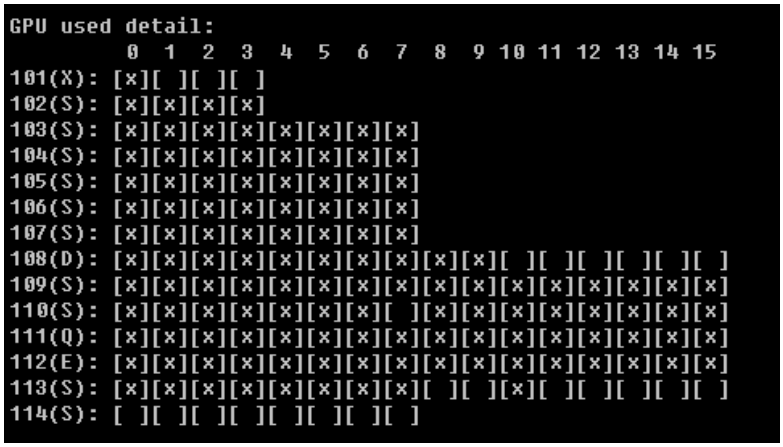
增加命令 ls_tmpfs [nodename] 用于列出目标结点 tmpfs 中文件详细情况。如：

$ls_tmpfs Gpu108

2017.10.20

调整 Gpu101 节点为调试节点，原调试节点 Gpu107 关闭调试功能。

结点类型调整到结点名称后的括号内显示。如 108(D)表示结点 Gpu108 的 job 类型是 D 双核。



2017.11.28

为了解决集群的多环境问题，在集群中引入 Docker 容器技术。集群采用 docker-ce-17.09 社区版本。在调试节点使用 docker 环境的方法是：

$startdocker -P <program path> [-D <data path>] -s <prog> <dockerimage>

其中：-P 是可执行文件或配置文件所在的路径的祖先节点，必须有。

-D 是数据文件输入输出路径，可以没有，如和-P 参数一样则必须省略。

-s 是可执行二进制或脚本文件名，可以给出绝对路径或相对于-P 的相对路径。脚本文件可以是 shell 脚本或 python，需要在第一行加入：#!/bin/sh 或#!/bin/python

-c 是执行命令行，和-s 的区别是本参数不会处理相对路径，只解释为命令行，和-s 两者中只能出现一个。

例如：/home/xxx/pro1/proc.sh 是要执行的文件，数据输出为 /data2/xxx/prox1/proc.log，docker 镜像为 mcc:5000/tf_py3，则可以使用：

$startdocker -P /home/xxx -D /data2/xxx -s “pro1/proc.sh” mcc:5000/tf_py3

-D 和 -P 参数只需要是数据或程序的路径祖先目录即可，但不能是文件。

$startdocker -P /home/xxx -D /data2/xxx -c “echo hello” mcc:5000/tf_py3

可以使用

$docker images

查看当前所有的 docker images，

```
[root@Gpu101 ypb]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mcc:5000/tf_py3     latest              a799c799a4f2        9 days ago          2.16GB
mcc:5000/cuda       latest              d2f66e42cac2        3 weeks ago         1.03GB
```

目前提供的 image 有：

mcc:5000/cuda 是 cuda8.0-cudnn6-centos7 环境；

mcc:5000/tf_py3 是在前一镜像基础上安装了 python3 和 tensorflow_gpu_1.4.0，可以使用下面命令查看镜像的环境：

$docker inspect mcc:5000/tf_py3 |grep Comment

在 pbs 中调用 docker，只需要把 $startdocker 语句放进 pbs 的 script 文件即可。

2017.12.10

在内网增加了查看计算结点资源使用的方式。使用浏览器打开：

http://<结点 ip>:61208

其中结点 ip 是 192.168.6.101~192.168.6.114. 例如查看 110 结点的资源情况：

http://192.168.6.110:61208

由于本服务的后台本身会占用内存和 CPU 资源，所以不需要查看时请关闭浏览器进程。