华东师范大学超算中心

作业调度系统策略配置和使用情况说明

华东师大超算中心作业调度系统策略配置情况说明

曙光为华东师范大学超算中心提供的作业调度系统 Gridview 之 DPBS，源于 PBS 系统，集成于 Gridview，带有自己的功能强大的调度器。兼容 OpenPBS 2.3 的所有语法。

Gridview 集成的 DPBS 曙光内部版本是 2.0；调度器版本为 2.0。

1．作业队列

目前整个系统共计 63 台计算节点(node1-63 每个节点 8 CPU CORE 和 16GB 内存），1 台登录节点（node64，配置 8CPU CORE，16 GB 内存）1 台管理节点（node65，配置 8 CPU CORE，16GB 内存）；1 台 I/O 节点（node65，配置 8 CPU CORE，16 GB 内存）。

作业调度系统的管理节点是 node64 和 node65，在这两个节点上用管理员帐户 root 都可以管理（删除或者取消）普通用户提交的作业。

共设定四个队列，high，parallel，debug， serial。 其中 high（具有特权的用户使用)为高优先级抢占队列，可以抢占其它队列的作业。parallel，debug，serial 为低优先级队列，其作业可以被 high 队列中的作业所抢占。默认队列为 parallel，所有用户的作业长度不受限制，但其优先级也最低，优先级值越大，优先级也越高。

Queue	***CPU-Time Limit***	***No. of Jobs in Parallel***	***Priority***
high	unlimited	unlimited	11000
debug	2 小时	4	6000
parallel	30 天	8	9000
serial	30 天	64	6000

资源分区，利用 Reservation 的方法实现资源分区概念： 分为串行作业计算节点区 (node1-8)、作业编译调试区 (node61-63) 和并行作业运行区 (node9-60)。

目前所有计算队列基本未加更进一步的限制，任何用户可以提交任意数目的长短作业。

2 调度策略

四个队列中，high（业务系统）为高优先级抢占队列，可以抢占其它队列的作业。 debug，parallel 和 serial 为低优先级队列，其作业可以被 high 队列中的作业所抢占。 默认队列为 parallel，所有用户的作业长度在主机试运行期间不受限制；优先级值越大，优先级也越高。

普通节点区未加限制，任何用户可以提交任意数目的长短作业。

3 作业调度系统使用举例

3.1 Job Arrays

Job Arrays 是一种将相关工作分组的机制，允许使用者提交，查询，修改以及显示一个集合的工作。这个新的功能对于一些必须提交以及管理大量相关工作的 user 来说是相当实用的。

测试算例为：

```
dolphin#    cat helloworld.cc
#include <iostream>
#include <stdlib.h>


int main()
{
        std::cout << "hello world!" << std::endl;
        system("echo my present working directory is `pwd` !");

}
```

测试脚本为：

```
#!/bin/bash -x

#PBS -N my.job.array
#PBS -t 0-3
#PBS -l nodes=1:ppn=1
#PBS -l walltime=60:00:00
#PBS -j oe
#PBS -q serial

#
#define variables
#
echo "This jobs is "$PBS_JOBID@$PBS_QUEUE

cd ${PBS_O_WORKDIR}/case${PBS_ARRAYID}

sleep 100s

date
time ~/bin/helloworld
date
```

脚本解释：

#PBS –t 0-3 指定 PBS_ARRAYID 的值从 0 递增到 3，即共有 0、1、2、3 四个值。

#PBS -l nodes=1:ppn=1 指定申请的 CPU 资源为 1 个 CPU。

这个脚本提交以后，实际上将有四个串行作业同时提交，工作目录将分别是 case0，case1，case2，case3。

3.2 并行作业

并行作业的脚本以 cpi 为例。这里采用 INTEL MPI impi 3.2.2.006 为例进行实例。作业脚本如下：

```
dolphin@CLOUD@ECNU:~/work/parallel> cat 1.job.impi
#!/bin/bash -x

#PBS -N mycpi
#PBS -l nodes=4:ppn=8
#PBS -l walltime=00:08:00
#PBS -j oe
#PBS -q parallel

#
#define variables
#
n_proc=$(cat $PBS_NODEFILE | wc -l)

#
#running jobs
#
cd $PBS_O_WORKDIR

#
# Setup the MPI topology
#

time -p /data/soft/compiler/mpi/impi/3.2.2.006/bin64/mpirun --rsh=ssh -env I_MPI_DEVICE rdma:OpenIB-cma -np ${n_proc} ./cpi

exit 0
```

其中，#PBS -l nodes=4:ppn=8 表明申请资源为 4 个节点，每个节点申请 8 个 CPU。

#PBS -q parallel 表示提交到 parallel 队列。

time -p /data/soft/compiler/mpi/impi/3.2.2.006/bin64/mpirun --rsh=ssh -env

I_MPI_DEVICE rdma:OpenIB-cma -np ${n_proc} ./cpi

是 INTEL MPI 运行基于 Infiniband 网络的应用程序时采用的格式。

3.3 串行作业

采用经典的 Helloworld，测试脚本如下：

```
#!/bin/bash -x

#PBS -N my.serial.array
#PBS -l nodes=1:ppn=1
#PBS -l walltime=60:00:00
#PBS -j oe
#PBS -q serial

#
#define variables
#
echo "This jobs is "$PBS_JOBID@$PBS_QUEUE

cd ${PBS_O_WORKDIR}

./helloworld

exit 0
```

#PBS -l nodes=1:ppn=1 表示申请 1 个节点上的 1 颗 CPU。

#PBS -q serial 表示提交到集群上的 serial 队列。

3.4 串行作业的 Checkpoint/Restart

Checkpoint/Restart 是曙光作业调度系统 Gridview 提供的一项高级功能，可以提供串行作业的断点续算。Checkpointing 的开启通常要耗费巨大的计算资源。所以用户需要斟酌是否开启 Checkpoint 功能；测试程序源代码如下：

```
dolphin@CLOUD@ECNU:~/work/cr> cat test.c
#include "stdio.h"
int main( int argc, char *argv[] )
{
int i;
        for (i=0; i<1000; i++)
        {
                printf("i = %d\n", i);
                fflush(stdout);
                sleep(1);
        }
}
```

编译生成可执行程序

```
dolphin@CLOUD@ECNU:~/work/cr> gcc test.c -o test
```

测试脚本如下：

```
dolphin@CLOUD@ECNU:~/work/cr> cat test.job
#!/bin/bash

#PBS -N my.cr.job
#PBS -j oe
#PBS -l walltime=00:10:00
#PBS -c enabled,periodic,shutdown,interval=1,dir=/public/users/dolphin/work/cr
#PBS -q serial

cd $PBS_O_WORKDIR
./test

dolphin@CLOUD@ECNU:~/work/cr>
```

#PBS -c enabled,periodic,shutdown,interval=1,dir=/public/users/dolphin/work/cr 表示开启 Checkpoint 功能（enabled）；periodic 表明进行周期性 Checkpoint；shutdown 表明当 PBS Server 宕机时进行 Checkpoint；interval=1 表明进行 Checkpoint 的最小 CPU 时间间隔为 1 分钟（单位为分钟）；dir=/public/users/dolphin/work/cr 表明 chekpoint 文件的保存目录为 /public/users/dolphin/work/cr。

```
dolphin@CLOUD@ECNU:~/work/cr> qsub test.job
1363.node65
dolphin@CLOUD@ECNU:~/work/cr> qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1363.node65               my.cr.job        dolphin                0 R serial
dolphin@CLOUD@ECNU:~/work/cr> qpeek 1363
i = 0
i = 1
i = 2
i = 3
i = 4
```

qsub 提交作业后，用 qstat 查看其运行状态，显示为 Run，运行在 serial 队列上；qpeek 1363 查看作业号为 1363 的运行输出文件；

```
dolphin@CLOUD@ECNU:~/work/cr> qhold 1363
dolphin@CLOUD@ECNU:~/work/cr> qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1363.node65               my.cr.job        dolphin         00:00:00 H serial
dolphin@CLOUD@ECNU:~/work/cr> qpeek 1363
i = 15
i = 16
i = 17
dolphin@CLOUD@ECNU:~/work/cr> qrls 1363
dolphin@CLOUD@ECNU:~/work/cr> qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1363.node65               my.cr.job        dolphin         00:00:00 R serial
dolphin@CLOUD@ECNU:~/work/cr> qpeek 1363
i = 19
i = 20
i = 21
```

qhold 1363 表示挂起 1363 这个作业；此时用 qstat 查看时，其状态处于 hold 状态；用 qpeek 1363 查看其结果输出情况，发现不再继续输出；

qrls 1363 表示把处于 hold 状态的 1363 号作业重新运行；此时用 qstat 查看，其状态处于 run 状态；用 qpeek 1363 查看器输出情况，发现该作业从原来停顿的地方继续计算下去了。

以上实验表明，当作业由于被高优先级作业抢占而 hold，当有资源可以计算时，可以从被 hold 的地方续算下去。