# Руководство пользователя

LINUX СЕГМЕНТ КЛАСТЕРА «ЛОБАЧЕВСКИЙ»

Версия 2017.01

ННГУ ИМ. Н.И. ЛОБАЧЕВСКОГО | Г. НИЖНИЙ НОВГОРОД, ПР. ГАГАРИНА 23

# Оглавление

# 1 Основная информация

Кластер доступен из сети интернет по ip адресу 85.143.2.188 с использованием протокола ssh. В большинстве дистрибутивов Linux имеется встроенный ssh-клиент, для ОС семества Windows можно использовать сторонние клиенты – PuTty, MobaXterm. При логине по заданному адресу пользователю предоставляется узел для компиляции и запуска своих задач. Помимо домашней директории (*$HOME*) пользователю предоставляется дисковое пространство в сетевой ФС */common/home/username*, где рекомендуется хранить все данные для своих задач и запускать их на счет.

Пользователь дополнительно может логиниться по ssh на вычислительные узлы, если на них в данный момент времени выполняется задача, запущенная от имени этого же пользователя. На все остальные узлы кластера, включая счетные, на которых в данный момент нет активных пользовательских задач, доступ по ssh пользователям закрыт.

Для управления ресурсами кластера и ведения очереди задач используется SLURM [1].

Кластер в SLURM разбит на 3 раздела:

- cpu – состоит из 10 узлов node101 - node110 без ускорителей;
- gpu – состоит из 100 узлов node1 - node90, node111 - node120 с ускорителями Nvidia Tesla K20X;
- phi – состоит из 10 узлов node91 - node100 с ускорителями Intel Xeon Phi 5110P и node141- node152 с ускорителями Intel Xeon Phi 7210.

При постановке задачи в очередь нужный раздел кластера задается ключем *-p* в командах *srun/sbatch/salloc*. По умолчанию используется раздел cpu. Дополнительно есть возможность указать раздел all, в который входит все 132 счетных узла кластера.

Используется единая очередь задач для всех пользователей кластера. По умолчанию лимит времени на одну задачу составляет 5 минут. Максимальное запрашиваемое время на одну задачу – 3 дня. Приоритет в очереди зависит от количества запрашиваемых ресурсов и времени нахождения задачи в очереди.

Возможны два основных вида запуска задач – в интерактивном режиме, когда вывод программы производится в stdout (shell), и в пакетном, когда подготавливается скриптовый файл с необходимыми командами, ставится в очередь на выполнение, а весь вывод направляется в заданный файл. Для интерактивного режима используются команды *salloc* и *srun*, для пакетного – sbatch.

Чтобы получить информацию о занятых/свободных узлах кластера, используется *sinfo*. Пример ее выдачи:

| PARTITION | AVAIL | TIMELIMIT | NODES | STATE | NODELIST |
|---|---|---|---|---|---|
| gpu | up | infinite | 1 | drain* | node82 |
| gpu | up | infinite | 6 | down* | node[20,44-45,49,71,114] |
| gpu | up | infinite | 8 | comp | node[1-8] |
| gpu | up | infinite | 5 | drain | node[10,42,53,90,118] |
| gpu | up | infinite | 5 | alloc | node[43,83-86] |
| gpu | up | infinite | 75 | idle | node[9,11-19,21-41,46-48,50-52,54-70,72-81,87-89,111-113,115-117,119-120] |
| cpu* | up | infinite | 3 | drain* | node[103,106,109] |
| cpu* | up | infinite | 7 | drain | node[101-102,104-105,107-108,110] |
| phi | up | infinite | 10 | idle | node[91-100] |
| all | up | infinite | 4 | drain* | node[82,103,106,109] |
| all | up | infinite | 6 | down* | node[20,44-45,49,71,114] |
| all | up | infinite | 8 | comp | node[1-8] |
| all | up | infinite | 12 | drain | node[10,42,53,90,101-102,104-105,107-108,110,118] |

```
all          up    infinite    5          alloc      node[43,83-86]

all          up    infinite    85         idle    node[9,11-19,21-41,46-48,50-
52,54-70,72-81,87-89,91-100,111-113,115-117,119-120]
```

В поле STATE указывается состояние счетных узлов:

- idle - узел свободен и готов для счета;

- alloc – на узле считается задача;

- comp – на узле выполняется epilog-скрипт;

- maint – узел зарезервирован для определенных пользователей;

- down, down*, drain, comp*, idle*,alloc* – узел недоступен/выведен из счета/неполадки на узле.

Для просмотра очереди задач используется *squeue*:

```
JOBID PARTITION NAME       USER  ST    TIME  NODES      NODELIST(REASON)
2112  cpu       benchuser1 PD    0:00   3          (Resources)
2109  cpu       bash       user1 R     10:43  2          node[101-102]
2110  gpu       test       user2 R     0:37   4          node[3-6]
2107  phi       task1      user3 CG    0:07   1          node91
2106  cpu       task2      user1 CG    0:03   4          node[103-106]
```

Выводится таблица со столбцами: JOBID – идентификатор задачи, PARTITION - название раздела кластера, в который помещена задача, NAME – имя задачи, USER – имя пользователя, от которого запущена задача, ST – статус задачи, TIME – время работы задачи, NODES – число узлов для задачи, NODELIST(REASON) – список узлов, или причина, по которой они еще не выделены (строка «Resources» в столбце NODELIST(REASON) говорит о недостаточном количестве ресурсов для выполнения задачи). В данном случае выполняется 2 задачи (R), для двух задач (CG) выполняется epilog-скрипт очистки узлов от пользовательских задач, еще одна задачи находится в ожидании (PD) ресурсов. Подробнее – *man squeue*.

# 2 Запуск однопоточных и многопоточных программ

Для запуска в интерактивном режиме используется команда *srun*:

srun [опции] [исполняемый файл] [аргументы]

Например,

```
~> srun -N 5 –n 5 -p phi hostname
node92
node91
node95
node93
node94
```

запускает hostname на 5 узлах из раздела phi.

Некоторые ключи-опции:

- -N x — задает количество узлов (x) для задачи;

- -n x — задает число исполняемых процессов (x); по умолчанию – 1 процесс на узел;

- -p <partition> - задает раздел класетра для задачи, доступные разделы – cpu, phi, gpu, all, по умолчанию – cpu.

Для запуска мультитредовых приложений используется ключ

- -c x — выделяет x ядер на один процесс, по умолчанию выделяется 1 ядро на процесс;

Для привязки процессов к определенным ядрам используется ключ --cpu_bind, с помощью которого можно назначить ядра для процессов по маске, id ядер, так же поддерживаются и высокоуровневые привязки (к сокетам, доменам памяти). Для привязки с указанием id ядер:

--cpu_bind=map_cpu:<id1>,<id2>,<id3>...

Для привязки openMP тредов используется переменная GOMP_CPU_AFFINITY:

```
GOMP_CPU_AFFINITY="0 2 4 6 8 10 12" OMP_NUM_THREADS=7 srun  -n1 ./loop
```

В данном примере loop запускается с привязкой 7 openMP-нитей к ядрам 0, 2, 4, 6, 8, 10, 12.

Подробнее о привязках в SLURM смотри [**Ошибка! Источник ссылки не найден.**].

С помощью ключа --begin=<time> задается время начала счета задачи, поддерживаются различные форматы времени, например HH:MM, YYYY-MM-DD[THH:MM[:SS]]

Для ограничения времени счета задачи используется ключ –t. Доступные форматы: MM, MM:SS, HH:MM:SS, DD-HH:MM:SS. Подробнее по этим и другим ключам – man srun.

Для запуска задач в пакетном режиме используется *sbatch*:

sbatch [options] script [args...] ,

где script – имя файла с командами для выполнения задания. Ключи могут задаваться не только в комадной строке([options]), но и в самом скриптовом файле, до выполнения команд, с префиксом #SBATCH

Некоторые ключи:

- -o filename – перенаправляет вывод в filename (по умолчанию – в slurm-%j.out, где %j – идентификатор задания);

- -J jobname – определяет имя задания;

- -w <nodelist> - задает список узлов для задачи, например, «node2,node5», «node[3-7]».

Например,

```
cat my.script
        #!/bin/bash
        #SBATCH --time=1
        /bin/hostname
        srun -l /bin/hostname
        srun -l /bin/pwd

        sbatch -w «node3,node4» -o output.txt ./my.script
        Submitted batch job 2116

cat output.txt
        node3
        0: node3
```

1: node4
1: /common/user
0: /common/user

# 3 Выделение узлов для пользователей

Пользователь может логиниться по ssh только на те узлы, которые SLURM выделила ему для счета его задачи и только во время выполнения этой задачи, в других ситуациях доступ пользователей на счетные узлы закрыт. Чтобы получить shell-доступ к N счетным узлам, не запуская конкретных задач, рекомендуется использовать *salloc*:

```
salloc -p <partition> -t <time_min> -N <num_nodes>
```

Эта команда возвращает bash-shell, из которого доступ на счетные узлы выполняется по ssh, при этом в очереди slurm появляется запись о соответствующем задании, из которой можно узнать список выделенных узлов:

```
Squeue

JOBID PARTITION NAME     USER ST  TIME  NODES     NODELIST(REASON)
2117  phi       bash     user R   0:06  2         node[91-92]
```

# 4 Удаление задач из очереди

Для принудительного удаления задачи из очереди (выполняющейся, ожидающей ресурсов) служит scancel:

scancel JOBID ,

где JOBID – идентификатор задания, его можно узнать по *squeue*:

squeue

```
JOBID PARTITION NAME     USER ST  TIME NODES NODELIST(REASON)
2120  debug     bash     alexiv R   0:03 2     node[1-2]
```

scancel 2120

# 5 Компиляция и запуск MPI-программ

На кластере доступно несколько версий библиотеки MPI – Intel MPI, OpenMPI, MVAPICH2. Перед компиляцией и запуском задач необходимо выбрать определенную версию командой

- module load mva (для MVAPICH2);

- module load ompi (для OpenMPI);

- module load impi (для IntelMPI).

Помимо штатной версии компиляторов GNU (4.4.7) доступна более новая (4.8.2 с библиотеками boost) по команде:

module load gcc-4.8.2

После выбора версии компиляторов и mpi компиляция выполняется стандартным образом с использованием mpic*, mpif* команд.

Запуск mpi-задач в интерактивном режиме выполняется следующей командой:

srun -t <time> -p <partition> --ntasks-per-node <ppn> -N <num_nodes> -n <num_procs> --cpu_bind=v,map_cpu:<core0>,<core1>...<coreM> ,

где:

- <time> – лимит времени на задачу;

- <partition> – раздел кластера;

- <ppn> – число процессов на узел;

- <num_nodes> – число узлов;

- <num_procs> – общее число процессов;

- <core0>,<core1>...<coreM> – привязка i-го процесса на узле к <corei> ядру.

Не все из перечисленных опций являются обязательными, есть и другие, подробнее – *man srun*.

Пример запуска теста IMB, собранного с OpenMPI с лимитов времени в 10 минут, в разделе gpu на 3 узлах, 6 процессов на узел, спривязкой процессов к 0-му и 8-му ядру:

```
module load ompi
srun -t 10 -p gpu --ntasks-per-node 2 -N 3 -n 6 \ --cpu_bind=v,map_cpu:0,8
/common/openmpi/imb/IMB-MPI1
```

Для запуска задач в неинтерактивном, пакетном режиме используется команда *sbatch*, для которой пишется скрипт запуска (см. раздел 6.1) или он создается автоматически:

```
sbatch -o ./out.txt --ntasks-per-node 2 -N 3 -n 6 -t 10 -p gpu --wrap="srun \ --
cpu_bind=v,map_cpu:0,8 /common/openmpi/imb/IMB-MPI1"
```

В последнем случае большинство параметров из *srun* выносится в *sbatch*, подробнее *man sbatch*.

# 6 Запуск программ на Xeon Phi

Для использования узлов с Xeon Phi при постановке задачи в очередь используется ключ –p phi для указания раздела кластера, содержащего узлы с сопроцессорами Xeon Phi. Для запуска задач в symmetric и native режимах вызывается скрипт mpirun.mic. Для использования скрипта предварительно нужно подключить модуль impi2017 с помощью команды module load impi2017.

Пример запуска задачи в интерактивном/пакетном режиме:

```
salloc/sbatch -N num_nodes -p phi mpirun.mic -x
ppn_host -c ./impi_native_hybrid -z ppn_mic -m
./impi_native_hybrid.mic
```

```
num_nodes - число узлов для задачи (содержащих по 2
сопроцессора Xeon Phi);
-x - число процессов на каждом x86-узле
-z - число процессов на каждом mic-узле
-c - исполнимый файл для x86
-m - исполнимый файл для mic
```

Данная команда может быть использована и для запуска приложений не использующих MPI. Для этого перед постановкой в очередь задачи необходимо указать число используемых Xeon Phi на одном узле в переменной.

```
MIC_NUM_PER_HOST=1 - по умолчанию значение переменной
равно 2

OMP_NUM_THREADS -задает число OpenMP нитей для x86-
узлов

MIC_OMP_NUM_THREADS - задает число OpenMP нитей для
mic-узлов

MPIEXEC_FLAGS_HOST - задает флаги, которые будут
переданы mpiexec в процессы для x86-узлов, например
MPIEXEC_FLAGS_HOST="-env VAR VALUE"

MPIEXEC_FLAGS_MIC - задает флаги, которые будут
переданы mpiexec в процессы для mic-узлов, например
MPIEXEC_FLAGS_MIC="-envlist VAR1,VAR2"
```

Пример скрипта для запуска в пакетном режиме:

```
[user@master test]$ sbatch batch.mic
[user@master test]$ cat batch.mic
#!/bin/bash
#SBATCH -J TestJobMIC
#SBATCH -N 6
#SBATCH -p phi
#SBATCH --time=30

export MIC_NUM_PER_HOST=2
export OMP_NUM_THREADS=1
export MIC_OMP_NUM_THREADS=1

mpirun.mic -v -x 16 -c ./test -z 4 -m ./test.MIC
```

Более подробная информация и пример скриптов для sbatch может быть получена с помощью команды

```
mpirun.mic -h
```

# 7 Список литературы

1. Slurm Workload Manager // SLURM URL: https://slurm.schedmd.com (дата обращения: 02.04.2017).