Jet Cluster User's Guide

Конфигурация кластера

Аппаратное обеспечение

18 вычислительных узлов (8 ядер на узел, всего 144 ядра)

- 2 x Intel Quad Core Xeon E5420 (2.5 GHz)
- RAM: 8 GB (4 x 2GB PC-5300)
- HDD: SATAII Seagate Barracuda 500 GiB
- Сеть Gigabit Ethernet (1 x Intel PRO/1000 MT Server Adapter, Dual Port Intel PRO/1000 EB (80003ES2LAN))

Программное обеспечение

- GNU/Linux Fedora 20 x86_64
- C/C++/Fortran: GCC 4.8.2, clang 3.3, gfortran
- Java: Oracle JDK 1.7.0 (amd64)
- MPI: MPICH2 3.0.4
- Batch system: TORQUE 4.6.2.1
- Apache Hadoop 2.3.0 (HDFS ~ 6 TiB, ~ 380 GiB per DataNode)

Доступ к кластеру

Для работы с кластером необходимо зайти по SSH на головную машину `jet.cpct.sibsutis.ru`, указав логин и пароль. Под операционной системой Microsoft Windows рекомендуется использовать клиент PuTTY.

```
$ ssh username@jet.cpct.sibsutis.ru
```

Дисковая квота на домашний каталог пользователя: 1 GiB (монтируется по NFS через Gigabit Ethernet на вычислительные узлы кластера). Проверить текущую квоту можно командой quota.

Для хранения временных файлов на жестком диске каждого узла создана директория `/scratch` (этот каталог также используется как хранилище распределенной файловой системы HDFS).

Для копирования файлов на кластер (или с кластера) используйте команду scp.

Изменить пароль можно командой passwd.

Система управления ресурсами TORQUE

На кластере установлена система управления ресурсами TORQUE. Этот пакет обеспечивает выполнение задач пользователей в пакетном режиме (batch system). Для работы с системой пользователь готовит job-файл (задание), в котором указывает число узлов необходимых для выполнения программы и команды для её запуска на выделенной подсистеме.

Далее командой qsub пользователь ставит задачу в очередь. Задаче присваивается идентификатор (jobid). Как только появятся доступные вычислительные ресурсы планировщик системы TORQUE запустит задачу на выполнение. Просмотреть состояние очередей можно командой qstat. Удалить задачу можно командой qdel <jobid>. Состояние узлов выводится по команде pbsnodes.

Пользователям доступно 2 очереди: debug и release. В очереди debug максимальное время выполнения задачи — 20 минут (это очередь используется по умолчанию), в очереди release — 7 суток. Для постановки задачи в очередь release необходимо указать в job-файле строку «#PBS -q release».

После завершения работы программы будет создан файл, содержащий вывод программы (stderr, stdout, cout, cerr).

Компиляция и запуск OpenMP-программ

Для компиляции OpenMP-программы используйте ключ -fopenmp компилятора GCC:

```
$ gcc -fopenmp ./myprog.c
```

Для запуска OpenMP-программы необходимо сформировать job-файл и поставить его в очередь системы управления ресурсами TORQUE. Рекомендуется захватывать все процессорные ядра одного узла (#PBS -l nodes=1:ppn=8), чтобы избежать запуска на нем задач нескольких пользователей.

Количество потоков в параллельных регионах OpenMP-программы регулируется значением переменной среды окружения OMP_NUM_THREADS. Ниже приведен пример файла task.job для запуска OpenMP-программы myprog:

```
#PBS -N MyOpenMP
#PBS -l nodes=1:ppn=8
#PBS -j oe
cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=6
./myprog
```

Постановка задачи в очередь выполняется командой qsub:

```
$ qsub ./task.job
1278
```

После завершения работы программы будет создан файл MyOpenMP.o1278, содержащий вывод программы (stderr, stdout, cout, cerr).

Компиляция и запуск MPI-программ

Для компиляция MPI-программ предназначены команды mpicc (C), mpicxx (C++) и mpif90 (Fortran):

```
$ mpicc ./myprog.c
$ mpicxx ./myprog.cpp
$ mpif90 ./myprog.f90
```

Для запуска MPI-программы необходимо сформировать job-файл и поставить его в очередь системы управления ресурсами TORQUE. В job-файле следует указать требуемое число вычислительных узлов (<=18) и число процессорных ядер используемых на каждом узле (#Pbs -l nodes=<nnodes>:ppn=<nprocs>).

Ниже приведен пример файла task.job для запуска MPI-программы myprog с 50 процессами (10 узлов по 5 процессорных ядер):

```
#PBS -N mpitask
#PBS -l nodes=10:ppn=5
#PBS -j oe

cd $PBS_O_WORKDIR

mpiexec ./test
```

Постановка задачи в очередь выполняется командой qsub:

```
$ qsub ./task.job
1279
```

После завершения работы программы будет создан файл mpitask.o1279, содержащий вывод программы (stderr, stdout, cout, cerr).

Работы с Apache Hadoop

Доступ к Web-интерфейсу Apache Hadoop

На головной машине кластера (jet.cpct.sibsutis.ru) закрыты все внешние порты, кроме SSH. Для доступа к Web-интерфейсам необходимо использовать SOCKS-прокси через SSH-туннель.

Для создания ssh-туннеля запустите на своей машине команду:

GNU/Linux:

```
$ ssh -D 7777 username@jet.cpct.sibsutis.ru
```

Microsoft Windows:

```
c:\> putty.exe -D 7777 username@jet.cpct.sibsutis.ru
```

После того как ssh-туннель создан необходимо настроить SOCKS proxy в браузере (host=localhost, port=7777).

Web-интерфейсы Hadoop будут доступны через браузер по следующим адресам:

- Состояние файловой системы HDFS (NameNode):
  http://192.168.1.254:50070
- Вычислительные ресурсы (ResourceManager):
  http://192.168.1.254:8088
- История выполнения MapReduce-программ (Job History server):
  http://192.168.1.254:19888

Не забудьте отключать SOCKS-proxy в настройках браузера, когда не работаете с кластером.

Первоначальная настройка рабочего окружения

В начале работы с Apache Hadoop необходим настроить переменные среды окружения (это необходимо выполнить один раз). В своем домашнем каталоге на кластере добавьте в конец файла `~/.bashrc` следующую строку:

```
source /opt/etc/hadoop-vars.sh
```

Отключитесь от кластера и зайдите на головную машину снова.

Работа с файловой системой HDFS

Распределенная файловая система HDFS (Hadoop Distributed File System) предназначена для хранения больших массивов данных. Она же используется для записи полученных в ходе выполнения MapReduce-программ результатов.

Каждый файл в рамках HDFS разбивается на блоки фиксированного размера (128 MiB) и распределяется по вычислительным узлам. Для обеспечения отказоустойчивости каждый блок реплицируется на несколько вычислительных узлов (в текущей конфигурации на 3 узла). Таким образом имеется возможность хранить файлы с размером превышающим размер жесткого диска одного узла.

HDFS можно использовать для хранения рабочих данных большого объема, вместо хранения их в домашней директории (NFS). Для просмотра содержимого HDFS рекомендуется использовать Web-интерфейс.

Для работы с HDFS используется набор команд, похожих по смыслу на файловые команды GNU/Linux.

Список всех команд:

```
$ hdfs dfs -help
```

Выведем на экран содержимое корневого каталога HDFS:

```
$ hdfs dfs -ls /
```

Создадим в домашнем каталоге папку bigdata:

```
$ hdfs dfs -mkdir ./bigdata
```

Скопируем файл с локальной файловой системы в HDFS-каталог bigdata:

```
$ hdfs dfs -put /home/pub/etwiki.xml ./bigdata
```

Скопируем еще один файл с локальной файловой системы в свой HDFS-каталог:

```
$ echo Hello > test.txt
$ hdfs dfs -put ./test.txt ./
```

Выведем содержимое файла test.txt на экран:

```
hdfs dfs -cat ./test.txt
```

Скопируем файл из HDFS-каталога в локальную файловую систему:

```
hdfs dfs -get ./test.txt ./mylocal.txt
```

Удалим файлы из HDFS:

```
hdfs dfs -rm ./test.txt
hdfs dfs -rm -r ./bigdata
```

Компиляция и запуск MapReduce-программ (Java API)

Компиляция MapReduce-программ, созданных с использованием Hadoop Java API:

```
$ javac -classpath `hadoop classpath` WordCount.java
$ jar -cvf wordcount.jar .
```

Далее необходимо загрузить входные данные программы в файловую систему HDFS:

```
$ hdfs dfs -mkdir ./wordcount
$ hdfs dfs -put ~/data.txt ./wordcount/input
```

Запускаем задание (jar-файл):

```
$ hadoop jar ./wordcount.jar pdccourse.lecture6.WordCount \
         -D mapreduce.job.reduces=1 \
         ./wordcount/input ./wordcount/output
```

Выгружаем данные из HDFS в локальную файловую систему

```
$ hdfs dfs -get ./wordcount/output/part* ./result
```

Удаляем результаты (для корректного повторного запуска)

```
$ hdfs dfs -rm -r ./wordcount/output
```

Если данные больше не нужны удаляйте их из HDFS.

Запуск Hadoop Streaming-задач

Процесс работы с задачами Hadoop Streaming подобен описанному выше процессу запуска MapReduce-задач на Java API.

Запуск Hadoop Streamin-задания (для mapper.py и reducer.py):

```
$ hadoop jar \
        /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.3.0.jar \
        -file ./mapper.py -mapper ./mapper.py -file ./reducer.py \
        -combiner ./reducer.py -reducer ./reducer.py \
        -input ./wordcount/input -output ./wordcount/output \
        -numReduceTasks 1
```

Запуск Hadoop Pipes-задач

Собираем нашу C++ программу (wordcount.cpp) с библиотекам Hadoop Pipes:

```
$ g++ -std=c++11 -O2 -I/opt/hadoop/include -c wordcount.cpp \
      -o wordcount.o
$ g++ -o wordcount wordcount.o -L/opt/hadoop/lib/x86_64 \
      -lhadooppipes -lhadooputils -lcrypto -lpthread
```

Исполняемый файл программы (wordcount) необходимо скопировать в файловую систему HDFS:

```
$ hdfs dfs -put ./wordcount ./wordcount/bin
```

Запускаем задание:

```
$ hadoop pipes -D hadoop.pipes.java.recordreader=true \
               -D hadoop.pipes.java.recordwriter=true \
               -program ./wordcount/bin/wordcount -reduces 1 \
               -input ./wordcount/input -output ./wordcount/output
```

Работа с очередями заданий

На кластере настроен планировщик Apache Hadoop CapacityScheduler. По умолчанию всем пользователям разрешено ставить задачи в очередь default.

Список доступных очередей задач:

```
$ hadoop queue -list
```

Список очередей со списком ваших прав доступа к ним:

```
$ hadoop queue -showacls
```

Список заданий (jobs) в очереди default:

```
$ hadoop queue -info default -showJobs
```

Состояние очередей удобнее отслеживать через Web-интерфейс (см. выше).