УЧРЕЖДЕНИЕ РОССИЙСКОЙ АКАДЕМИИ НАУК
ИНСТИТУТ СИСТЕМ ОБРАБОТКИ ИЗОБРАЖЕНИЙ РАН

САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ
имени академика С. П. КОРОЛЁВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Н.Л. Казанский, С.Б. Попов, П.Г. Серафимович

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА НА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ СИСТЕМАХ

Учебное пособие

САМАРА
2010

УЧРЕЖДЕНИЕ РОССИЙСКОЙ АКАДЕМИИ НАУК
ИНСТИТУТ СИСТЕМ ОБРАБОТКИ ИЗОБРАЖЕНИЙ РАН

САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С. П. КОРОЛЁВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Н.Л. Казанский, С.Б. Попов, П.Г. Серафимович

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА НА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ СИСТЕМАХ

Учебное пособие

САМАРА
2010

УДК 681.324.006.3
Н.Л. Казанский, С.Б. Попов, П.Г. Серафимович Организация вычислительного эксперимента на высокопроизводительных системах. ИСОИ РАН. Самара. 2010, 120 с.
ISBN **5-93673-020-3**

В настоящее время основной операционной системой высокопроизводительных вычислительных систем является Linux.

В данном руководстве даны базовые приемы работы на вычислительном Linux-кластере. Описываются правила доступа к вычислительным кластерам Самарского государственного аэрокосмического университета имени академика С.П. Королева (национального исследовательского университета) (СГАУ). Сравнивается использование сетевой и локальной файловых систем. Обсуждаются вопросы пакетного управления заданиями пользователей. Рассмотрены основные приемы мониторинга работы кластера.

Учебное пособие предназначено для обеспечения учебного процесса базовой кафедры высокопроизводительных вычислений и ориентировано на массовую подготовку студентов в рамках программ подготовки бакалавров, магистров, слушателей ФПКП и других категорий, изучающих технологии параллельных вычислений. Учебное пособие может быть полезно также при проведении научных исследований, выполнении курсовых и дипломных проектов по физико-математическим и техническим направлениям подготовки.

Пособие предназначено для студентов специальностей и направлений «Прикладная математика и информатика», «Прикладная математика и физика».

Печатается по решению ученого совета ИСОИ РАН.
Рецензенты: д.ф.-м.н., профессор, Радченко В.П.,
д.ф.-м.н., доцент, Головашкин Д.Л.

ISBN 5-93673-020-3

ОГЛАВЛЕНИЕ

1 Основные приемы работы пользователя на кластере

1.1 Способы доступа пользователя на кластер

1.1.1 Удаленный доступ на кластер для компиляции и запуска расчетных программ пользователя

Пользователи операционной систем Linux могут воспользоваться стандартным ssh-клиентом. Пользователям Microsoft Windows рекомендуется использовать программу PuTTy (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)

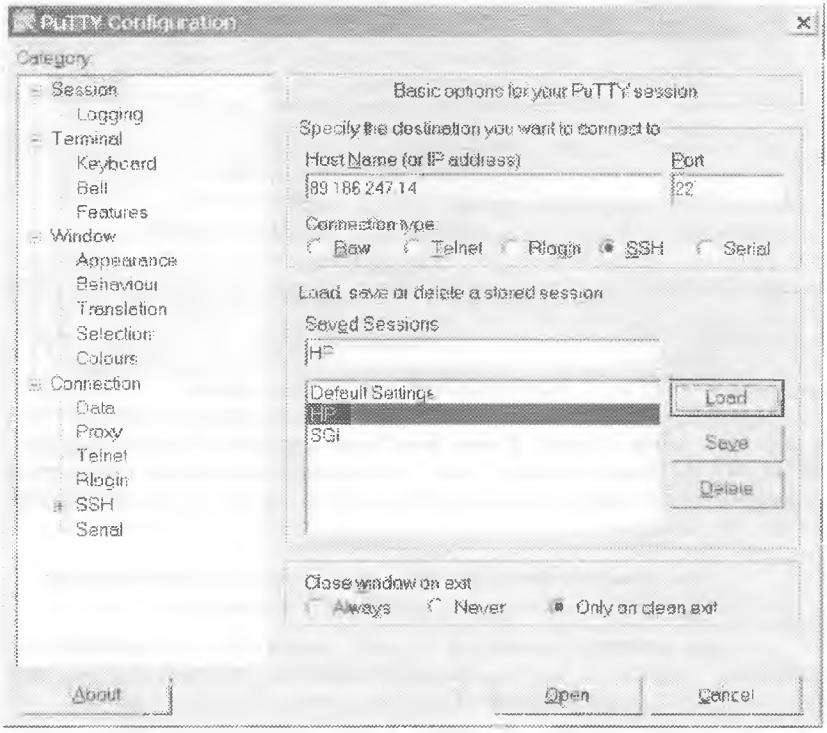

Рис. 1. Конфигурация программы PuTTy

При первом запуске программы в окне HostName набрать IP адрес кластера – 89.186.247.14. В поле Saved Sessions ввести любое удобное название (на рис.1 выбрано название «HP»).

В поле Category – Window – Translation – Character set translation on received data выбрать UTF-8 (см. рис. 2). Вернуться в окно Category – Session. Нажать кнопку “Save”. Под именем HP будут сохранены настройки.

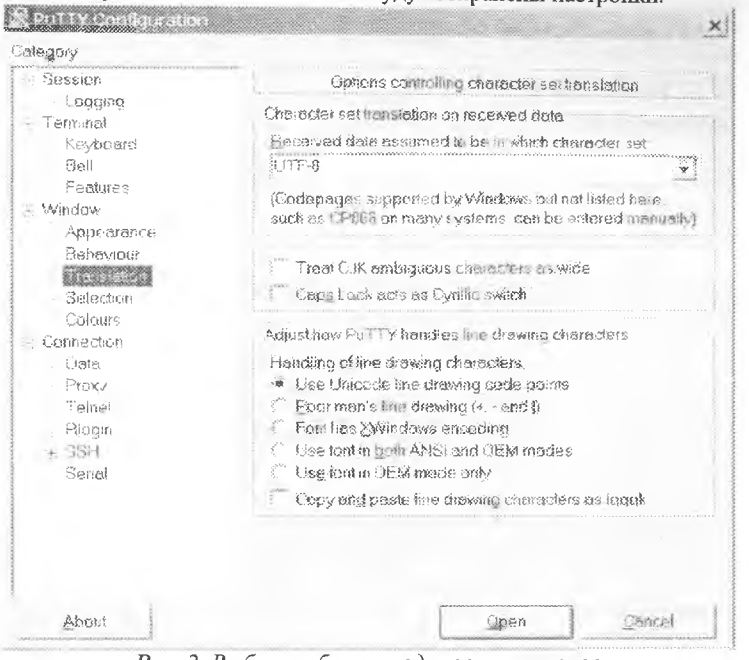

Рис. 2. Выбор таблицы кодировки символов

При последующих запусках PuTTy в окне конфигурации выбрать «HP» и щелкнуть кнопку “Load”. В окне Host Name появится IP адрес кластера – 89.186.247.14. Нажать кнопку “Open”. Произойдет соединение с кластером, откроется окно терминала кластера. В этом окне сначала ввести имя пользователя, затем пароль.

1.1.2 Удаленный доступ на кластер для копирования файлов между персональным компьютером пользователя и кластером

Сетевой файловый менеджер WinSCP может быть использован для файловых операций с удаленными и локальными файлами. Скачать программу WinSCP можно по адресу http://winscp.net.

При первом запуске программы WinSCP необходимо ввести в поле Host name IP адрес кластера – 91.222.131.14, а в поле User name имя пользователя (см. рис. 3). Поле Password можно не заполнять, его вы будете вводить при каждом следующем соединении. Введенные данные сохраняются кнопкой “Save”.

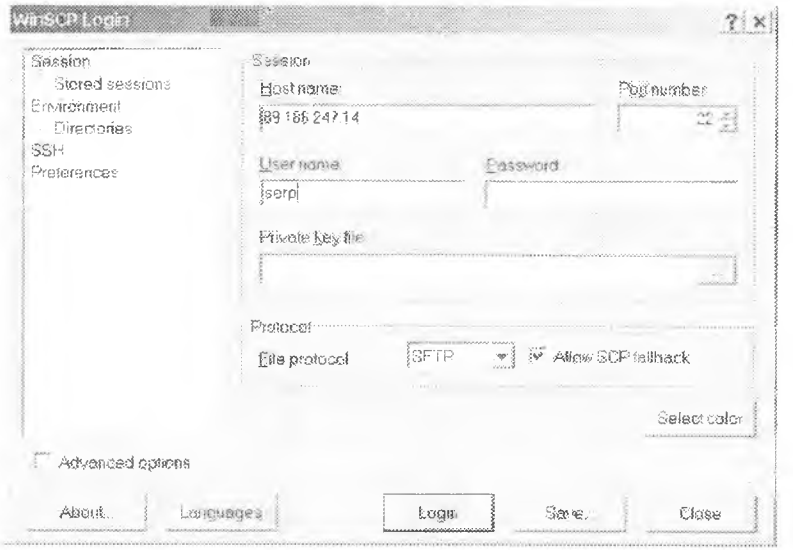

Рис. 3. Стартовое окно программы WinSCP при первом запуске

При последующих запусках программы окно WinSCP Login будет с заполненными личными данными (рис. 4).

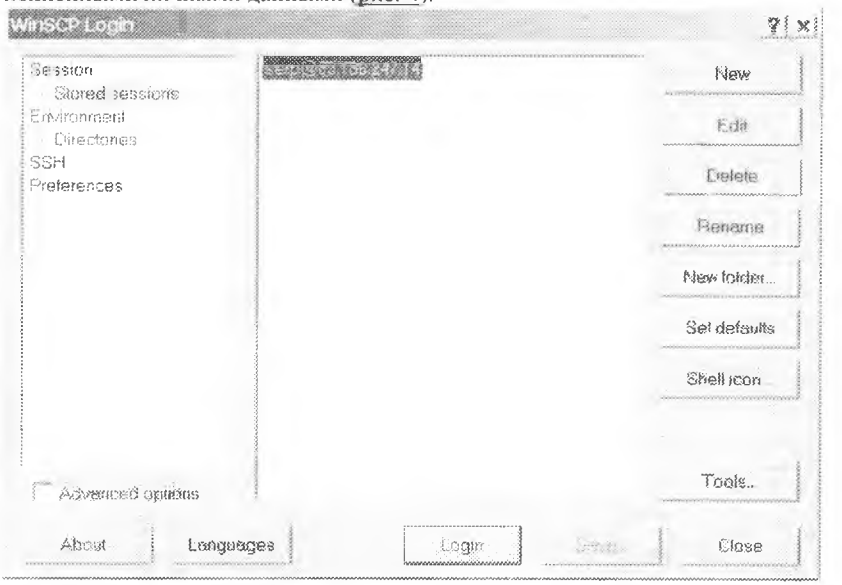

Рис. 4. Окно соединения

Соединение начинается выбором пункта “Login”. В окне Server prompt надо ввести свой пароль (рис. 5).

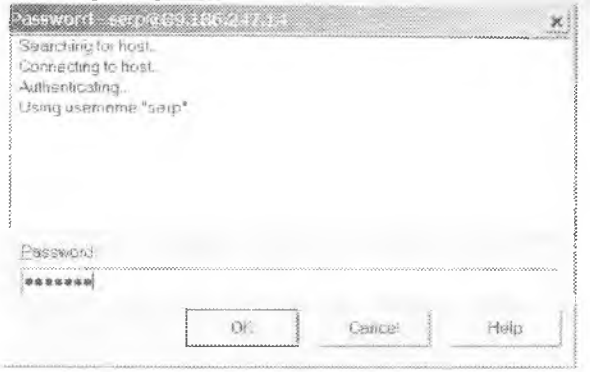

Рис. 5. Окно ввода пароля

Для соединения с удаленной ЭВМ необходимо время. Поэтому окно программы WinSCP появится с некоторой задержкой. В зависимости от варианта, выбранного при установке программы, откроется окно, по внешнему виду сходное с Windows Explorer либо двухпанельное окно в стиле Total Commander. В этом окне будет отображена файловая система кластера, и вы можете копировать файлы на кластер и обратно также, как это делается в Windows (рис. 6).

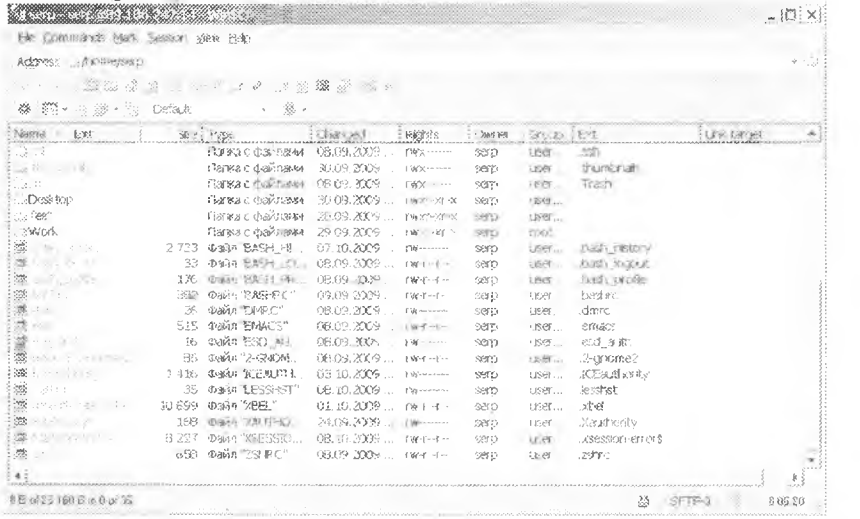

Рис. 6. Рабочее окно программы WinSCP

Поддерживаются операции Drag and Drop. Кроме того, программа WinSCP позволяет проводить другие операции с файлами: редактирование, переименование, удаление, изменение прав доступа и прочее.

Завершение работы с программой и прекращение связи проводится клавишей F10 или выбором в меню пункта Commands/Quit. Требуется подтвердить окончание работы в окне завершения.

1.2 Настроить окружение выполнения MPI программы

Настроить окружение выполнения MPI программы можно с использованием команд утилиты Switcher.

1.2.1 Посмотреть текущее состояние

[tester@master ~]$ **switcher mpi --show**

user:default=lam-7.1.4 user:exists=1

1.2.2 Посмотреть список возможных настроек

[tester@master ~]$ **switcher mpi --list**

openmpi-1.2.4

mpich-ch_p4-gcc-1.2.7

lam-7.1.4

1.2.3 Установить окружение выполнения MPI программы

В примере будет установлено окружение mpich-ch_p4-gcc-1.2.7

[tester@master ~]$ **switcher mpi --add-attr default mpich-ch_p4-gcc-1.2.7**

Warning: mpi:default already has a value:

lam-7.1.4

Replace old attribute value (y/N)? y

Attribute successfully set; new attribute setting will be effective for future shells

1.2.4 Перезагрузить программную оболочку

[tester@master ~]$ **bash**

1.2.5 Проверить установленные настройки

[tester@master ~]$ **switcher mpi --show**

user:default=mpich-ch_p4-gcc-1.2.7

user:exists=1

и/или

[tester@master ~]$ **which mpirun**

/opt/mpich/1.2.7/ch_p4/gcc/bin/mpirun

и/или

[tester@master ~]$ **cexec which mpirun**

**********oscar_cluster **********

--------- *n1*---------

/opt/mpich/1.2.7/ch_p4/gcc/bin/mpirun

--------- *n2*---------

/opt/mpich/1.2.7/ch_p4/gcc/bin/mpirun

--------- *n3*---------

/opt/mpich/1.2.7/ch_p4/gcc/bin/mpirun

--------- *n4*---------

/opt/mpich/1.2.7/ch_p4/gcc/bin/mpirun

1.3 Подготовка исполняемого файла MPI приложения

1.3.1 Редактирование текста программ пользователя

Процесс разработки программного обеспечения, независимо от размеров и предназначения программы содержит этап редактирования исходных текстов программы. Конечно, вы можете изменять свои программы на рабочем компьютере, а потом копировать их на кластер. Также вы можете редактировать файлы прямо на кластере. Для этого на кластере имеется несколько текстовых редакторов разной степени удобства: vim, ed, emacs, joe. Наиболее простым для освоения, на наш взгляд, является редактор, встроенный в оболочку MidnightCommander. (рис. 7)

Для запуска этой программы используется команда mc.

Midnight Commander – это файловый менеджер с текстовым интерфейсом. Его предназначение – упростить основные действия пользователя, связанные с управлением файлами. Принцип работы Midnight Commander такой же, как и у Far Manager, TotalCommander, или Norton Commander. Экран состоит из двух панелей, в которых отображается список файлов и каталогов в выбранных каталогах, и пользователь может выполнять некоторый набор действий над этими файлами. В нижней части экрана расположена командная строка и панель горячих клавиш F1-F10. Можно вызвать верхнее меню, нажав клавишу F9. Одна из панелей всегда является активной, а в ней курсор установлен на активный файл. Пользователь может выполнять действия либо с активным файлом или каталогом, либо групповые операции со всеми объектами активной панели. Также доступны некоторые общие опе-

рации: поиск файлов, помощь по работе с MC, выполнение команд операционной системы и т.д.

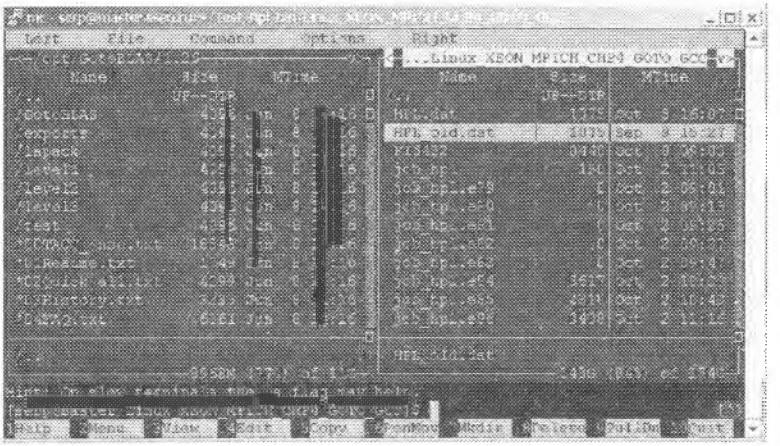

Рис. 7 Окно оболочки MidnightCommander

Для редактирования файла необходимо клавишами управления курсором выбрать нужный файл, и нажать клавишу F4. Запустится редактор текстовых файлов, выйти из которого можно, нажав клавишу F10 либо Esc. Чтобы сохранить изменения, необходимо нажать клавишу F2, либо выбрать нужный вариант при выходе из редактора.

Для создания нового файла нужно нажать Shift+F4 (при нажатой клавише Shift нажмите клавишу F4). Откроется окно редактора, и при сохранении изменений программа предложит ввести имя сохраняемого файла.

1.3.2 Компиляция программ пользователя

Для компиляции и сборки параллельных программ используются следующие утилиты:

- *mpicc* – для программ написанных на языке программирования C;
- *mpixx (mpiCC)* – для программ написанных на языке программирования C++;
- *mpif77 и mpif90* – для программ написанных на языке программирования Fortran.

Синтаксис данных утилит во многом похож на синтаксис компилятора ***gcc***, более полная информация о синтаксисе доступна по команде **`имя_утилиты --help`** (напр., **`mpif77 -help`**).

Например, для обработки программы myprog.c можно выполнить команду

```
mpicc -o myprog -lm myprog.c
```

Ключ `-o` указывает, что результат компилирования должен быть помещен в файл `myprog`. Если мы просто выполним команду

```
mpicc  -lm myprog.c
```

то в текущем каталоге появится исполняемый файл `a.out`.

Если программа представлена не одним файлом, а несколькими исходными и заголовочными файлами, то для сборки можно использовать Makefile или shell-скрипт.

1.4 Запустить MPI приложение на кластере

Выделение ресурсов и запуск приложений на кластере обеспечивает система пакетной обработки заданий Torque и менеджер ресурсов MAUI. Поэтому для того чтобы запустить MPI приложение на кластере необходимо выполнить следующее:

- подготовить PBS-задание
- поставить PBS-задание в очередь на выполнение.

1.4.1 Подготовка PBS-задания

PBS–задание это некоторый скрип написанный на языке командного интерпретатора, которой содержит как директивы для самой системы пакетной обработки на выделение ресурсов, так и директивы для запуска задачи пользователя.

Пример. Для запуска MPI программы PBS- задание может быть оформлено следующим образом:

```
#PBS -l walltime=00:30:00,nodes=4:ppn=8
#PBS -q workq@master
#PBS -N job_name
#PBS -o /home/tester/out
#PBS -e /home/tester/err
#!/bin/sh
cd  /home/tester/hpl/bin/Linux_XEON_MPICH_CHP4_GOTO_GCC/
mpirun -np 32 -machinefile $PBS_NODEFILE ./xhpl
```

Значение параметров PBS задания смотрите в соответвующем разделе данного пособия.

1.4.2 Постановка PBS-задания в очередь на выполнение

После того как PBS задание готово, его необходимо поставить в очередь на выполнение командой

qsub [имя PBS-задания]

[tester@master ~]$ **qsub job_hpl**

1.4.3 Мониторинг запущенного задания

Мониторинг очереди заданий может быть выполнен с использованием терминальных команд системы пакетной обработки Torque или менеджера ресурсов MAUI.

1.4.4 Состояние очереди заданий

[tester@master ~]$ **qstat**

1.4.5 Полная инфомация по заданию

qstat -f [Job ID]

1.4.6 Информация о состоянии очереди заданий от менеджера ресурсов

[tester@master ~]$ **/opt/maui/bin/showq**

1.4.7 Полная информация по узлам кластера

[tester@master ~]$ **pbsnodes**

1.5 Пример тестовой программы MPI – расчет числа пи.

1.5.1 Текст программы расчета числа пи для MPI

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a )
{
   return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
   int done = 0, n, myid, numprocs, i;
   double PI25DT = 3.14159265358979323846264 3;
   double mypi, pi, h, sum, x;
   double startwtime = 0.0, endwtime;
   int  namelen;
   char processor_name[MPI_MAX_PROCESSOR_NAME];

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Get_processor_name(processor_name,&namelen);
```

```
    fprintf(stderr,"Process %d on %s\n",
    myid, processor_name);

    n = 0;
    while (!done)
    {
      if (myid == 0)
      {
/*
        printf("Enter the number of intervals: (0 quits) ");
        scanf("%d",&n);
*/
    if (n==0) n=100; else n=0;

    startwtime = MPI_Wtime();
      }
      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
      if (n == 0)
        done = 1;
      else
      {
        h   = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs)
        {
          x = h * ((double)i - 0.5);
          sum += f(x);
        }
        mypi = h * sum;

MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
           MPI_COMM_WORLD);

        if (myid == 0)
    {
          printf("pi is approximately %.16f, Error is %.16f\n",
              pi, fabs(pi - PI25DT));
        endwtime = MPI_Wtime();
        printf("wall clock time = %f\n",
            endwtime-startwtime);
    }
      }
    }
    MPI_Finalize();

    return 0;
}
```

1.5.2 Компиляция программы расчета числа пи для MPI

>/usr/mpi/intel/openmpi-1.2.8/bin/mpicc -lm cpi.c -o cpi

1.5.3 Запуск программы расчета числа пи для MPI

Здесь переменная окружения **$PBS_O_WORKDIR** создается в момент запуска PBS-скрипта и содержит список доступных вычислительных узлов.

```
#PBS -l walltime=00:03:00,nodes=2:ppn=8
cd $PBS_O_WORKDIR
mpirun -np 16 -hostfile $PBS_NODEFILE ./cpi
```

1.6 Пример тестовой программы OpenMP – HelloWorld.

1.6.1 Текст программы HelloWorld для OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

int nthreads, tid;

/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
 {

 /* Obtain thread number */
 tid = omp_get_thread_num();
 printf("Hello World from thread = %d\n", tid);

 /* Only master thread does this */
 if (tid == 0)
  {
  nthreads = omp_get_num_threads();
  printf("Number of threads = %d\n", nthreads);
  }
 }
}
```

1.6.2 Компиляция программы HelloWorld для OpenMP

```
>icc -openmp helloworld.c -o helloworld.out
```

1.6.2 Запуск программы HelloWorld для OpenMP

Здесь переменная окружения **$PBS_O_WORKDIR** создается в момент запуска PBS-скрипта и содержит список доступных вычислительных узлов. Переменная **$NUM_PROCS** содержит количество процессов на которых будет работать задача.

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=1:ppn=8

# Script for running OpenMP sample program on 8 processors on cluster

cd $PBS_O_WORKDIR

NUM_PROCS=`/bin/awk 'END {print NR}' $PBS_NODEFILE`

export OMP_NUM_THREADS=$NUM_PROCS

./helloworld.out
```

2 Использование сетевой файловой системы

2.1 Причины использования сетевой файловой системы

Одной из особенностей запуска MPI-программ является необходимость наличия копий программы на всех узлах кластера, на которых она исполняется [1]. Например, если ваша программа *myprog* расположена в каталоге */home/mpiuser/program1*, то на всех узлах кластера должен присутствовать этот каталог и в него должна быть помещена ваша программа.

Это условие порождает необходимость каким-либо образом распределить копии исполняемого модуля программы между узлами кластера. Аналогичное требование относится и к хранимым на диске данным, которые программа будет использовать.

Существуют различные механизмы, позволяющие выполнять подобное распределение. В большинстве случаев это разнообразные скрипты, осуществляющие синхронизацию каталогов на узлах кластера с помощью команд *scp* или *rsync*. Подобные способы синхронизации имеют свои недостатки. Например, в случае, когда различные копии программы должны обращаться к одним и тем же данным, хранимым на диске, и изменять их определенным образом, возникает проблема, связанная с необходимостью постоянной синхронизации файлов на узлах кластера. Другая проблема возникает при использовании в качестве узлов кластера бездисковых станций. В этом случае вся файловая система таких узлов хранится в оперативной памяти компьютера и чем больше мы закачиваем данных на такой узел, тем меньше остается памяти для выполнения программы.

Для избавления от подобных проблем используются сетевые файловые системы. Существует большое количество реализаций таких систем, как платных, так и распространяемых под лицензией GPL. Мы будем рассматривать сетевую файловую систему NFS, имеющуюся в любом Linux-дистрибутиве общего назначения. Файловая система NFS - это аналог того, что продуктах Майкрософт известно под названием windows share.

Сетевая файловая система NFS состоит из двух компонентов: сервера и клиента. Сервер осуществляет сетевой доступ к каталогам базовой файловой системы на основе определенных правил разграничения доступа. Клиент используется для подключения к расшаренным ресурсам. Установку NFS мы рассматривать не будем, поскольку она достаточно тривиальна. Вы либо в процессе инсталляции операционной системы указываете необходимость установки NFS, либо вручную устанавливаете два пакета rpm: nsf-server и nfs-clients. Далее мы рассмотрим процесс конфигурации сетевой файловой системы.

2.2 Конфигурация сетевой файловой системы

2.2.1 Конфигурация сервера

Для обеспечения сетевого доступа узлов кластера к расшаренным на сервере кластера ресурсам необходимо вначале разрешить подключение nfs-клиентам к nfs-серверу. Далее будем предполагать, что узлы кластера имеют ip-адреса в диапазоне 192.168.1.2-192.168.1.254. Консоль кластера, к каталогам файловой системы которой мы будем подключаться через NFS, имеет ip-адрес 192.618.1.1. Для разрешения сетевого доступа к NFS с этих адресов мы в файле **/etc/hosts.allow** прописываем следующую строчку:

portmap: 192.168.1.

Отметим, очка в конце строки обязательна. Далее мы должны определить к какому каталогу мы разрешаем сетевой доступ. То есть, какой каталог расшариваем. К примеру, мы хотим обеспечить узлам кластера доступ в каталог /home/mpiuser/data-and-progs. Для этого в файле **/etc/exports** прописываем строку:

/home/mpiuser/data-and-progs
192.168.1.0/255.255.255.0(rw,no_root_squash)

На этом настройка серверной части закончена. Чтобы изменения вступили в силу необходимо перезапустить службу NFS с помощью команды "**service portmap restart**".

2.2.2 Конфигурация клиентов.

Переходим от сервера кластера (консоли кластера) к остальным узлам. Все что будет описано ниже необходимо выполнить на каждом компьютере кластера кроме консольного.

Для подключения любой файловой системы (дискеты, раздела диска, сетевого ресурса) используется команда *mount*, если подключение происходит вручную, или запись в файле */etc/fstab*, если подключение происходит в момент загрузки системы. Нас будет интересовать последний случай.

Для обеспечения запуска mpi-программ нам нужно, чтобы содержимое каталога /home/mpiuser/data-and-progs на узлах кластера совпадало с содержимым этого же каталога на консоли кластера. Для этого мы должны в домашнем каталоге пользователя mpiuser (/home/mpiuser) создать пустой каталог data-and-progs. После чего прописать в файле **/etc/fstab** следующую строку:

192.168.1.1:/home/mpiuser/data-and-progs/home/mpiuser/data-and-progs nfs rw 0 0

Чтобы удаленный (сетевой) каталог монтировался автоматически при загрузке узла кластера, сервис клиента NFS должен запускаться в процедуре начальной загрузки.

На этом установка сетевой кластерной файловой системы завершена. При включении кластера, консоль кластера должна быть загружена до того, как вы начнете включать остальные узлы.

2.2.3 Разделение сетей

Поскольку сеть является самым узким местом кластера, желательно организовать работу так, чтобы операции межпроцессорной пересылки данных не пересекались с файловыми операциями NFS. Для этого необходимо компьютеры кластера оснастить дополнительными сетевыми картами, объединенными в отдельную сеть, физически не пересекающуюся с первыми картами. То есть использовать дополнительный набор хабов или свитчей. Сетевой интерфейс на вторых картах должен иметь ip-адреса, из другой, отличной от первого интерфейса сети. Например, на сервере кластера сетевая карта, через которую будет осуществляться доступ к расшаренному каталогу, имеет адрес 192.168.1.1, а карта, через которую будет происходить межпроцессорное взаимодействие, имеет адрес 192.168.2.1.

Таким образом, NFS мы настраиваем так, как это было описано выше, а при конфигурировании MPI, список узлов кластера составляем из ip-адресов 192.168.2.*.

Естественно, разделение сетей не необходимо, но желательно.

Важное замечание. *Файлы, хранящиеся на диске, в условиях параллельной задачи, выполняемой на кластере, могут понадобиться только для сохранения состояния задачи в контрольных точках. Конечно, дисковые ресурсы можно использовать и для организации виртуальной памяти, подгружая по мере необходимости данные в оперативную память, увеличивая тем самым размер разностной сетки. Однако при наличии кластера, увеличение размера разностной сетки логичнее и эффективнее может быть выполнено посредством использования дополнительных вычислительных узлов кластера. Если же дисковые ресурсы используются только для сохранения контрольных точек и эти контрольные точки расположены не в каждой итерации (а в каждой десятой или сотой), то разделение локальной сети кластера на два независимых сегмента (NFS и сеть межпроцессорного обмена данных) является не обязательной. Вполне можно обойтись всего одним сегментом, используя его и для NFS и для обмена данными. Поскольку NFS будет использоваться достаточно редко, то и отрицательное влияние ее на эффективность кластера будет минимально.*

2.3 Пример: решение уравнения Лапласа

Важным моментом при написании параллельной программы является работа по сохранению данных в файлы или восстановлению данных из них. Примерами необходимости таких действий могут служить задачи загрузки начальных данных в массивы при запуске программы и сохранение промежуточных данных и/или результатов счета.

Если говорить о загрузке начальных данных или сохранении результатов счета, то эта процедура происходит в самом начале работы программы или в самом ее конце, и никаких особенностей или отличий о того, как это может быть сделано в последовательной программе, для параллельной программы нет.

Однако, в том случае, когда речь идет о сохранении промежуточных данных, появляются некоторые нюансы, связанные со структурой параллельной программы. Дело в том, что файловые операции - очень медленные операции. Если сохранение данных происходит в каждом итерационном цикле, то такое действие может существенно снизить быстродействие программы, что кстати верно и в случае обычного последовательного алгоритма. Поэтому при проектировании параллельной программы следует придерживаться двух принципов:

1. Сохранение данных должно происходить не в каждом итерационном цикле, а, скажем, в каждом десятом. В любом случае время между моментами сохранения должно быть много больше времени, требующемся для записи данных в файл.
2. Сохранение данных должно быть реализовано с использованием неблокирующих операторов ввода-вывода, позволяющих совместить по времени файловые операции и вычисления.

Рекомендации, данные на этой странице, проверялись на компиляторе Intel Fortran. Проверьте, поддерживает ли ваш компилятор процедуры асинхронной записи. Если нет – ну что ж, вам придется отказаться от распараллеливания процедур вычисления и сохранения данных. Постарайтесь делать запись на диск как можно реже, чтобы это минимальным образом сказалось на быстродействии вашего кластера.

Посмотрим на конкретном примере, как можно реализовать эти два принципа. Рассмотрим фрагмент программы, которая решает уравнение Лапласа, а именно - блок, описывающий итерационный цикл (листинг 1):

Листинг 1

```
55  c    Подпрограмма вычисления искомой функции
56       subroutine iter(f0,f1,xmax,ymax,df)
57       include 'mpif.h'
58       integer xmax,ymax
59       real*8 f0(xmax,ymax), f1(xmax,ymax)
60       real*8 dt,dx,dy
61       real*8 df
62       integer x,y,n,myid,np,ierr,status(MPI_STATUS_SIZE)
63       common myid,np
64       dt=0.01
65       dx=0.5
66       dy=0.5
67  c    Обмениваемся границами с соседом
68       if ( myid .gt. 0 )
69      x call MPI_SENDRECV(
70      x f0(1,2), xmax, MPI_REAL8, myid-1, 1,
71      x f0(1,1), xmax, MPI_REAL8, myid-1, 1,
72      x MPI_COMM_WORLD, status, ierr)
73       if ( myid .lt. np-1 )
74      x call MPI_SENDRECV(
75      x f0(1,ymax-1), xmax, MPI_REAL8, myid+1, 1,
76      x f0(1,ymax), xmax, MPI_REAL8, myid+1, 1,
77      x MPI_COMM_WORLD, status, ierr)
```

```
 78  c    Вычисляем функцию в ячейках сетки
 79       do x=2,xmax-1
 80         do y=2,ymax-1
 81         f1(x,y)=f0(x,y)+dt*(
 82      x    (f0(x+1,y)-2*f0(x,y)+f0(x-1,y))/(dx*dx)
 83      x    +
 84      x    (f0(x,y+1)-2*f0(x,y)+f0(x,y-1))/(dy*dy)
 85      x    )
 86         end do
 87       end do
 88  c    Копируем границу для следующей итерации
 89       do x=1,xmax
 90         f1(x,1)=f0(x,1)
 91         f1(x,ymax)=f0(x,ymax)
 92       end do
 93       do y=1,ymax
 94         f1(1,y)=f0(1,y)
 95         f1(xmax,y)=f0(xmax,y)
 96       end do
 97  c    Находим максимальную дельту
 98       df=0.0
 99       do x=1,xmax
100         do y=1,ymax
101           if ( df .lt. abs(f0(x,y)-f1(x,y)) ) then
102               df = abs(f0(x,y)-f1(x,y))
103           endif
104         end do
105       end do
106       return
107       end
```

Поскольку мы решили, что сохранение данных будет происходить на каждой 10-й итерации, то есть запись будет происходить в зависимости от номера итерации, то мы должны будем передавать в тело подпрограммы номер текущего цикла. Поэтому вначале изменим описание подпрограммы, добавив дополнительный параметр ее вызова (строка 56).

Рассматриваемая нами подпрограмма по неизменяемым в течение цикла данным массива f0, вычисляет значения массива f1. Поэтому в ее теле мы должны решить, наступил ли десятый цикл, и если да, то инициировать неблокирующую запись массива f0 в файл. Что мы и делаем в строке 77b (листинг 2). Далее мы открываем файл и инициируем процесс записи в него массива данных (строки 77k и 77q).

Кроме того, надо позаботиться о том, чтобы каждый процесс параллельной программы сохранял свою часть данных в отдельный, принадлежащий только ему файл. Вычислением название файла данных мы в строке 77d.

После инициации записи управление передается циклу, в котором и происходят необходимые нам вычисления, причем процессы вычисления и записи идут параллельно. Перед возвратом управления из подпрограммы мы

должны удостовериться, что процедура записи в файл закончена и закрыть файл (строки 105b и 105c).

Теперь посмотрим, как это можно реализовать в программном коде.

Листинг 2

```
55  c    Подпрограмма вычисления искомой функции
56       subroutine iter(f0,f1,xmax,ymax,df,n)
57       include 'mpif.h'
58       integer xmax,ymax
59       real*8 f0(xmax,ymax), f1(xmax,ymax)
60       real*8 dt,dx,dy
61       real*8 df
62       integer x,y,n,myid,np,ierr,status(MPI_STATUS_SIZE)
63       common myid,np
-------------------------------------------------------------------
63a      character(LEN=20):: st,fname
63b      logical ex
-------------------------------------------------------------------
64       dt=0.01
65       dx=0.5
66       dy=0.5
67  c    Обмениваемся границами с соседом
68       if ( myid .gt. 0 )
69      x call MPI_SENDRECV(
70      x f0(1,2), xmax, MPI_REAL8, myid-1, 1,
71      x f0(1,1), xmax, MPI_REAL8, myid-1, 1,
72      x MPI_COMM_WORLD, status, ierr)
73       if ( myid .lt. np-1 )
74      x call MPI_SENDRECV(
75      x f0(1,ymax-1), xmax, MPI_REAL8, myid+1, 1,
76      x f0(1,ymax), xmax, MPI_REAL8, myid+1, 1,
77      x MPI_COMM_WORLD, status, ierr)
-------------------------------------------------------------------
77a c     Если десятый цикл, то сохраняем данные
77b       if ( ((n/10)*10 .EQ. n ) then
77c c       Вычисляем название файла данных
77d         write(fname,'(A,I2.2,A)') "node",myid,".dat"
77e c       Проверяем существует ли файл для записи
77f         inquire(file=fname, exist=ex)
77g c       Если существует, то открываем его для перезаписи,
77h c       если файл не существует - открываем его как новый
77i         write(st,'(A)') 'new'
77j         if ( ex ) write(st,'(A)') 'old'
77k         open(unit=20,
77l     $       file=fname,
77m     $        asynchronous='yes',
77n     $       status=st,
77o     $       form='unformatted')
77p c       Инициируем запись массива f0 в файл
77q         write(20,ID=idvar,asynchronous='yes') f0
77r       endif
-------------------------------------------------------------------
78  c    Вычисляем функцию в ячейках сетки
```

```
 79      do x=2,xmax-1
 80        do y=2,ymax-1
 81        f1(x,y)=f0(x,y)+dt*(
 82     x    (f0(x+1,y)-2*f0(x,y)+f0(x-1,y))/(dx*dx)
 83     x    +
 84     x    (f0(x,y+1)-2*f0(x,y)+f0(x,y-1))/(dy*dy)
 85     x    )
 86        end do
 87      end do
 88  c    Копируем границу для следующей итерации
 89      do x=1,xmax
 90        f1(x,1)=f0(x,1)
 91        f1(x,ymax)=f0(x,ymax)
 92      end do
 93      do y=1,ymax
 94        f1(1,y)=f0(1,y)
 95        f1(xmax,y)=f0(xmax,y)
 96      end do
 97  c    Находим максимальную дельту
 98      df=0.0
 99      do x=1,xmax
100        do y=1,ymax
101          if ( df .lt. abs(f0(x,y)-f1(x,y)) ) then
102              df = abs(f0(x,y)-f1(x,y))
103          endif
104        end do
105      end do
-----------------------------------------------------------------
105a c    Дожидаемся окончания записи и закрываем файл
105b     wait(unit=20,ID=idvar)
105c     close(unit=20)
-----------------------------------------------------------------
106      return
107      end
```

Процедура записи данных, так, как она представлена на этой странице, не вполне оптимальна. Дело в том, что после выхода из подпрограммы до начала следующего цикла выполняется код, который не модифицирует записываемый массив. И код этот выполняется достаточно долго. Но для простоты восприятия мы всю процедуру сохранения данных уложили внутрь рассматриваемой подпрограммы.

В заключение посмотрим, как изменились временные (скоростные) характеристики нашей программы от добавления процедуры записи. В представленной ниже таблице дана длительность работы нашей программы в трех вариантах. Первый вариант – наша исходная программа без записи данных в файл. Второй вариант – программа с записью данных в синхронном режиме, то есть без распараллеливания процедур вычисления и записи. И третий вариант – программа с записью данных на диск в асинхронном режиме, то есть с распараллеливанием счета и записи в файл.

Точность вычислений в программе немного увеличена по сравнению с исходной программой, которую мы рассматривали в пердыдущих разделах, чтобы потребовалось большее число итераций для нахождения решения уравнения. Все варианты программы запускались для работы на трех процессорах. Запись данных велась на расшаренный в NFS ресурс, кроме первого процесса, где запись велась на локальный диск. Кроме того даны результаты, полученные при записи данных на локальный диск (всеми процессами параллельной задачи).

Вариант програмы	Файловая система	Время счета
Записи данных нет	не исп.	1 мин. 26 сек.
Синхронная запись	NFS	2 мин. 46 сек.
Синхронная запись	локальная	1 мин. 37 сек.
Асинхронная запись	NFS	2 мин. 37 сек.
Асинхронная запись	локальная	1 мин. 28 сек.

Резюме. *По результатам тестов можно сделать следующее заключение. Использование NFS сильно замедляет процесс счета. Поэтому стоит подумать об использовании на вычислительных узлах кластера локальных файловых систем для сохранения данных. Однако в этом случае вам придется перед запуском программы и после окончания счета синхронизировать данные на всех узлах, поскольку вообще говоря заранее неизвестно, на каком узле кластера запустится процесс программы с определенным номером. Другими словами файл, созданный процессом 2 на узле node005 в следующий раз может понадобиться процессу 2 на узле node007.*

Следующий вывод, который можно сделать - использование асинхронной записи позволяет увеличить общее быстродействие программы, хотя и не существенно. Однако преимущество асинхронной записи будет тем заметнее, чем больше времени у вас будет занимать вычисление одного цикла итерации.

2.4 Преимущества и недостатки NFS по сравнению с локальной файловой системой

В предыдущих разделах мы рассматривали варианты периодического сохранения данных в файлы на диске. Если задача требует достаточно много процессорного времени, то может возникнуть ситуация, когда вычислительный процесс требуется прервать, и возобновить его через некоторое время с той точки, где он был прерван.

Есть два способа хранения файлов данных: на "расшаренном" в NFS ресурсе и на локальном диске. Оба способа имеют свои преимущества и недостатки. В случае, когда используется виртуальный кластер, то альтернативы нет - может использоваться только NFS. Когда же мы используем стационарный кластер, имеются варианты.

2.3.1 Преимущества NFS

Преимущества NFS заключаются в простоте ее использования и экономии места на диске. Ресурс, экспортированный в NFS, доступен каждому узлу кластера. Нет необходимости копировать исполняемые файлы и файлы данных на все вычислительные узлы. Если данные занимают много места, а обычно бывает именно так - сохраняется состояние всей разностной сетки на какой-то момент времени,- то хранятся они в одном экземпляре на главном узле кластера. Нет необходимости заботиться о доступности нужных для конкретного процесса параллельной программы данных - все доступно всем.

2.3.2 Недостатки NFS

Плоха она тем, что запись данных на сетевой ресурс происходит существенно медленнее по сравнению с записью на локальный диск. Это драматически сказывается на быстродействии всего кластера. Конечно, можно минимизировать такое отрицательное влияние, как можно реже используя процедуры записи. В идеальном случае - сохранять данные только в конце работы программы. Однако это не всегда удобно, поскольку не всегда можно определить, сколько будет работать программа и не придется ли остановить ее раньше срока. Например, придя утром, вы обнаруживаете, что программа еще не закончила работу, а кластер нужно уже освободить. Наша тестовая программа решения уравнения Лапласа, которую мы рассматривали ранее, в модификации, когда запись происходит на каждом втором цикле, в случае записи на локальный диск выполнялась 1 мин. 12 сек., а в случае записи на сетевой ресурс время счета уже составило 4 мин. 53 сек., при общем объеме записываемых данных 1.125 Гб и 30 итерационных циклах.

2.3.3 Преимущества локальной файловой системы

Локальная файловая система хороша тем, что у нее отсутствуют недостатки NFS. Запись данных на современные винчестеры происходит достаточно быстро, позволяя нам по мере необходимости более часто выполнять сохранение данных.

2.3.4 Недостатки локальной файловой системы

Недостаток локальной файловой системы заключен в ее локальности. Необходимым условием запуска параллельной программы является доступность исполняемого файла и файлов данных всем вычислительным узлам кластера, на которых программа будет запущена. Если в отношение исполняемого модуля все не так страшно - MPI имеет средства для автоматического копирования программы на те узлы, где она будет выполняться, то с файлами данных все не так хорошо. У параллельной программы есть особенность - в общем случае мы заранее не знаем, на каких узлах будет выполняться тот или иной процесс. Поэтому на потребуется скопировать все файлы на все узлы кластера, что приводит к многократному дублированию

информации и неоптимальному расходованию дискового пространства. Конечно, мы можем явно указать, на каких узлах кластера будет запущен данный конкретный параллельный процесс, однако это требует от пользователя знания архитектуры и топологии кластера, что не всегда удобно или возможно.

Тем не менее, если проблема свободного дискового пространства не является для вас проблемой, то предпочтительнее использовать локальную файловую систему. Давайте посмотрим, как можно сгладить недостатки локальной ФС, о которых мы упомянули выше.

2.5 Использование локальной файловой системы

Нет необходимости копировать исполняемый модуль на узлы кластера. Команда запуска параллельной программы **mpirun** имеет для этого встроенные средства. Достаточно указать дополнительный ключ **-s** и программа будет автоматически распространена между узлами. Например:

mpirun -hostfile ./../mpi/mpi.hosts -np 3 -s ./diffpw

Благодаря этому ключу mpirun непосредственно перед запуском параллельной програмы скопирует исполняемый файл этой програмы на все вычислительные узлы кластера, на которых она будет запущена. Таким образом, пользователю нет необходимости заботиться о наличии копии программы на узлах кластера и нет надобности в наличии сетефой файловой системы NFS.

Однако для этого придется установить двунаправленные доверительные отношения между узлами. Необходимо организовать беспарольный доступ главного компьютера кластера на все узлы. Точно такую же процедуру надо провести в обратном направлении, то есть обеспечить беспарольный доступ узлов к главному компьютеру.

Если используется локальная файловая система, то локальные для процесса данные будет лучше сохранять в отдельные файлы - так, как это сделано в нашей тестовой программе, то есть запись данных происходит в файлы с названиями (например) nodeNN.dat, где NN - номер процесса, который монопольно использует данный файл. В противном случае, когда данные пишутся в различные сегменты одного и того же файла, синхронизация такого единого файла между узлами кластера превращается в нетривиальную задачу.

Для синхронизации данных между многочисленными узлами кластера удобно использовать программу pdsh. Эта программа не входит в список программ, устанавливаемых в систему по умолчанию, поэтому придется ее доустановить. В Ubuntu это можно сделать следующей командой: **sudo apt-get install pdsh**.

При использовании для хранения данных локальной ФС, работа должна проходить в три этапа: (1) передача файлов данных с главного компьютера кластера на все вычислительные узлы, (2) запуск и выполнение параллельной задачи, (3) получение на главный компьютер измененных файлов

данных с вычислительных узлов. Пример последовательности команд для выполнения этих трех этапов такой:

```
cat $HOME/nodes| pdsh -R ssh -w - "rsync -uti supergate:$HOME/mpi1/node*.dat $HOME/mpi1/"
mpirun -hostfile $HOME/mpi.hosts -np 3 -s ./diffpw
cat $HOME/nodes| pdsh -R ssh -w - "rsync -uti $HOME/mpi1/node*.dat supergate:$HOME/mpi1/"
```

Здесь мы предполагаем, что файлы с данными называются node00.dat, node01.dat, node02.dat и т.д. Рабочая директория программы - $HOME/mpi1. Текстовый файл $HOME/mpi.hosts содержит список всех узлов кластера (по одному узлу в строке), первый среди которых - главный компьютер, и, наконец, файл $HOME/nodes содержит список всех узлов кластера (аналогично $HOME/mpi.hosts), но в котором отсутствует главный компьютер.

В случае использования для запуска параллельной задачи менеджера ресурсов Torque, задание на размещение задачи в очередь может быть оформлено примерно так:

```
#!/bin/sh
#
#PBS -l nodes=supergate:ppn=1+2:ppn=1
#PBS -N Flops_TEST
#PBS -m abe
#PBS -M yuri@linux-ckb.info
#PBS -l pmem=100mb
#PBS -l pcput=10:00:00
cd /home/mpiuser/mpi1
cat $HOME/nodes| pdsh -R ssh -w - "rsync -uti supergate:$HOME/mpi1/node*.dat $HOME/mpi1/"
mpirun -np 3 -s ./diffpw
cat $HOME/nodes| pdsh -R ssh -w - "rsync -uti $HOME/mpi1/node*.dat supergate:$HOME/mpi1/"
```

Для синхронизации данных используется команда pdsh, которая на всех компьютерах, перечисленных в $HOME/nodes, в параллельном режиме запускает процедуру копирования файлов с главного компьютера кластера, или в обратном направлении. Копирование выполняется с помощью утилиты rsync, которая анализирует время модификации файлов и копирует только файлы с наиболее свежими данными. Перед запуском нашей параллельной программы мы распространяем среди всех узлов наиболее свежие данные, которые, если программа это предусматривает, могут быть использованы для загрузки начального состояния разностной сетки. После окончания работы программы процедура копирования загружает на главный компьютер все результаты счета.

Таким образом, использование локальной файловой системы совсем немного усложняет процедуру выполнения параллельной программы, существенно увеличивая скорость операций ввода-вывода, повышая тем самым эффективность кластера.

2.6 Прерывание параллельной программы

Рекомендации, данные в этом разделе, относятся скорее к общей практике программирования, нежели параллельному программированию, как таковому. Тем не менее, думаю, что стоит заострить на них внимание.

Поскольку методы и способы параллельного программирования имеют отношение к *«тяжелым»* задачам, требующим большого времени счета, то вполне представима ситуация, когда потребуется по каким-то причинам досрочно, до получения окончательного результата, прервать счет. Например, по причине истечения выделенного для работы на кластере времени. В случаях, когда параллельная программа выполняет периодическое сохранение данных на диск, весьма желательно осуществить такое прерывание аккуратно, способом, не приводящим к риску повреждения сохраненных данных. Риск повреждения данных связан с выбором момента времени прерывания. Если задача будет принудительно остановлена в тот самый момент, когда происходят файловые операции ввода-вывода, то есть большая вероятность, что данные сохранятся не полностью, что приведет к невозможности их дальнейшего использования.

Наибольшую актуальность корректное завершение программы приобретает в случае, когда для ее запуска используется менеджер ресурсов Torque, а для сохранения данных - локальная файловая система. Если остановить выполнение параллельной программы системными средствами Torque, то возникает не только риск нарушения целостности сохраненных данных, но и процедуры синхронизации сохраненных данных, которые должны быть запущены по завершению задачи, и которые прописаны в файле-задании Torque, выполнены не будут.

Наиболее простой в реализации способ принудительной остановки программы - это выставление семафоров. Семафор - это простой файл произвольного содержания (или без такового, то есть нулевой длины), наличие которого является сигналом остановки для программы. Наиболее безопасным с точки зрения сохранности данных моментом остановки является промежуток между окончанием очередного цикла итерации, за которым может следовать вызов процедуры сохранения данных, и началом следующего цикла. Рассмотрим фрагмент нашей тестовой программы решения уравнения Лапласа (листинг 3).

Листинг 3

```
      c   Получаем разницу значений функции на слоях
      c   с каждого узла кластера и находим максимальное значение
      c   этой разницы для всей разностной сетки
          call MPI_REDUCE(df, gdf, 1, MPI_REAL8,
     x     MPI_MAX, 0, MPI_COMM_WORLD, ierr)
      c   Сообщаем найденную максимальную разницу
      c   всем узлам кластера
          df=gdf
          CALL MPI_BCAST(df, 1, MPI_REAL8, 0, MPI_COMM_WORLD, IERR)
      c   Выводим на экран разницу между итерациями
          if ( myid .eq. 0 ) write(*,*) 'Diff:',df
      c   Если разница больше заданной, то решение не найдено
      c   и мы идем на следующий цикл
          if (df .GT. 0.01) goto 1
```

Именно в этом месте предыдущий цикл итерации уже закончился, следующий еще не начался, а файловые операции, если они имели место в предыдущем цикле, так же уже завершены.

Обратим внимание на строку 40. В этом месте программы переменной **df** присваивается значение, на основе которого все процессы параллельной задачи принимают решение о необходимости продолжения счета. Логично будет именно в этом месте произвести проверку существования файла-семафора, и в случае его наличия занести в переменную **df** значение, останавливающее дальнейшие вычисления.

С учетом вышесказанного, фрагмент нашей программы приобретает следующий вид (листинг 4):

Листинг 4

```
33  c    Получаем разницу значений функции на слоях
34  c    с каждого узла кластера и находим максимальное значение
35  c    этой разницы для всей разностной сетки
36       call MPI_REDUCE(df, gdf, 1, MPI_REAL8,
37      x    MPI_MAX, 0, MPI_COMM_WORLD, ierr)
38  c    Сообщаем найденную максимальную разницу
39  c    всем узлам кластера
40       df=gdf
40a c    Проверяем существует ли глобальный файл-семафор
40b      inquire(file="/etc/stop.stop", exist=ex)
40c      if ( ex ) df=0.0
40d c    Проверяем существует ли локальный файл-семафор
40e      inquire(file="stop.stop", exist=ex)
40f      if ( ex ) df=0.0
41       CALL MPI_BCAST(df, 1, MPI_REAL8, 0, MPI_COMM_WORLD, IERR)
42  c    Выводим на экран разницу между итерациями
43       if ( myid .eq. 0 ) write(*,*) 'Diff:',df
44  c    Если разница больше заданной, то решение не найдено
45  c    и мы идем на следующий цикл
46       if (df .GT. 0.01) goto 1
```

Как вы заметили, мы проверяем два семафора (файлы с именем "stop.stop"): один, находящийся в общесистемном, каталоге, другой - в рабочей директории программы. Наличие хотя бы одного из них приводит к завершению вычислительного процесса. Первый файл, находящийся в каталоге /etc, может быть создан только тем, кто имеет в системе права суперпользователя. Наличие этого файла влияет на выполнение всех, запущенных на кластере параллельных задач, вне зависимости от того, какой у этих задач рабочий каталог и кому они принадлежат. Второй файл влияет на выполнение только той задачи, в рабочем каталоге которой этот файл находится. Создать такой файл может только владелец задачи.

Проверка двойного семафора сделана для удобства системного администратора, которому для остановки всех задач нет необходимости выяснять, какие задачи на данный момент запущены и в каких рабочих директориях.

Таким образом, для корректного завершения программы пользователю необходимо любым удобным способом создать в рабочей директории программы файл с именем **stop.stop**. Администратор же (предполагаем, что кластер используется несколькими пользователями посредством менеджера Torque) для принудительного завершения всех параллельных задач и освобождения кластера может создать файл с таким же именем в системной директории /etc и минут через 10-20 выдать команду **"qdel all"** на случай, если кто-то не воспользовался рекомендацией проверять семафор /etc/stop.stop.

3 Мониторинг вычислительных систем с помощью пакета Ganglia

3.1 Необходимость мониторинга кластеров

В условиях роста вычислительных центров и экономии на персонале потребность в эффективных инструментах мониторинга вычислительных ресурсов становится важной как никогда. Применительно к вычислительным центрам под термином мониторинг могут пониматься различные задачи в зависимости от того, кто и кому о нем говорит. Например:

- Человеку, запускающему на кластере приложения интересны ответы на такие вопросы: «Когда мое задание будет запущено? Когда завершится его выполнение? Как оно выполняется по сравнению с предыдущим разом?»
- Системному инженеру интересно: "Какова производительность наших машин? Все ли сервисы функционируют правильно? Какие тенденции наблюдаются, и как мы можем лучше использовать наши вычислительные ресурсы?"

Для всего этого многообразия требований вам необходимо выделить среди терабайтов выполняемого кода именно то, что вы хотите отслеживать. На этом дело не кончается: также необходимо выбрать что-то среди множества продуктов и сервисов. Однако, к счастью, существует много средств мониторинга с открытым исходным кодом, и при этом многие из них делают свою работу лучше, чем некоторые коммерческие приложения, созданные для тех же целей.

Самое сложное в использовании инструментов мониторинга с открытым исходным кодом – выработать такой вид установки и конфигурации, который подходит для вашей сетевой среды. Есть две серьезные проблемы, связанные с использованием инструментов мониторинга с открытым исходным кодом:

- Нет такого инструмента, который бы отслеживал все параметры, которые вы хотите и именно так, как вы хотите. Почему? Потому что разные пользователи по-разному видят для себя мониторинг (см. выше).
- Ввиду первой проблемы вам, возможно, придется серьезно адаптировать инструмент, чтобы заставить его работать в вашем вычислительном центре именно так, как вы хотите. Почему? Потому что каждая сеть, какой бы стандартной она ни была, все равно является уникальной.

Кстати, те же самые проблемы существуют и с коммерческими инструментами мониторинга.

Мы будем рассматривать программный пакет Ganglia - инструмент мониторинга вычислительных центров [2]. Этот инструмент широко используется в среде высокопроизводительных вычислений (high performance computing или HPC), но он обладает качествами, которые делают их привлека-

тельными и в других средах (таких как среды cloud-вычислений, комплексы визуализации и центры хостинга). Основное применение Ganglia - это сбор значений различных параметров (метрик) и отслеживании их изменений с течением времени.

Есть и другие проекты с открытым исходным кодом, предназначенные для тех же самых целей, каждый со своими достоинствами и недостатками. Самые популярные среди них - это Cacti, Zenoss, Zabbix, Performance Copilot (PCP), и Clumon. Многие из них (включая Ganglia и некоторые модули расширения Nagios) используют инструмент Тоби Этикера RRDT или MRTG (Multi Router Traffic Grapher) для хранения данных и создания содержательных графиков.

Рассмотрим следующие темы:

- Использование модулей Python для расширения функциональности с помощью интеллектуального интерфейса управления платформой (IPMI, Intelligent Platform Management Interface).
- Использование техники подмены узла Ganglia для мониторинга IPMI.

Наша цель – настроить базовую систему мониторинга HPC кластера Linux, которая бы в той или иной степени соответствовала двум взглядам на мониторинг, указанным выше; систему, в которой:

1. человек, запускающий приложения, может видеть заполненность очереди и видеть узлы, доступные для запуска на них заданий
2. системный инженер может строить графики данных, получать отчеты об использовании кластера и, руководствуясь этими данными, принимать решения о приобретении дополнительного оборудования.

Ganglia – это система мониторинга с открытым исходным кодом, спроектированная для работы с тысячами узлов, изначально разрабатывавшаяся в университете Berkeley. На каждой машине запускается демон gmond, который собирает системную информацию (скорость процессора, использование памяти и т.д.) и посылает ее на определенную машину. Машина, получающая информацию, может отображать ее, а также передавать некоторую обобщенную форму данных вверх по иерархии. Именно благодаря этой иерархической схеме Ganglia так хорошо масштабируется. Накладные расходы, связанные с работой gmond, очень малы, поэтому этот код можно запускать на всех машинах кластера без ущерба для производительности.

Случается, что такой сбор данных все-таки влияет на производительность узлов. В сетях это называется «джиттер» – когда много маленьких сообщений приходят в одно и то же время. Это может произойти в результате, например, синхронизации часов на узлах. Таких ситуаций можно и нужно избегать.

3.2 Ведение наблюдений с помощью Ganglia

Будем полагать, что пакет Ganglia на нашем кластере уже установлен, настроен и запущен. Поэтому перейдем сразу к ведению наблюдений с помощью Ganglia.

Откройте на управляющем сервере браузер и перейдите по адресу: http://localhost/ganglia. По этому адресу вы теперь можете наблюдать за состоянием управляющего сервера. Вы увидите несколько отслеживаемых показателей и их графики.

Многие системные инженеры тратят много сил на то, чтобы понять, как выполняются задания и какую они создают нагрузку на систему. Возможно, для этого они используют свой собственный код или же вовсе не интересовались тем, как работают их коммерческие продукты. Ganglia может помочь собрать профили работы приложений.

Рассмотрим с помощью Ganglia поведение системы во время выполнения тестов Linpack. На рис. 9 показан промежуток времени, в течение которого запускались три различных задания Linpack.

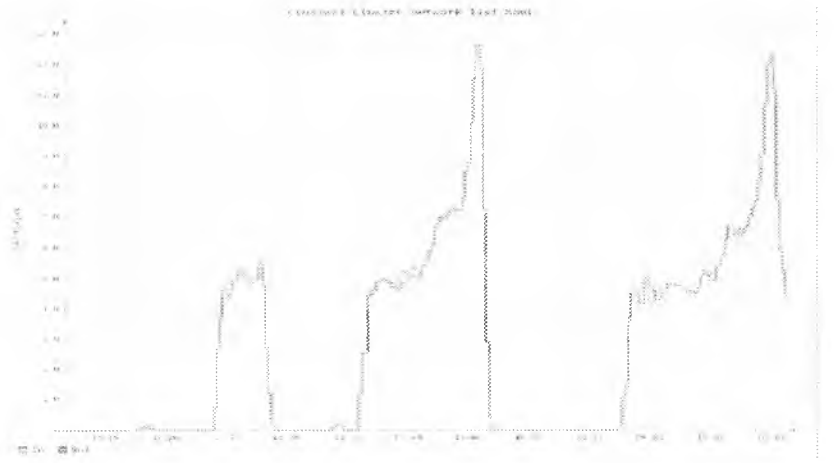

Рис 9. Мониторинг теста Linpack

Как видно из графика, во время выполнения заданий наблюдается некоторая сетевая активность. Также интересно то, что сетевой трафик заметно увеличивается к концу заданий. Даже не зная ничего о Linpack, мы уже можем кое-что сказать о его работе: объем трафика возрастает к концу выполнения заданий.

На рисунках 10 и 11 показано использование процессора и памяти соответственно. Из них видно, что мы выжимаем почти все возможное из нашего процессора и использование памяти также достаточно интенсивное.

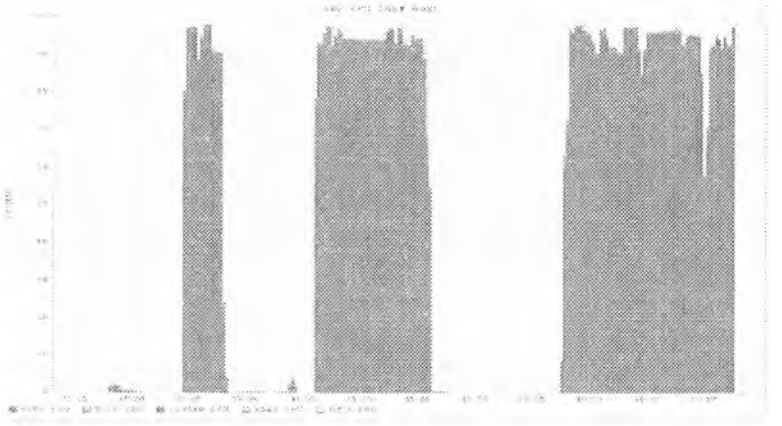

Рис. 10. Использование процессора

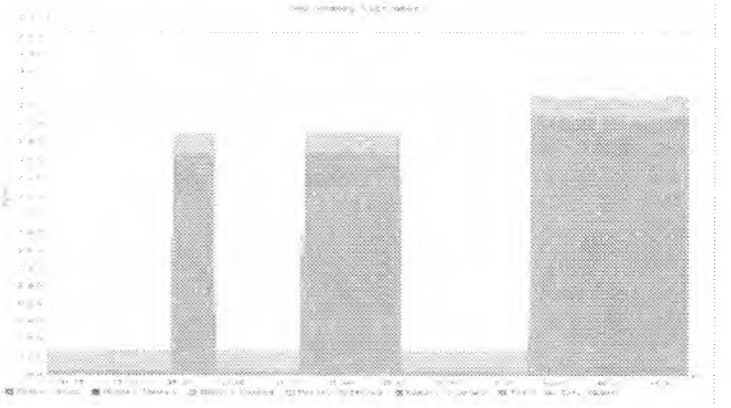

Рис. 11. Использование памяти

Эти графики дают нам представление о работе приложения: мы используем много процессорного времени и памяти, а также начинаем генерировать больше сетевого трафика к концу задания. Конечно, еще очень многих параметров выполнения работы мы не знаем, но все-таки для начала это очень неплохие данные.

Знание таких вещей поможет в будущем сделать лучший выбор, например, при покупке дополнительного оборудования.

3.3 Расширение возможностей

Основной пакет Ganglia предоставляет нам множество полезной информации. Использование модулей расширения (плагинов) Ganglia позволяет расширить его возможности двумя способами:

- Добавлением внутренних модулей расширения
- Добавлением внешних источников данных с помощью техники подмены узла.

Довольно долго в Ganglia широко применялся первый способ. Второй способ является более поздней разработкой. Давайте изучим эти два метода на практических примерах.

3.3.1 Внутренние модули расширения

Внутренние модули расширения реализуются двумя путями.

1. С помощью задач cron, в которых вызывается команда gmetric, осуществляющая ввод данных в Ganglia
2. С помощью написания скриптов в новых модулях расширения для Python

Первый метод широко применялся в прошлом и о нем будет рассказано в следующем разделе, посвященном внешним модулям расширения. Однако использование gmetric довольно неудобно. Чтобы сделать расширение Ganglia более естественным, в Ganglia 3.1.x были добавлены модули расширения для C и Python. Рассмотрим этот метод.

Во-первых, включим модули расширения Python для Ganglia. Надо сделать следующее:

- Отредактировать файл /etc/ganglia/gmond.conf

Найдите в этом файле раздел modules, выглядящий примерно так:

```
modules {
  module {
      name = "core_metrics"
  }
  ...
}
```

Нужно добавить в этот раздел еще один модуль:

```
 module {
  name = "python_module"
  path = "modpython.so"
  params = "/usr/lib64/ganglia/python_modules/"
 }
```

Таким образом, мы включили в Ganglia модули расширения Python. Также несколькими строчками ниже после инструкции include ('/etc/ganglia/conf.d/*.conf') нужно добавить строчку include ('/etc/ganglia/conf.d/*.pyconf'). Это подключает определения добавляемых нами модулей.

- Создать следующие директории:

mkdir /etc/ganglia/conf.d
mkdir /usr/lib64/ganglia/python_modules

1. Выполнить шаги 1 и 2 на всех ваших узлах.

Для этого:

- Скопируйте новый файл gmond.conf на все узлы за которыми мы будем наблюдать.
- Создайте две директории указанные в пункте 2 на каждом узле, чтобы узлы также могли использовать расширения Python.

Доступ на вычислительные узлы выполняется командой:

ssh n1

где n1 – имя первого из четырнадцати вычислительных узлов на нашем кластере.

Копирование на вычислительные узлы выполняется командой:

cpush /source/directory/file /target/directory/file

Теперь, когда узлы подготовлены, давайте создадим новый модуль Python. В нашем примере мы создадим модуль расширения, который будет использовать IPMI драйверы для Linux. Если вы не знакомы с IPMI и работаете с процессорами от Intel или AMD, ознакомьтесь со ссылкой .

Мы будем использовать инструмент IPMItool с открытым исходным кодом для связи с устройствами IPMI на локальной машине. Есть и другие инструменты, например, OpenIPMI и freeipmi. Так как это просто пример, вы можете свободно использовать любой другой инструмент.

Перед началом работы с Ganglia убедитесь, что IPMItool работает на вашей машине. Запустите команду ipmitool -c sdr type temperature | sed 's/ /_/g'; если команда не работает загрузите драйверы IPMI устройства и запустите её снова:

```
modprobe ipmi_msghandler
modprobe ipmi_si
modprobe ipmi_devintf
```

Результат выполнения команды ipmitool выглядит примерно так:

```
Ambient_Temp,20,degrees_C,ok
CPU_1_Temp,20,degrees_C,ok
CPU_2_Temp,21,degrees_C,ok
```

Итак, с помощью модуля расширения Ganglia будем отслеживать температуру окружающей среды. Вот пример модуля расширения ambientTemp.py, который использует IPMI (листинг 23):

Листинг 23. Модуль расширения Python ambientTemp.py

```
import os
def temp_handler(name):
```

```
  # команды, которые мы собираемся выполнить
  sdrfile = "/tmp/sdr.dump"
  ipmitool = "/usr/bin/ipmitool"
  # Перед запуском нужно загрузить драйверы IPMI:
  # modprobe ipmi_msghandler
  # modprobe ipmi_si
  # modprobe ipmi_devintf
  # также надо выставить в nobody права доступа к файлу /dev/ipmi0:
  # chown nobody:nobody /dev/ipmi0
  # разместите это в /etc/rc.d/rc.local

  foo = os.path.exists(sdrfile)
  if os.path.exists(sdrfile) != True:
    os.system(ipmitool + ' sdr dump ' + sdrfile)

  if os.path.exists(sdrfile):
    ipmicmd = ipmitool + " -S " + sdrfile + " -c sdr"
  else:
    print "file does not exist... oops!"
    ipmicmd = ipmitool + " -c sdr"
  cmd = ipmicmd + " type temperature | sed 's/ /_/g' "
  cmd = cmd + " | awk -F, '/Ambient/ {print $2}' "
  #print cmd
  entries = os.popen(cmd)
  for l in entries:
    line = l.split()
  # print line
  return int(line[0])

def metric_init(params):
    global descriptors

    temp = {'name': 'Ambient Temp',
        'call_back': temp_handler,
        'time_max': 90,
        'value_type': 'uint',
        'units': 'C',
        'slope': 'both',
        'format': '%u',
        'description': 'Ambient Temperature of host through IPMI',
        'groups': 'IPMI In Band'}

    descriptors = [temp]

    return descriptors

def metric_cleanup():
    '''Очистка модуля метрик.'''
```

```
    pass

#Это код для отладки и юнит-тестирования
if __name__ == '__main__':
    metric_init(None)
    for d in descriptors:
        v = d['call_back'](d['name'])
        print 'value for %s is %u' % (d['name'],  v)
```

Скопируйте листинг 1 и сохраните его в /usr/lib64/ganglia/python_modules/ambientTemp.py. Сделайте это для всех узлов кластера.

Теперь, когда мы разместили скрипт на всех узлах кластера, сообщим Ganglia, как его нужно выполнять. Создадим новый файл /etc/ganglia/conf.d/ambientTemp.pyconf со следующим содержимым:

Листинг 24. Ambient.Temp.pyconf

```
modules {
  module {
    name = "Ambient Temp"
    language = "python"
  }
}

collection_group {
  collect_every = 10
  time_threshold = 50
  metric {
    name = "Ambient Temp"
    title = "Ambient Temperature"
    value_threshold = 70
  }
}
```

Сохраните листинг 24 на всех узлах.

И, наконец, перед перезапуском gmond надо выставить права на работу с устройством IPMI в *nobody*, чтобы никто не мог с ним работать. Иначе интерфейс IPMI станет очень уязвимым для атак злоумышленников!

Вот пример того, как это сделать: chown nobody:nobody /dev/ipmi0.

Теперь перезапустите gmond на всех машинах. Если все заработает, то, обновив окно браузера, вы должны увидеть что-то вроде этого (рис. 12):

Преимуществом внутренних метрик является то, что они позволяют запускать на машинах программы и передавать от них информацию вверх по цепочке с помощью того же механизма, который используют другие метрики. Недостатком этого подхода, особенно для IPMI, является необходимость дополнительного конфигурирования всех машин.

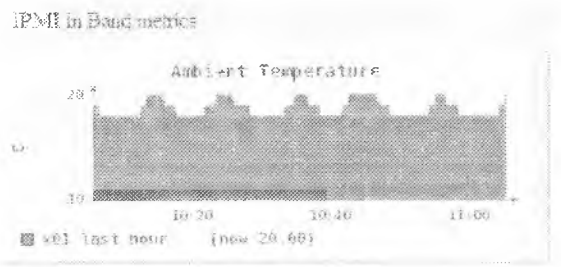

Рис. 12. Внутренние метрики IPMI

Обратите внимание, что нам пришлось написать скрипт на Python'е, подготовить конфигурационный файл и задать в gmond.conf правильные настройки. Выполнение такой работы на каждой машине для каждой метрики может быть весьма утомительным. IPMI – это внешний инструмент, поэтому существует более удобный способ.

3.3.2 Внешние модули расширения (использование техники подмены узла - Host spoofing)

Воспользуемся программой gmetric - мощным инструментом командной строки для добавления информации в Ganglia, и сообщим ему на каких машинах мы работаем. Таким способом мы можем наблюдать за всем, чем угодно.

На gmetric существует огромное количество уже написанных скриптов.

В качестве учебного примера рассмотрим способ запуска ipmitool для получения удаленного доступа к машинам:

- ***Убедитесь, что ipmitool работает сам по себе.***

Можно настроить BMC (контроллер мониторинга на целевой машине) так, чтобы иметь возможность запускать на нем команды IPMI. Пусть сервер мониторинга называется redhouse. С redhouse можно наблюдать за всеми остальными узлами кластера. Именно на redhouse запущен gmetad и именно с него через web браузер можно получать всю информацию о Ganglia.

Один из узлов моего кластера называется n1. На n1 установлен для BMC IP-адрес, разрешающийся в имя n1-bmc. Вот как можно получить удаленный доступ к этой машине:

```
# ipmitool -I lanplus -H n1-bmc -U USERID -P PASSW0RD sdr dump \
/tmp/n1.sdr
Dumping Sensor Data Repository to '/tmp/n1.sdr'
# ipmitool -I lanplus -H n1-bmc -U USERID -P PASSW0RD -S /tmp/n1.sdr \
sdr type
                        Temperature
Ambient Temp     | 32h | ok  | 12.1 | 20 degrees C
CPU 1 Temp       | 98h | ok  |  3.1 | 20 degrees C
```

```
CPU 2 Temp       | 99h | ok  |  3.2 | 21 degrees C
```

Теперь поместим это в скрипт и передадим его в gmetric.

[1] Делаем скрипт, который передает информацию от ipmitool в gmetric.

Создадим следующий скрипт и разместим его на сервере мониторинга по пути usr/local/bin/ipmi-ganglia.pl:

```
#!/usr/bin/perl
# vallard@us.ibm.com
use strict;  # чтобы сделать все чище
use Socket;  # для разрешения имен в IP-адреса

# для зачистки после ветвлений
use POSIX ":sys_wait_h";
# nodeFile: это простой текстовый файл со списком узлов:
# например:
# node01
# node02
# ...
# nodexx
my $nodeFile = "/usr/local/bin/nodes";
# бинарный файл gmetric
my $gmetric = "/usr/bin/gmetric";
# бинарный файл ipmitool
my $ipmi = "/usr/bin/ipmitool";
# id пользователя для BMC
my $u = "xcat";
# пароль для BMC
my $p = "f00bar";
# открываем файл со списком узлов и проходим по всем узлам
open(FH, "$nodeFile") or die "can't open $nodeFile";
while(my $node = <FH>){
 # делаем ветвления (fork), чтобы каждый удаленный вызов выполнялся
    параллельно
 if(my $pid = fork()){
  # родительский процесс
  next;
 }
 # здесь начинается порожденный процесс
 chomp($node);  # убираем символы новой строки
 # разрешаем IP адрес узла и подменяем его
 my $ip;
 my $pip = gethostbyname($node);
 if(defined $pip){
  $ip = inet_ntoa($pip);
 }else{
```

```
  print "Can't get IP for $node!\n";
  exit 1;
 }
 # проверяем, существует ли файл кэша SDR.
 my $ipmiCmd;
 unless(-f "/tmp/$node.sdr"){
  # кэша SDR нет, поэтому попробуем его создать...
  $ipmiCmd = "$ipmi -I lan -H $node-bmc -U $u -P $p sdr dump
                /tmp/$node.sdr";
  `$ipmiCmd`;
 }
 if(-f "/tmp/$node.sdr"){
  # выполняем команду для кэша, так как это быстрее
  $ipmiCmd = "$ipmi -I lan -H $node-bmc -U $u -P $p -S /tmp/$node.sdr sdr
                type
                                        Temperature ";
  # помещаем весь результат в массив @out
  my @out = `$ipmiCmd`;
  # проходим по элементам массива @out.
  foreach(@out){
   # каждая строка результата выглядит так:
   # Ambient Temp     | 32h | ok  | 12.1 | 25 degrees C
   # мы разбираем строку следующим образом:
   chomp(); # убираем символ новой строки
   # выбираем первое и пятое поля.  (Description и Temp)
   my ($descr, undef, undef, undef,$temp) = split(/\|/);
   # убираем пробелы в описании
   $descr =~ s/ //g;
   # выбираем только температуру (полагая ее всегда в градусах
      Цельсия C)
   $temp = (split(' ', $temp))[0];
   # убеждаемся, что температура является числом:
   if($temp =~ /^\d+/ ){
    #print "$node: $descr $temp\n";
    my $gcmd = "$gmetric -n '$descr' -v $temp -t int16 -u Celcius -S
                 $ip:$node";
    `$gcmd`;
   }
  }
 }
 # порожденный поток завершается.
 exit;
}
# ждем окончания всех ветвлений fork...
while(waitpid(-1,WNOHANG) != -1){
 1;
}
```

Помимо разбора строк, скрипт просто запускает команду ipmitool и получает с ее помощью значения температуры. Затем для каждой метрики с помощью команды gmetric он помещает эти значения в Ganglia.

3. Запускаем скрипт как задачу cron.

Запустите crontab -e.

Можно добавить в скрипт crontab следующую строку, чтобы скрипт запускался каждые 30 минут:

30 * * * * /usr/local/bin/ipmi-ganglia.sh.

4. Открываем Ganglia и смотрим результаты.

Открыв в web браузере Ganglia и взглянув на графики для одного из узлов, мы можем видеть, что произошла подмена и теперь для каждого узла на всех графиках показано подмененное имя:

Одним из недостатков подмены узла является то, что таким метрикам присваивается категория no_group. В gmetric, похоже, нет удобного способа изменить группировку, как в случае внутренних расширений.

Итак, мы увидели, как Ganglia может помочь понять характеристики приложения. Мы рассмотрели, как расширять возможности Ganglia с помощью внутренних скриптов, а также как использовать внешние скрипты с подменой адреса узла.

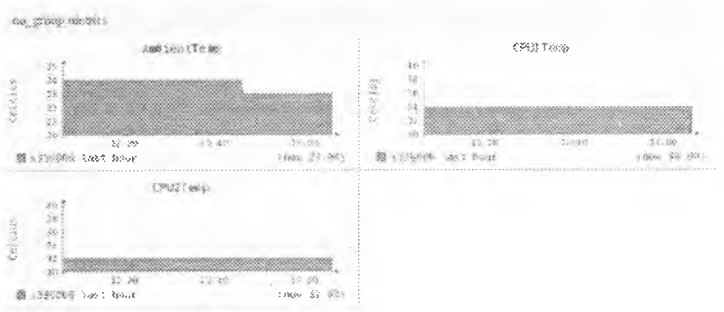

Рис. 13. Метрика no_group

Таким образом, мы решали задачи мониторинга, поставленные системным инженером. Теперь мы можем отслеживать производительность всей системы и видеть, насколько используются машины. Мы можем сказать, простаивают ли машины, или постоянно используют, например, 60 процентов своих возможностей. Теперь мы можем даже сказать, какие машины нагреваются больше или меньше остальных, и подумать, можно ли улучшить их расположение в стойке.

4. Система пакетной обработки заданий torque

4.1 Обзор torque

В этой главе описываются общие принципы, которые реализует система пакетной обработки заданий torque [4-6]. Рассматриваются ее компоненты, представление физических вычислительных узлов в системе, понятие задания и ресурса.

4.1.1 Общая характеристика

Torque – одна из версий системы PBS (Portable Batch System - система пакетной обработки заданий). Torque управляет загрузкой вычислительных комплексов, состоящих из определенного количества вычислительных узлов, работающих под управлением операционной системы семейства Unix. Система пакетной обработки заданий (далее - СПО) необходима при одновременном выполнении заданий (jobs) несколькими пользователями на одном вычислительном комплексе.

В результате применения СПО вычислительные ресурсы используются оптимально: сводится к минимуму как перегрузка какого-либо одного узла (об узлах см. далее по тексту), так и его простой. Torque обычно применяется в областях, где высока интенсивность использования вычислительных мощностей.

Таким образом, torque обеспечивает контроль над вычислительными ресурсами, что, в конечном итоге, снижает зависимость от системных администраторов и операторов, освобождая их для решения других задач. Также torque дает возможность контролировать выполнение заданий, используя очереди и планировщик заданий.

Принцип работы torque заключается в следующем. Задания создаются и управляются сервером заданий. Клиенты СПО взаимодействуют с сервером заданий, который предоставляет соответствующие сервисы. Пользователь взаимодействует с СПО посредством утилит командной среды. Сервер заданий является демоном (daemon), который осуществляет постановку заданий в очередь, управление очередями и выполнение задания от имени клиента СПО. Сервисы, предоставляемые сервером заданий, доступны посредством утилит командной строки (batch utilities), которые запускают пользователи. В следующем разделе описаны компоненты torque.

Утилиты torque можно запускать как из командной строки операционной системы, так и посредством графического интерфейса. Набор, синтаксис и семантика (т.е. выполняемые операции) пакетных утилит соответствуют стандарту POSIX 1003.2d. Графический интерфейс в настоящем руководстве рассматриваться не будет.

4.1.2 Структура torque

В этом разделе рассматриваются основные компоненты torque и их взаимосвязь. Схема компонентов представлена на рис. 8.

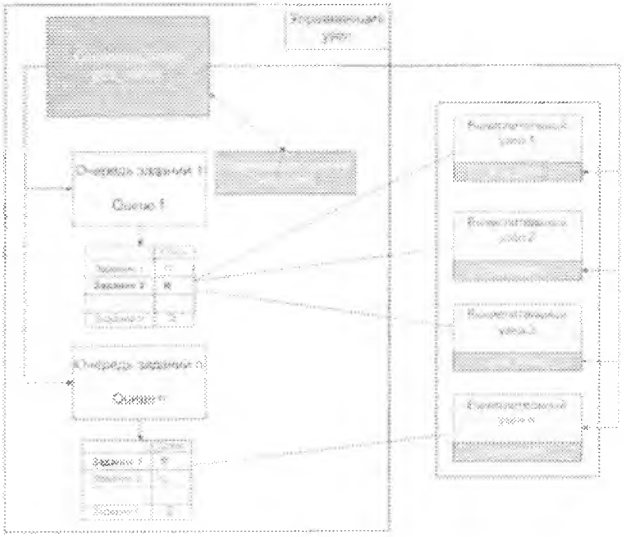

Рис. 8 Взаимодействие компонентов torque

4.1.2.1 Сервер заданий

Демон сервера заданий (Job Server daemon) – центральная точка torque. В настоящем Руководстве будет применяться термин «сервер» или имя процесса pbs_server. Все команды и другие процессы-демоны взаимодействуют с сервером посредством сети по протоколу IP. Главная задача сервера - обеспечить базовые сервисы для исполнения пакетных заданий, такие как получение/создание задания (batch job), изменение задания, обеспечение надежности функционирования системы заданий путем защиты от неполадок в системе, и исполнение задания.

Обычно имеется единственный сервер, управляющий конкретным набором ресурсов. Однако в общем случае серверов может быть несколько.

4.1.2.2 Сервис вычислительного узла: процесс pbs_mom

Клиент, запускающий задания (Job Executor; или просто – клиент заданий) – служба (service) операционной системы, физически осуществляющая запуск задания. Эта служба, называемая pbs_mom, неформально обозначается MOM, поскольку является спредкомї (mother) всех исполняемых заданий. MOM запускает задание, когда получает его копию с сервера заданий. MOM создает новый сеанс от имени одного из пользователей (которым не

является root), зарегистрированных в системе. Запуск задания производится в сеансе оболочки данного пользователя. Например, если пользователь работает в оболочке csh, то MOM создает сеанс, где запускается как файл .login, так и файл .cshrc. MOM также ответственен за предоставление пользователю результата работы задания, по умолчанию выводимого на консоль сервера. Этот результат может быть сохранен в нужном месте. Сервис MOM запускается на вычислительном узле (или узлах), который будет выполнять задание.

4.1.2.3 Представление вычислительных узлов

Вычислительные узлы кластера представляются в torque определенным образом. Прежде чем обсуждать работу с вычислительными узлами (compute nodes) кластера, необходимо ввести некоторые определения.

Вычислительный узел (compute node) – это отдельная компьютерная система (или просто компьютер) с одним образом (image) операционной системы, унифицированным виртуальным адресным пространством, одним или более процессором и одним или более IP-адресом. Часто термин исполняющий хост (execution host) также используется для обозначения узла. Компьютер, содержащий несколько процессоров и работающий под управлением одной операционной системой, является одним узлом. Узлы делятся на два типа: узлы общего типа (cluster node) и разделяемые по времени узлы (timeshared).

Узел общего типа (cluster node) – узел, назначением которого является параллельное выполнение заданий. Если такой узел имеет более одного виртуального процессора, они могут быть назначены разным заданиям (англ. jobshared – распределены между заданиями) или использованы для выполнения единственного задания (англ. exclusive – эксклюзивный доступ).

Такая возможность непрерывно распределять ресурсы каждого узла важна для некоторых приложений, работающих одновременно на нескольких узлах (multi-node applications). Обратите внимание, что torque обязывает придерживаться схемы один-к-одному при выделении виртуальных процессоров (см. далее) заданию. Таким образом, один ВП работает только с одним заданием.

Разделяемый во времени узел (timeshared node). В противоположность узлам общего типа, такие узлы всегда могут обслуживать несколько заданий одновременно. Часто термин хост (host) используется вместо термина узел (node) совместно с термином "timeshared". Разделяемый во времени узел никогда не будет эксклюзивно выделен для выполнения единственного задания.

Виртуальный процессор (VP – virtual processor; далее ВП). Для узла может декларироваться наличие одного или нескольких виртуальных процессоров. Слово виртуальный используется, поскольку обозначенное число виртуальных процессоров может не соответствовать числу реальных процессоров в узле. Число ВП в узле по умолчанию есть число реально функционирующих ядер физических процессоров.

Атрибуты узла

Вычислительные узлы кластера настраиваются в torque установкой атрибутов. Атрибуты также используются в файле конфигурации узлов (см. раздел 2.1.2.3). Список основных атрибутов приведен в этом разделе. Установка атрибутов командой qmgr описывается в разделе 2.2.2.1.

comment

Комментарий для узла.

max_running

Максимальное количество заданий, которое может быть одновременно запущено на узле.

max_user_run

Максимальное число заданий, принадлежащих одному пользователю, которое допускается одновременно выполнять на узле.

no_multinode_jobs

Если этот атрибут имеет значение true, то задания, которые запрашивают для своего запуска несколько узлов, не будут выполняться на данном узле.

np

Количество виртуальных процессоров.

ntype

Задает тип узла. Типы узлов описаны выше, в предисловии к данному разделу. Значения могут быть следующими: *time-shared, cluster*.

properties

Свойства узла, определяемые пользователем. Значением может быть любая строка, начинающаяся с буквенного символа или такие строки, разделенные запятой.

resources_available

Ресурсы, доступные на узле. Конкретный ресурс задается после символа точки «.»:

resources_available.ncpus. Соответственно, все ресурсы, доступные torque, можно указывать в качестве значения этого атрибута. Понятие ресурсов описывается в разделе 2.1.4.

state

При помощи этого атрибута можно задать или просмотреть статус (state) узла. Возможные значения для состояния: *free, offline, down, job_busy, job_exclusive, busy, state_unknown*. Первые два состояния может установить пользователь, остальные – только системные процессы.

Файл конфигурации узлов

Вычислительные узлы, где запускаются задания, определяются при взаимодействии сервера и других компонентов torque (в частности, планировщика заданий, см. раздел 2.1.2.4). Взаимодействие возможно благодаря файлу конфигурации nodes. Файл располагается по следующему пути:

PBS_HOME/server_priv/nodes

В файле содержится список узлов и их атрибуты. Без списка узлов сервер не сможет создать взаимодействовать с MOM посредством специально-

го потока (communication stream). У MOM также не будет возможности отчитываться о запущенных заданиях и уведомлять сервер о завершении задания. Здесь *PBS_HOME* - переменная окружения, содержащая путь к рабочей директории torque. Простой файл конфигурации узлов создается в процессе установки torque. Этот файл содержит только название хоста, с которого была запущена инсталляция. Этот узел будет считаться разделяемым по времени (timeshared). Файл конфигурации узлов можно изменять двумя путями. Если сервер не запущен, это можно делать напрямую в текстовом редакторе. Если же сервер работает, следует использовать команду qmgr для изменения списка узлов.

Файл конфигурации является обычным текстовым файлом, каждая строка которого записана в форме:

node_name[:ts] [attributes]

Здесь *node_name* – это сетевое имя узла. Опциональный параметр "*:ts*" добавляется к имени, указывая таким образом, что узел является разделяемым по времени (timeshared). Также узлы могут иметь ассоциированные с ними атрибуты. Атрибуты перечисляются в виде:

attribute_name=value

Например, выражение np=<число> может быть использовано для определения числа виртуальных процессоров на узле. Если это выражение не указано для узла общего типа (cluster node), то число виртуальных процессоров будет равно 1. Пример файла содержит Листинг 5.

Листинг 5

```
master:- #cat /var/torque/server_priv/nodes
node-1 np=4
node-2 np=4
node-3 np=4
node-4 np=4
node-5 np=4
node-6 np=4
node-7 np=4
node-8 np=4
node-9 np=4
node-10 np=4
master:- #
```

Для вывода сведений об узлах используется команда pbs_nodes.

4.1.2.4 Планировщик заданий

Демон планировщика заданий (Job scheduler daemon), процесс которого называется *pbs_sched*, занимается распределением ресурсов между заданиями. Он определяет, когда данное задание будет запущено и какие ресурсы ему будут выделены. Планировщик взаимодействует с MOM на узлах, запрашивая у них состояние системных ресурсов; а также с сервером заданий для получения списка заданий, доступных для выполнения. Планировщик

использует файл конфигурации узлов для определения узла или узлов, где будет запущено задание.

4.1.3 Понятие задания

Задание torque представляет собой абстрактную сущность, состоящую из набора команд и параметров. Задание представляется пользователю в виде скрипта для оболочки (shell), содержащего требования к ресурсам, атрибуты задания и набор команд, которые необходимо выполнить.

Единожды создав скрипт задания, им можно пользоваться столько раз, сколько необходимо, также возможна его модификация. Задание сначала необходимо поставить в очередь torque (submit), затем из этой очереди оно будет передано на один узлов для выполнения. Очередей заданий (batch queue) может быть несколько. После установки torque очередей заданий не существует. Необходимо сначала создать очередь заданий, а затем уже ставить их в эту очередь.

Настроенный вариант системы включает одну очередь заданий.

Задание может быть обычным (regular) и интерактивным (interactive). Обычное задание ставится в очередь и затем ожидает своего выполнения, результат будет записан в указанное пользователем место. Интерактивное задание отличается тем, что потоки ввода и вывода перенаправляются соответственно на экран и клавиатуру, соответственно, команды задания вводятся с клавиатуры непосредственно. Об интерактивных заданиях более подробно будет рассказано далее.

4.1.3.1 Пример задания

Вот пример простого скрипта задания:

```
1 #!/bin/sh
2 #PBS -l walltime=1:00:00
3 #PBS -l mem=400mb
4 #PBS -l ncpus=4
5 #PBS -j oe
6
7 ./subrun
```

Первая строка является стандартной для любого скрипта с описанием задания, она определяет, какая оболочка используется для исполнения сценария. Оболочка sh используется по умолчанию для запуска сценария, но можно использовать и другую. Строки со 2-й по 5-ю являются директивами torque. Система будет читать скрипт до тех пор, пока не найдет первую строку, которая не является валидной директивой torque, и останавливается. Это означает, что оставшаяся часть сценария содержит список команд или задач, которые пользователь желает запустить. При выполнении данного примера, torque обнаружит такие команды в строках 6 и 7.

Далее будет приведено описание команды qsub, выступающей в роли командного интерпретатора. Она используется, в том числе для постановки

задания в очередь torque. Любая опция, которую определяется в команде qsub, может также выступать в роли директивы внутри скрипта torque.

Строки 2–4 определяют опцию ресурса "–l", далее следует запрос определенного ресурса. Конкретно, строки 2–4 сообщают, что запрашиваются ресурсы, объем которых предполагает: не более 1 часа на выполнение, а также 400 Мб памяти и 4 процессоров.

Строка 5 не является директивой запроса ресурса. Опция "–j oe" требует, чтобы torque объединила (join) потоки вывода stdout и stderr в единый поток stdout.

И, наконец, строка 7 является командой для выполнения, которую пользователь хочет запустить. Данный пример запускает программу *subrun*. Хотя в примере имеется только одна команда, можно добавить необходимое количество программ, заданий и шагов.

4.1.3.2 Понятие атрибутов задания

Задание обладает определенным набором атрибутов, значения которых задаются изначально при создании задания и могут быть изменены в процессе его выполнения. Примеры атрибутов задания - название задания, время выполнения, путь к выходному файлу и др.

Атрибуты задания задаются при постановке задания в очередь. Также их можно изменить уже после этого по различным причинам (например, была сделана ошибка при определении ресурсов или истекло время выполнения задания и его необходимо продлить). Независимо от причины изменения атрибутов, для этого имеется команда *qalter*.

Большинство атрибутов может изменить владелец задания, которым может быть пользователь, поставивший задание в очередь командой qsub, либо произвольная учетная запись, указанная при постановке в очередь. Тем не менее, если задание уже выполняется, то лимиты ресурсов не могут быть изменены. Такими ными ресурсами являются: процессорное время, обычное время, число задействованных процессоров, объем памяти.

Примером атрибутов задания может служить название задания, его идентификатора, желаемое время начала выполнения задания.

4.1.3.3 Очереди заданий

Очередь torque - это сущность, содержащая задания. Torque поддерживаст два типа очередей:

Исполняемая очередь (execution queue), содержащая задания, готовые для выполнения. Задания могут запускаться только из очередей этого типа.

Очередь перемещения (routing queue), в которой находятся задания, предназначенные для перестановки в другие очереди, в том числе те, которые находятся на других серверах заданий. Очередей обоих типов может быть несколько. Также очереди имеют свои атрибуты. Более подробно об атрибутах см. в Руководстве администратора torque.

4.1.3.4 Возможные состояния задания

Задания в очередях могут находиться в различных состояниях. Возможные состояния перечислены далее.

Сокращение	Описание
C	complete; Задание успешно завершило свою работу
E	exit; Прерывание работы задания
H	hold; Задание заблокировано
Q	queued; Задание поставлено в очередь и готово для выполнения
R	running; Задание выполняется
T	transiting; Задание перемещается в другое место (очередь)
W	waiting; Задание ожидает, пока подойдет очередь для его выполнения, например, задание может ожидать определенного времени для своего выполнения или завершения выполнения другого задания, от котого зависит. Задание в этом состоянии не может быть выполнено
S	suspended; Пауза в работе задания

4.1.4 Понятие ресурса. Типы ресурсов, управляемых torque

Задание может запросить для запуска множество разных ресурсов, таких как процессоры, память, время (обычное и процессорное). Также может понадобиться дисковое пространство. Список ресурсов определяется с использованием опции –l список_ресурсов команды qsub или в скрипте задания. Таким образом, выделяются ресурсы, необходимые для выполнения задания или определяется их лимит, который может быть выделен. Если лимит не устанавливается для какого-либо ресурса, то он считается равным бесконечности.

Аргумент список_ресурсов записывается в виде:

resource_name[=[value]][,resource_name[=[value]],...|

Здесь resource_name – название ресурса, value – значение. Значения могут представляться в нескольких единицах измерения, зависящих от природы самого ресурса (например, время записывается в соответствующем формате [[часы:минуты]:секунды[.миллисекунды]; размер памяти указывается в байтах - b, килобайтах - kb, мегабайтах - mb).

Ресурсы могут быть следующих видов: количество процессоров, объем памяти, требуемое ПО, объем виртуальной памяти, количество времени и др.

4.2 Настройка torque

СПО предоставляет вычислительную среду, абстрагирующую пользователя от физической (аппаратной) реализации вычислительного кластера. Пользователь определяет задания, которые необходимо выполнить. Система в нужный момент принимает решение о запуске и возвращает результаты операции. Если все доступные узлы заняты, то СПО ожидает, пока ресурсы будут доступны. С точки зрения torque, задание сначала создается, а затем ставится в очередь.

Еще один пример задания содержит Листинг 6. Обратите внимание, что опции всех команд чувствительны к регистру.

Листинг 6

```
test@master:-> cat ./test.script
#!/usr/bin/csh

#PBS -d /home/test
#PBS -l ncpus=4
#PBS -N Hostname
/bin/hostname
test@master:->
```

4.2.1 Взаимодействие torque с пользовательской средой

Чтобы системная среда надлежащим образом взаимодействовала с torque, необходимо проверить несколько моментов. В большинстве случаев среда настраивается системным администратором.

Чтобы torque работала правильно, необходимо выполнение следующих условий:

- необходимо, что все скрипты запуска оболочки были корректными;
- пользователь должен иметь учетную запись, отличную от root всех па вычислительных узлах (подробности см. в следующем разделе).

4.2.1.1 Настройка пользовательской среды на примере оболочки csh

В выполнении пользовательского задания могут возникнуть сложности, если скрипты запуска пользовательской оболочки (например, для оболочки csh - это файлы .cshrc, .login или .profile; для оболочки bash - .bashrc) содержат команды, которые пытаются использовать стандартные потоки. Подобная последовательность команд в таких файлах должна быть пропущена путем проверки переменной окружения PBS_ENVIRONMENT. Вот пример использования подобной методики в файле .login:

```
...\\
setenv MANPATH /usr/man: /usr/local/man :$MANPATH
if ( ! $?PBS\_ENVIRONMENT ) then
 использование стандартных потоков (например, вывод на консоль )
endif
```

Нужно внимательно относиться к тем командам в сеансовых файлах пользователя, которые выводят на консоль какой-либо текст при работе в torque. Как и в предыдущем примере, команды, которые выводят текст в поток stdout, не должны быть выполнены при запуске через torque. Это достигается так же, как и в приведенном примере с файлом .login, а именно:

```
...\\
setenv MANPATH /usr/man: /usr/local/man :$MANPATH
if ( ! $?PBS\_ENVIRONMENT ) then
  команды , выполняющие стандартный вывод
endif
```

При запуске задания torque, «выходное состояние» ("exit status") последней команды, выполненной в задании, являющееся отчетом для оболочки выполнения, будет таковым и для torque. Это важно для зависимых заданий и для построения цепочек заданий. Однако последняя исполненная команда может не быть последней командой в задании. Такое может иметь место, если задание выполняется в оболочке csh на хосте и там имеется файл .logout. В данной ситуации последняя команда, выполненная из файла .logout, не является командой задания.

Чтобы предотвратить это, необходимо сохранить выходное состояние в файле .logout путем запоминания его в начале файла, а затем выполнить выход с этим статусом в конце, как показано ниже:

```
set EXITVAL = $status
  содержимое файла .logout
exit $EXITVAL
```

4.2.1.2 Переменные окружения

Для задания в системе torque существует некоторое количество переменных окружения. Одни переменные берутся из пользовательской среды и передаются заданию, другие создаются самой torque, третьи могут явно создаваться пользователем для эксклюзивного использования заданием torque.

Примечание: *Переменные окружения torque существуют в сеансе, создаваемом командой qsub. В обычной оболочке, из которой происходит запуск qsub, эти переменные не видны.*

Все переменные, существующие в задании, имеют имена, начинающиеся с "PBS_". Некоторые из них также предваряются заглавной О: "PBS_O_", что говорит о происхождении переменной из среды выполнения задания (например, пользовательской).

Далее приведен короткий пример, демонстрирующий использование наиболее полезных переменных и их типичных значений:

```
PBS_O_HOME=/home/test
PBS_O_LOGNAME=test
PBS_O_PATH=/usr/new/bin:/usr/local/bin:/bin
PBS_O_WORKDIR=/share/hpl/bin/
```

Полный список переменных окружения torque приведен в приложении А.

4.2.2 Команды настройки torque

В этой главе описываются команды настройки torque, такие как qmgr, pbsnodes.

4.2.2.1 Настройка узлов с помощью команды qmgr

Команда qmgr предоставляет пользователю интерфейс взаимодействия с сервером заданий torque. Эта команда позволяет настраивать узлы и их атрибуты. Qmgr можно также как интерактивный интерфейс к менеджеру torque.

Описание команды qmgr

Команда считывает директивы из стандартного потока ввода, синтаксис директив проверяется и соответствующий запрос отсылается к одному или нескольким серверам заданий. По умолчанию команду может выполнять только пользователь root.

Синтаксис команды qmgr таков:

qmgr [–a] [–c command] [–e] [–n] [–z] [server...]

Опции команды описываются далее.

Опции команды qmgr

–a	Прервать работу qmgr в случае любых синтаксических ошибок или запросов, отклоненных сервером.
–c < “команда” >	Выполнение единственной команды и завершение работы qmgr.
–e	Перенаправить эхо-вывода в стандартный поток.
--n	Только проверка синтаксиса, без выполнения команд.
–z	Не выводить сообщения об ошибках в стандартный поток ошибок.

Если команда qmgr запускается без опции –c и стандартный поток вывода ассоциирован с терминалом, qmgr выведет приглашение и директивы будут считываться с клавиатуры (стандартного потока ввода).

Создание и удаление узлов

Указав опцию –c при выполнении qmgr, можно задать команду создания нового или удаления существующего узла. Указанные операции необходимо при помощи qmgr всегда, если процесс pbs_server запущен.

Для добавления нового узла используйте подкоманду create команды qmgr:

qmgr "create node node_name [<атрибут>=<значение>]"

Например:

qmgr -c "create node node003"

Для изменения параметров узла после создания узла используйте подкоманду set:

set node node_name [attribute[+|-]=value]

Символы "+" и "-" следует использовать, если атрибут допускает несколько значений.

Для удаления узлов используется подкоманда delete:

Qmgr -c "delete node mars"

Qmgr -c "delete node Pluto"

Примеры работы с командой qmgr

Далее приводятся примеры работы с qmgr. Листинг 7 содержит пример вывода информации об объектах типа server. В этом примере qmgr используется с опцией –c.

Листинг 7

```
master:- # qmgr -c "list server"
Server master.localdomain
server_state = Active
scheduling = True
total_jobs = 0
state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
default_queue = batch
log_events = 511
mail_from = adm
scheduler_iteration = 600
node_check_rate = 150
tcp_timeout = 6
pbs_version = 2.1.8
master:- #
```

В листинге 8 приводится пример изменения типа узла в интерактивном режиме команды qmgr:

Листинг 8

```
master:- #qmgr
Max open servers: 4
Qmgr: set node node-32 ntype=time-shared
Qmgr:
```

Листинг 9 приводит пример выполнения команды "list queue" для вывода информации об объекте типа "queue" (очередь), который называется "batch":

Листинг 9

```
test@master:-> qmgr -c "list queue batch"
Queue batch
queue_type = Execution
total_jobs = 0
state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
mtime = Tue Sep 4 15:36:09 2007
enabled = True
```

started = True

4.2.2.2 Команда pbsnodes

Команда pbsnodes может быть использована для получения сведений об узлах и изменения их состояния.

Синтаксис команды pbsnodes следующий:

pbsnodes [-a|-l|-s][-c узлы][-d узлы][-o узлы][-r узлы][узел1 узел2 . . .]

Пример запуска команды без опций содержит Листинг 10 :

Листинг 10

```
master:- #pbsnodes
node-1
state = down
np = 4
ntype = cluster
node-2
state = down
np=4
ntype = cluster
```

Далее приводится описание опций команды.

узел1 узел2 …	Если указаны только наименования узлов без дополнительных опций, то выводится состояние этих узлов.
–a	Выводит список всех узлов и значения каждого атрибута узла.
–c узлы	Изменяет состояние down или offline на free, т.е. узел становится доступным для выполнения заданий.
–d узлы	Устанавливает состояние down для указанных узлов. Эти узлы будут в дальнейшем не доступны для выполнения заданий. Если команда приводится без списка узлов, то все узлы переводятся в состояние down.
–l	Выводит список всех узлов.
–o узлы	Переводит указанные узлы в состояние offline, даже если они на данный момент используются. Это состояние не может быть изменено никакими автоматическими, не зависимыми от пользователя, средствами. Листинг 7 содержит результат выполнения команды с этой опцией.
-r узлы	Отменяет перевод в состояние offline для указанных узлов.
-s	Определяет сервер заданий, на который будет послан запрос.

Листинг 11

master:- #pbsnodes -o node-1
master:- #pbsnodes
node-1
state = down, offline
np = 4
ntype = cluster
node-2
state = down
np=4
ntype = cluster

4.3 Использование torque

В этой главе описывается команды, предназначенные для взаимодействия с пользователем, запускающим задания и контролирующим их выполнение. Рассматривается команда постановки задания в очередь qsub, а также команды изменения состояния задания qhold и qrls. Даются сведения о команде получения информации о заданиях qstat.

4.3.1 Команда qsub

В этом разделе рассмотрена команда qsub, предназначенная для постановки задания с различными параметрами в одну из существующих очередей.

Допустим, что ранее скрипт с описанием задания находится в файле под названием test.script.

Поставим в очередь соответствующее задание, используя команду qsub. Листинг 12 содержит пример создания задания.

Листинг 12

master:/home/test #su test
test@master:~> qsub ./test.script
57.master.localdomain
test@master:->

После успешной постановки в очередь задания, torque возвращает идентификатор задания (job identifier), каковым в данном примере является "57.master.localdomain". Формат идентификатора таков:

число.название-сервера.домен

Идентификатор необходим для любого действия, затрагивающего задание, такого как проверка статуса задания, модификации задания, отслеживания или удаления задания.

В приведенном примере задание ставилось в очередь torque; предварительно читались директивы ресурсов, содержащихся в скрипте с заданием. Но существует способ перекрыть (override) атрибуты ресурсов, содержащиеся в скрипте, путем определения их в командной строке. Фактически любая операция постановки задания в очередь или директива, которая определяется в скрипте задания, может быть перекрыта в командной строке че-

рез qsub. Это особенно полезно, если нужно просто поставить в очередь новый экземпляр задания без редактирования скрипта.

Пример содержит Листинг 13.

Листинг 13

```
test@master:-> cat ./test.script
#!/usr/bin/csh
#PBS -d /home/test
#PBS -l ncpus=4
#PBS -N Hostname
/bin/hostname
test@master:-> qsub ./test.script
58.master.localdomain
```

В этом примере значения, равные 16 процессорам и 4 часам времени, перекроют значения, определенные в скрипте задания. Нужно также учитывать, что не требуется использовать ключ –l для каждого ресурса. Можно комбинировать запросы нескольких ресурсов, разделив их запятой. Пример такого сценария содержит Листинг 14.

Листинг 14

```
#!/usr/bin/csh
#PBS -l walltime=1:00:00, mem@0mb
#PBS -l ncpus=4
#PBS -oe
./subrun
```

Команда qsub предлагает различные опции для постановки задания в очередь, которые описаны ниже. Еще раз стоит напомнить, что опции чувствительны к регистру.

Определение учетной записи для задания

Опция –A

Задает строку, описывающую локальную учетную запись, ассоциированную с заданием. Строка, заданная в качестве аргумента, не интерпретируется сервером заданий.

Дата и время выполнения задания

Опция –a дата_время

Опция задает дату и время, когда задание станет доступным для выполнения. Аргумент задается в формате: *[[[[ВВ]ГГ]ММ]ДД]ччмм[.сс]*. Здесь:

- дата: ВВ – первые две цифры года (век); ГГ – последние две цифры года; ММ – две цифры месяца; ДД – две цифры дня месяца;
- время: чч – часы; мм – минуты; сс – секунды.

Квадратные скобки означают, что наличие части аргумента не обязательно. Если не указан месяц, то по умолчанию будет выбран текущий, если задан будущий день. Иначе будет выбран следующий месяц. Если не указан день месяца, то будет выбрана сегодняшняя дата, если указано будущее

время. Иначе будет выбран следующий день. Например: если указано время "1110"(11:10), а на данный момент уже 11:15, то задание будет доступно для запуска на следующий день в 11:10.

Листинг 15 содержит пример применения этой опции.

Листинг 15

```
test@master:-> date
Втр Сен 4 16:16:58 MSD 2007
test@master:-> qsub -a 1700 ./test.script
59.master.localdomain
```

Интерактивные задания

Опция –I

Опция позволяет сделать задание интерактивным. Задание ставится в очередь обычным образом, но при его исполнении стандартные потоки ввода, вывода и ошибок подключаются к терминалу, на котором запущена qsub. Если опция –I указана в командной строке или в директиве внутри сценария, задание становится интерактивным. Если же сценарий набран с клавиатуры, будут обработаны директивы, но исполняемые команды не будут включены в задание.

Когда задание начнет свою работу, все данные, поступающие из потока ввода, будут переданы в терминальную сессию, где работает qsub.

После постановки интерактивного задания в очередь, работа qsub не будет прервана, она будет продолжена до завершения выполнения задания (job terminate), его отмены (job aborted), или принудительного выхода из qsub (Ctrl+C). Если работа qsub будет завершена до запуска задания, появится запрос о выходе из среды. При получении подтверждения, задание не будет выполнено.

При выполнении задания, прерывания от клавиатуры передаются в qsub. Строки, начинающиеся со знака “тильда” () и содержащие специальные последовательности, интерпретируются qsub. Распознаваемые специальные последовательности включают в себя:

- ~. - прерывание выполнения qsub. Работа задания также будет прервана.
- ~susp - приостанавливает выполнение qsub, если она запущена в оболочке C shell. “susp” - специальный символ, обычно - Ctrl+Z.
- ~asusp - приостанавливает часть, отвечающую за ввод в qsub, но вывод разрешен и сообщения продолжают отображаться. Работает также в оболочке C shell. “asusp” – символ дополнительной приостановки (auxiliary suspend), обычно Ctrl+Y.

Перенаправление потоков

Опция –e

Опция –o

Опции перенаправляют вывод, позволяя задать имена файлов, в которые будет перенаправлен стандартный вывод (поток stdout) и помещаться

ошибки (поток stderr). Опция "–o" задается для потока stdout, опция "–e" – для потока stderr.

Аргумент "путь" задается в виде:

[hostname:]path_name

Здесь hostname - имя хоста, path_name – путь на заданном хосте. Допустимы абсолютные и относительные пути.

Пример сценария, выводящего на экран название хоста, поток вывода которого перенаправлен в файл mylog, содержит Листинг 16.

Листинг 16

```
test@master:-> qsub -o ./mylog ./test.script
73.master.localdomain
test@master:-> cat ./test.script
#!/usr/bin/csh
#PBS -d /home/test
#PBS -l ncpus=4
#PBS -N Hostname
/bin/hostname
test@master:-> cat ./mylog
node-32
test@master:->
```

Пауза в работе задания

Опция –h

Пауза в работе задания. Опция переводит задание в состояние пользовательской блокировки в момент постановки в очередь. Опция работает аналогично команде qhold. До тех пор, пока блокировка не будет снята, задание не доступно для выполнения.

Объединение потоков

Опция –j

Опция позволяет объединить стандартный поток вывода задания и его поток ошибок. Аргумент "join" может принимать значения: "oe" – в этом случае поток ошибок stderr будет перенаправлен в поток вывода stdout; "eo" – поток вывода stdout будет перенаправлен в поток ошибок stderr. Если в качестве аргумента указана буква "n" или аргумент опущен, перенаправления не происходит и результат работы двух потоков будет находиться в двух отдельных файлах.

Пример:

```
% qsub –j oe mysubrun
```

Перенаправление потоков на исполняющий узел

Опция –k keep

Опция позволяет перенаправить потоки вывода и ошибок на исполняющий узел. Аргумент может содержать буквы "e" и "o" в любой комбинации, а также букву "n". Значение "e" размещает поток ошибок на испол-

няющем хосте, в домашней папке пользователя, чье задание исполняется. Название файла потока - название_задания.последовательность. Здесь sпоследовательностьї – первая часть идентификатора задания, содержащая числовую последовательность. Пример:

% qsub –k oe mysubrun

Определение ресурсов для задания

Опция –l "выражение"

Аргумент "выражение" опции –l интерпретируется одним из трех способов: либо он обозначает список ресурсов, запрашиваемых для выполнения задания; либо определяет список узлов; либо использует логические выражения для определения ресурсов.

Отсылка по электронной почте

Опции –m опции_отправки, –M список_респондентов Опции настраивают параметры уведомления по электронной почте. Опция s–mї определяет условия, при которых сервер посылает уведомление о выполнении задания. Аргумент «опции_отправки» является строкой, состоящей:

1. либо только из символа "n";
2. либо из одного или более символов: "a", "b", "e".

В первом случае уведомления не отсылаются. Во втором случае буквы определяют условия отсылки: a - прерывание задания (abort); b - начало выполнения задания (begin); e - завершение выполнения задания (end).

Пример:

% qsub –m ae mysubrun

Опция "-M" декларирует список пользователей, кому будут отосланы уведомления. Аргумент для этой опции записывается в виде:

пользователь[@хост][, пользователь[@хост],...]

Если аргумент пустой и задана опция "-m", то уведомления будут отсылаться пользователю- владельцу задания, от чьего имени запущена qsub.

Пример:

% qsub –M james@pbspro.com mysubrun

Изменение названия задания

Опция –N название

Опция определяет название задания. Название должно состоять из печатаемых символов, с первым буквенным символом, пробелы не допускаются. Длина имени не может превышать 15 символов. Если название не указано, то заданию присваивается имя файла сценария, заданное в командной строке. В случае ввода задания с клавиатуры, в консольном режиме, заданию присваивается имя stdin.

Пример:

% qsub –N myName mysubrun

Приоритет задания

Опция –p приоритет

Опция устанавливает приоритет задания. Аргумент является числом от -1024 до 1023 (вклю чительно). По умолчанию задание не имеет приоритета, что эквивалентно установке нулевого значения аргумента. Опция распределяет приоритеты для заданий, которыми владеет текущий пользователь. Следует обратить внимание, что устанавливаемый приоритет служит толь ко ориентиром для планировщика заданий. Планировщик может выбрать свой собственный приоритет.

Пример:

% qsub –p 120 mysubrun

Определение очереди или сервера

Опция –q назначение

Опция определяет очередь или сервер. Эта опция определяет параметры постановки задания в очередь, задавая название очереди, сервера или очереди на сервере. Команда будет передана тому серверу, который указан в аргументе. Если аргумент представляет собой название очереди, задание на сервере перемещается в заданную очередь. Если опция –q не задана, задание ставится в очередь, определенную по умолчанию. Сервер в этом случае также выбирается заданным по умолчанию. Формат аргумента опции таков: [очередь[@хост]].

Листинг 17 содержит пример сценариев, ставящих задание в очередь batch2. Перед этим выводятся параметры очередей batch и batch2.

Листинг 17

```
test@master:-> qmgr -c "list queue batch"
Queue batch
queue_type = Execution
total_jobs = 0
state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
mtime = Tue Sep 4 15:36:09 2007
enabled = True
started = True
test@master:-> qmgr -c "list queue batch2"
Queue batch2
queue_type = Execution
total_jobs = 0
state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
mtime = Tue Sep 4 16:44:09 2007
enabled = True
started = True
test@master:-> qsub -q batch2 ./test.script
66.master.localdomain
test@master:->
```

Оболочка интерпретации сценария

Опция –S список_путей

Опция задает используемую оболочку (shell), которая используется для интерпретации сценария. Аргумент sписок_путей задается в следующем формате: путь[@host][, путь[@host],. . .].

Для одного хоста можно указать только один путь и только один путь можно указать без соответствующего имени хоста. Если опция –S не определена, предполагается, что аргументом является пустая строка, поэтому используется текущая оболочка для пользователя на исполняемом хосте.

Примеры:

% qsub –S /bin/tcsh mysubrun
% qsub –S /bin/tcsh@mars,/usr/bin/tcsh@jupiter mysubrun

Переменные задания

Опция –v список_переменных

Опция задает переменные, которые будут доступны заданию в процессе его выполнению. Имена переменных должны быть разделены запятыми. Переменные и их значения будут переданы заданию после определения их в списке.

Пример:

qsub –v DISPLAY,myvariable2 mysubrun

Зависимые задания

Опция –W список_зависимостей

Опция определяет зависимости между заданиями, или, другими словами, очередность запуска заданий. Листинг 18 содержит пример создания зависимых заданий. Сначала создаются задания с номерами 75 и 76. Затем задание 77 становится зависимым от заданий с номерами 75 и 76 с помощью рассматриваемой опции. После этого используется команда qstart для запуска заданий в очереди batch, а затем запускается задание 75.

Листинг 18

```
master:/home/test # qstop batch
master:/home/test #qsub ./test.script
75.master.localdomain
test@master:-> qsub ./test.script
76.master.localdomain
test@mater:-> qsub -W dependIterok:75:76 ./test.script
test@mater:-> exit
exit
master:/home/test # qstart batch
master:/home/test # qrun 75
master:/home/test #
```

4.3.1.1 Скрипты, запускаемые перед и после выполнения задания

Torque позволяет выполнять скрипты перед запуском задания и после завершения его выполнения - пролог- и эпилог-скрипты. Эти скрипты можно использовать, например, для:

- выполнения инициализации;

- освобождения занятых ресурсов, например, удаления временных директорий, после выполнения;
- записи какой-либо информации в выходной файл задания.

Для запуска пролог- и эпилог-скриптов должны выполняться следующие условия:

1. текст самого сценария должен находиться в директории (pbs_home)/mom_priv, имя файла - prologue для пролог-скрипта и epilogue - для эпилог-скрипта;
2. владельцем должен быть пользователь root;
3. у пользователя root должны быть права на чтение и запуск;
4. прав записи в скрипт не должно ни у кого кроме root.

В качестве скрипта может выступать как сценарий оболочки (shell script), так и исполняемый файл.

Аргументы, передаваемые в пролог- и эпилог-скрипты

Внутри скрипта будут доступны следующие аргументы:

Для пролог- и эпилог-скриптов

argv[1] Идентификатор задания argv[2] Имя пользователя, который запускает задание argv[3] Название группы, из-под которого запускается задание

Только для эпилог-скриптов

argv[4] Название задания
argv[5] Идентификатор сессии (session id)
argv[6] Список требуемых ресурсов
argv[7] Список использованных ресурсов
argv[8] Название очереди
argv[9] Строка учетной записи (account string), если существует

Для скриптов определены также следующие параметры: – рабочей директорией является домашняя папка пользователя; – поток ввода – при запуске потоком ввода является стандартный файл системы, ассоциированный с потоком; – поток вывода – потоки вывода и ошибок скриптов ассоциированы с файлами вывода и ошибок задания. Но если задание является интерактивным, потоки вывода и ошибок указывают на файл /dev/null.

4.3.2 Выполнение программ MPI

Для запуска mpi-приложения в torque используется команда mpirun-ipath-ssh, исполняемая внутри сценария. Ниже приведен простейший пример запуск теста High Performance Linpack через систему пакетной обработки заданий на всех процессорах вычислительного кластера.

Пример:

Тест High Performance Linpack установлен в каталоге /share/hpl, исполняемый файл под данную архитектуру находится в /home/test/hpl. Скрипт run.test находится в том же каталоге и выглядит следующим образом:

#bin/bash

NP=‘cat ${PBS_NODEFILE} | wc -l‘ mpirun-ipath-ssh -np ${NP} -m ${PBS_NODEFILE} ./xhpl

Запуск скрипта осуществляется по команде

“qsub -l nodes#:ppn=2 -d /home/test/hpl /home/test/hpl/run.test”

В результате работы команды в очередь на выполнение будет поставлена задача со следующими параметрами:

Количество требуемых ресурсов: 16 узлов по 2 процессора на каждом;
Рабочая директория задачи: /home/test/hpl
Исполняемый файл: /home/test/hpl/run.test

4.3.3 Удаление заданий. Команда qdel

В системе torque имеется команда qdel для удаления заданий. Эта команда удаляет задания в том порядке, в котором указаны их идентификаторы в списке параметров. Удаленное задание более не будет управляться torque. Задание может быть удалено владельцем, оператором или администратором torque. Команда qdel является одной из причин, по которой задание может быть удалено. Другими причинами являются:

- превышение допустимого предела используемых ресурсов;
- завершение работы сервера заданий.

При этом задания удаляются автоматически, без вмешательства пользователя.

Пример командного удаления содержит Листинг 19. В этом примере впервые применена команда qstat, которая выводит информацию заданиях.

Листинг 19

```
test@master:-> qstat
Job id   Name              User            Time Use  S  Queue
-------- ----------------- --------------- -------- - -----
78.master SuperEngine      test                    0  Q  batch
test@master:-> qdel 78
test@master:-> qstat
test@master:->
```

4.3.4 Изменение атрибутов задания. Команда qalter

Команда qalter применяется для модификации атрибутов задания.
Синтаксис команды qalter таков:
qalter атрибуты список_заданий
Здесь «атрибуты» эквивалентны опциям команды qsub, они модифицируются соответствующими значениями. Если в списке присутствует один или несколько атрибутов, которые не могут быть изменены либо динамически, либо по другой причине, то не изменятся и все остальные атрибуты.

Листинг 20 содержит пример использования команды qalter. Происходит модификация наименования задания.

Листинг 20

```
master:/home/test # qstop batch
master:/home/test # su test
test@master:-> qsub ./test.script
78.master.localdomain
test@master:-> qalter -N SuperEngine 78
```

4.3.5 Изменение состояния заданий. Команды qhold и qrls

Система torque поддерживает две команды, работающих в паре, позволяющих «блокировать» (hold) и «восстановить» (release) задание. Блокировка задания, или установка его состояния в "hold" означает, что задание не может быть выполнено, пока оно не будет восстановлено (release), т.е. метка "hold" не будет снята соответствующей командой.

Команда qhold выдает серверу запрос на установку одной или нескольких меток "hold" на одно или несколько заданий. Блокированное задание не может быть выполнено. Имеется три типа блокировки: пользовательская (user), операторская (operator) и системная (system).

Пользователь может установить пользовательскую блокировку на любое задание, которым он владеет. Оператор, который является пользователем с особыми sоператорскимиї привилегиями, может устанавливать пользовательскую или операторскую блокировку. Пользователь с правами менеджера может устанавливать любой тип блокировки.

4.3.5.1 Команда qhold

Синтаксис команды qhold таков:

qhold [-h hold_list | job_identifier ...

Параметр hold_list определяет тип блокировок, устанавливаемых для задания. Этот аргумент является строкой, состоящей из кратких обозначений одного или нескольких типов блокировок в любой комбинации: n (none) - нет блокировки; u (user) - пользовательская; o (operator) - операторская; s (system) - системная.

Если опция -h не указана, устанавливается пользовательская блокировка ко всем заданиям с указанными в списке job_identifier идентификаторами.

Если задание, на который указывает один из идентификаторов в списке job_identifier, уже поставлено в очередь, блокировано или находится в одном из состояний ожидания (waiting states), то все, что происходит - это добавление метки "hold" к заданию. Затем задание устанавливается в заблокированное состояние, если оно находится в очереди на исполнение.

Если же задание уже выполняется, то дополнительно происходит прерывание задания. Если операционной системой поддерживаются контрольные точки (checkpoint) или перезапуск (restart), запрос на блокировку задания вызывает следующее:

- задание устанавливается в состояние контрольной точки (checkpointed);
- ресурсы, ассоциированные с заданием очищаются;
- задание устанавливается в блокированное состояние в очереди на выполнение.

В случае если операционная система не поддерживает контрольные точки или перезапуск, команда qhold только запрашивает состояние блокировки. Таким образом, никакого эффекта не будет до тех пор, пока задание не будет перезапущено (командой qrerun).

4.3.5.2 Команда qrls

Команда qrls снимает блокировку с задания. Тем не менее, пользователь, который выполняет эту команду, должен обладать необходимыми привилегиями, чтобы снять данную блокировку.

Правила, действующие для установки блокировок, аналогично действуют и для их снятия.

Синтаксис команды:

qrls [-h hold_list] job_identifier ...

Листинг 21 содержит пример, который демонстрирует, как используется команды qhold и qrls.

Листинг 21

```
test@master:-> qsub ./test.script
79.master.localdomain
test@master:-> qstat
Job id   Name              User            Time Use  S  Queue
-------- ----------------- --------------- -------- - -----
79.master      HostName          test                 0  Q      batch
test@master:-> qhold -h u 79
test@master:-> qstat
Job id   Name              User            Time Use  S  Queue
-------- ----------------- --------------- -------- - -----
79.master      HostName          test                 0  H      batch
test@master:-> qrls 79
test@master:-> qstat
Job id   Name              User            Time Use  S  Queue
-------- ----------------- --------------- -------- - -----
79.master      HostName          test                 0  Q      batch
test@master:->
```

4.3.6 Информация о заданиях. Команда qstat

Для получения информации о заданиях и сервере заданий имеется команда qstat. Запрашиваемая информация выводится в стандартный поток вывода. При запросе состояния задания, на которое пользователь не имеет прав (привилегий), это состояние отображено не будет.

4.3.6.1 Стандартная информация о заданиях

Выполнение qstat без каких-либо опций отображает информацию о заданиях в формате по умолчанию. Отображается следующая информация:

- идентификатор задания, присвоенный системой;
- название задания, присвоенное инициатором задания;
- владелец задания;
- используемое процессорное время;
- состояние (статус) задания;
- очередь, в которой находится задания.

Пример вывода стандартной информации о задании содержит, например, Листинг 21.

4.3.6.2 Расширенная информация о заданиях

Если задать опцию "-a" для команды qstat, то будет для задания будет отображена следующая информация (в дополнение к основной):

- идентификатор сессии (Session ID);
- требуемое количество узлов;
- число параллельных задач (tasks);
- требуемый объем памяти;
- требуемое количество времени;
- количество времени, которое задание находится в текущем состоянии.

Пример вывода расширенной информации о задании приводит Листинг 22.

Листинг 22

```
test@master:-> qsub ./test.script
86.master.localdomain
test@master:-> qsub ./test.script
87.master.localdomain
test@master:-> qsub ./test.script
88.master.localdomain
test@master:-> qstat -a
```

Job ID	Username	Queue	Jobname	SessID	NDS	Req'd TSK	Req'd Memory	Elap Time	S	Time
86.master.localdomai	test	batch	Hostname --	--	--	4	--	--	Q	
87.master.localdomai	test	batch	Hostname --	--	--	4	--	--	Q	
88.master.localdomai	test	batch	Hostname --	--	--	4	--	--	Q	

5 Общие рекомендации администраторам кластеров

5.1 Автоматизация типовых задач администрирования кластера

Использование скриптов в консоли и прочая автоматизация – вот основной инструментарий администратора Linux. Применение скриптов обеспечивает две полезные вещи [7]:

- Первое и самое очевидное – это экономит работу по вводу с клавиатуры и позволяет производить повторяющиеся действия.
- Второе – это самодокументируемость и возможность позднейших модификаций.

У опытных администраторов имеются каталоги, заполненные написанными ими собственноручно скриптами. Эти скрипты делают все: от проверки версий встроенного программного обеспечения на узлах до преобразования GUID в кластере на основе InfiniBand.

Примером, где применение скриптов весьма уместно, является создание образа операционной системы, будь то изменяемый образ (stateful) или неизменяемый (stateless). Если у администратора есть "золотой образ", который необходимо скопировать на каждый вычислительный узел в системе, он должен знать его содержимое. Скрипт, создающий этот образ – лучшая доступная документация, поскольку чётко объясняет то, что делает, и его можно запускать повторно. Без скриптов для их создания образы неконтролируемо размножаются, съедая дисковое пространство и замедляя работу системы.

5.2 Использование средств удаленного администрирования

По мере повышения затрат на энергию/охлаждение/обслуживание растут тенденции по перемещению вычислительных центров в менее дорогие в эксплуатации и, соответственно, менее комфортные для пребывания человека помещения. Ввиду этого наличие удалённого управления следует расценивать как необходимую основу для управления кластером.

Поставщики оборудования в значительной степени учли пожелания клиентов касательно стандартов удалённого управления системами. IPMI 2.0 стал текущим стандартом для большинства управляемых Linux-кластеров. IPMI дает возможность удалённо включать и отключать питание машины, а также предоставляет удаленный терминал для просмотра начальной загрузки машины из BIOS.

IPMI – необычайно мощный инструмент. Он позволяет удаленно изменять параметры настройки BIOS, перезагружать вычислительные узлы и т.п. Минимальные требования здесь следующие:

- Возможность удаленного включения машин.

- Возможность наблюдения за возможными проблемами при загрузке кластера.

Можно рекомендовать стандартные инструментальные средства с открытым исходным кодом, такие, как ipmitool, входящие в состав большинства дистрибутивов Linux.

5.3 Тщательный выбор кластерного программного обеспечения

Редко когда вычислительная среда кластера бывает настолько уникальной, что никакие инструментальные средства с открытыми исходными кодами не подходят для работы с ней. Среди самых распространенных программ – OSCAR (клонирование систем), ROCKS, Perceus, xCAT 2.

Возможно, самое популярное на сегодня решение по развертыванию/управлению кластерами среди программ с открытыми исходными кодами – это ROCKS. ROCKS разрабатывается и поддерживается в Калифорнийском университете Сан-Диего (UCSD).

Ещё одно подобное решение – Perceus. Отличие его от ROCKS, в том, что это – не "stateless" инсталляция. В контексте данного раздела "stateless" означает операционную систему, работающую только в памяти, без записи на диск. Диск не требуется, но может использоваться как локальное хранилище или как файл подкачки.

Также интересен xCAT, который активно разрабатывается с 31 октября 1999 года, а его исходные тексты были открыты IBM в 2007 году.

5.4 Использование программных средств мониторинга кластера

Вот, пожалуй, наиболее популярные инструментальные средства мониторинга с открытыми исходными кодами, применяемые при работе с масштабируемыми кластерами:

- Nagios.
- Ganglia.
- Cacti.
- Zenoss.
- CluMon.

Отметим некоторые особенности мониторинга кластеров:

- Для мониторинга нет универсального "безразмерного" решения. Под мониторингом разные люди понимают под мониторингом разные вещи. Некоторых интересует только система оповещений, других – показатели производительности, других третьих – профилирование заданий, а другим четвертым только нужно знать, запущены ли определенные службы или приложения.
- Настройка будет различной в различных учреждениях и на различных кластерах в пределах одного и того же учреждения.
- Нужен взвешенный компромисс между тем, что мониторится, для чего этот мониторинг используется и необходимыми ресурсами.

5.5 Использование планировщиков заданий

При использовании кластера важно гарантировать, чтобы у пользователей не было административных привилегий или прав доступа. Планировщики обеспечивают эту функциональность: пользователь представляет задание, а планировщик очереди решает, на каких узлах оно будет выполняться.

Среди популярных сегодня планировщиков есть платные программы, такие как LSF и PBS Pro. Эти продукты используются многими коммерческими заказчиками, а также правительственными лабораториями и университетами. Но для многих систем прекрасно подходят решения с полностью открытым исходным кодом, такие как TORQUE и SLURM.

5.6 Необходимость тестирования производительности

Аппаратная диагностика это, как правило, система удачных либо неудачных прохождений тестов, с порогом допустимости, определенным производителем.

Вот некоторые проблемы, которые могут оказывать заметное отрицательное влияние на производительность систем, даже успешно прошедших диагностику производителя:

- Двухбитовые ошибки памяти.
- Однобитовые ошибки памяти.
- Ошибки четности SRAM.
- Ошибки шины PCI.
- Численные ошибки.
- Ошибки кэша.
- Дисковые ошибки.
- Неустойчивая производительность

Часто проблемы не имеют отношения к аппаратным средствам, а возникают по вине программного обеспечения. Приложения, библиотеки, компиляторы, встроенное программное обеспечение и любая часть операционной системы могут служить источниками множества проблем, которые аппаратная диагностика обнаружить не в состоянии. Аппаратная диагностика часто проводится в другой среде выполнения, чем приложения, и не нагружает подсистемы так, как приложения – поэтому проблемы, создаваемые приложениями, пропускаются.

Понятно, что для того, чтобы удостовериться, что кластер работает исправно, нужна некоторая адекватная рабочая нагрузка. Для этого можно запустить некоторые из стандартных промышленных тестов. Цель при этом не в том, чтобы получить наилучшие результаты, а чтобы получить результаты стабильные, повторяемые и аккуратные, что и является лучшим результатом.

Как понять, являются ли полученные результаты лучшими? Кластер можно условно разбить на следующие главные подсистемы:

- Память.

- Центральный процессор.
- Диск.
- Сеть.

У поставщика кластера должны быть в наличии контрольные данные тестирования, фиксирующие ожидаемую производительность памяти, ЦП (количество операций с плавающей запятой в секунду), диска и сети.

Как получить стабильные результаты? Путем статистического усреднения. Каждый тест запускается один или несколько раз на каждом узле (или на нескольких узлах, в случае соответствующих тестов), а затем средние показатели для каждого узла (или нескольких узлов) группируются и анализируются как единая совокупность. Интересны не столько результаты сами по себе, как форма их распределения. Согласно эмпирическому опыту результаты должны формировать нормальное распределение. Нормальное распределение – это классическая колоколообразная кривая, так часто встречающаяся в статистике. Оно возникает как результат суммарного воздействия небольших, независимых (может быть, неразличимых), одинаково распределенных переменных или случайных событий.

В тестах производительности также содержится много маленьких независимых (может быть неразличимых), одинаково распределенных переменных, которые могут повлиять на производительность. Это, например:

- Небольшие конкурирующие процессы.
- Переключение контекста.
- Аппаратные прерывания.
- Программные прерывания.
- Распределение памяти.
- Планирование процесса/потока.

Их присутствия нельзя избежать и все они вносят свой вклад в формирование нормального распределения.

Тесты производительности могут также иметь нетождественно распределенные наблюдаемые переменные, которые могут влиять на производительность:

- Значительные конкурирующие процессы.
- Конфигурация памяти.
- Параметры настройки и версия BIOS.
- Скорость процессора.
- Операционная система.
- Тип ядра (например, NUMA либо SMP либо однопроцессорное) и его версия.
- Плохая память (чрезмерное число исправлений ошибок).
- Версии чипсетов.
- Гиперпоточность или SMT-многопоточность.
- Неоднородные конкурирующие процессы (такие, как httpd, работающий на некоторых узлах, но не на всех).
- Версии общих библиотек.

Присутствия этих переменных можно избежать. Преодолимые несогласованности могут привести к возникновению многомодального или отличного от нормального распределения и могут оказать измеримое воздействие на производительность приложения.

Чтобы получить стабильные, повторяемые, точные результаты лучше начать с возможно меньшим набором переменных. Можно начать с теста для одного узла, такого, как STREAM. Если на всех машинах результаты STREAM будут одинаковые, то при аномалиях с другими тестами, память как фактор нестабильности может быть исключена. Затем можно переходить к тестам процессора и диска, затем к тестам для двух узлов (параллельным), затем для нескольких узлов (параллельным). После каждого более сложного теста, прежде чем продолжать, необходимо проверять результаты на стабильность, повторяемость и точность.

Вот примерный набросок последовательности тестов, которую можно рекомендовать (тест проверяет производительность компонента, выделенного **жирным шрифтом**):

- Тесты для одного узла (последовательные):
    1. STREAM (**память Мбит/с**)
    2. NPB Serial (**однопроцессорная производительность с плавающей запятой** и память)
    3. NPB OpenMP (**многопроцессорная производительность с плавающей запятой** и память)
    4. HPL MPI Shared Memory (**многопроцессорная производительность с плавающей запятой** и память)
    5. IOzone (**дисковый ввод/вывод Мбит/с**, память и процессор)
- Параллельные (только для систем MPI):
    1. Ping-Pong (**соединения – задержки в микросекундах** и **пропускная способность в Мбит/с**)
    2. NAS Parallel (**производительность с плавающей запятой**, память и соединения для узлов)
    3. HPL MPI Distributed Memory (**производительность с плавающей запятой**, память и соединения для узлов)

5.7 Организация обменом информацией

Современным средством обмена информацией по кластерной системе является пакет Media Wiki.

Media Wiki может содержать ответы на все задаваемые вопросы и всю доступную информацию, относящуюся к кластеру, например:

- Журнал административных изменений в системе, включая сведения о том, когда они выполнялись.
- Опись кластера: версии программно-аппаратного обеспечения, модели, серийные номера и число узлов в кластере, тип процессора, памяти.

- Ссылки на открытые заявки на поддержку поставщику и источники новостей по этому вопросу.
- Документация для стандартных задач управления, выполняемых с использованием программного обеспечения.
- Информация на том, как создаются образы ОС.

Чем вики лучше других форм документации?

- Одна из основных причин – вики можно редактировать и управлять доступом к ней. Мы видели способы хранения документации в совместно используемом хранилище, но чтобы просмотреть документ, необходимо было попасть в репозиторий, сохранить документ на жёсткий диск, и только затем открыть его. Одна-единственная ссылка – это гораздо более быстрый и безболезненный способ.
- Излагать информацию в виде документа Word или в электронной таблице – статичный способ, а также довольно трудный для редактирования, повторного сохранения и рассылки версий. Не говоря уже о том, что нам попадались множественные копии одного документа, и люди не имели понятия о том, какая из версий является самой свежей. Особенно если документ редактируется не одним человеком. По опыту, «живая» документация, помещённая в вики, живёт дольше.

Синтаксис Wiki намного проще, чем синтаксис HTML, и в Интернете есть много полезной информации о том, как с ним работать. Также существует множество полезных расширений для подсветки синтаксиса perl или bash, если вы используете их.

5.8 Тенденции развития систем управления кластерами

Сегодня в мире управления кластерами на основе Linux происходят большие изменения. Отметим, пожалуй, самые интересными из них:

1. Большие подвижки в сторону "stateless computing", что облегчает администрирование образов и синхронизацию узлов.
2. Взгляд на кластеры с точки зрения вычислительных центров, что означает учет оплаты энергии и охлаждения на три года вперёд, в противоположность подсчитыванию только стоимости приобретения.
3. Эффективность жидкостного охлаждения в противоположность воздушному. Знаете ли вы, что жидкостное охлаждение на 90% эффективнее воздушного?
4. Гибкое управление. Именно здесь планировщик может предоставлять узлы по запросу. Это путь к подлинному cloud computing.

Приложение 1. Обзор необходимых команд Linux.

Ниже приводятся некоторые наиболее употребляемые команды Linux. Большинство этих команд можно выполнить на управляющем узле с помощью MidnightCommander. Однако на вычислительных узлах MidnightCommander, как правило, отсутствует.

Чтобы получить более полную информацию по любой отдельной команде command, нужно ввести

```
man command
```

Выход из описания команды производится при нажатии клавиши «q».

Работа с каталогами

pwd – показывает название текущей директории;

cd dir – устанавливает текущим каталогом каталог с именем dir, вызов команды cd без параметров возвращает в домашний каталог /home/username ($HOME);

mkdir subdir – создает новый подкаталог с именем subdir;

rmdir subdir – удаляет пустой подкаталог с именем subdir;

ls – показывает список файлов и подкаталогов текущей директории,

ls dir – показывает список файлов и подкаталогов каталога dir;

ls -A – показывает все файлы, в том числе и скрытые;

ls -l – показывает атрибуты (владелец, разрешение на доступ, размер файла и время последней модификации);

mv oldname newname – изменяет имя подкаталога или перемещает его;

cp -R dirname destination – копирует подкаталог dirname в другое место destination.

Работа с файлами

file filename(s) – определяет тип файла (например, ASCII , JPEG image data и др.);

cat filename(s) – показывает содержание файлов (используется только для текстовых файлов!);

more filename(s) – действует так же, как и cat, но позволяет листать страницы;

less filename(s) – улучшенный вариант команды more;

head filename – показывает первые десять строк файла filename;

tail filename – показывает последние десять строк файла filename;

wc filename(s) – показывает число строк, слов и байт для указанного файла;

rm filename(s) – уничтожает файлы или директории, для рекурсивного удаления следует использовать rm с ключом -rf.

cp filename newname – создает копии файлов с новыми именами;

cp filename(s) dir – копирует один или более файлов в другой каталог;
mv oldname newname – изменяет имя файла или каталога;
mv filename(s) dir – перемещает один или более файлов в другой каталог;
find dir -name filename – пытается локализовать файл (подкаталог) filename рекурсивно в подкаталоге dir.

Другие полезные команды
passwd – изменяет пароль пользователя системы Linux; требует подтверждения старого;
who – показывает, кто в настоящее время работает в сети;
finger – дает более подробную информацию о пользователях сети;
write – позволяет послать сообщение пользователю, работающему в сети в данное время;
top – отображает информацию о процессах, использующих процессоры узла;
ps -U user_name – показывает номера процессов(pid), инициированных пользователем user_name;
kill xxxxx – досрочно завершает работы процесса с номером xxxxx;
killall proc_name – досрочно завершает работу процесса proc_name;
date – отображает дату и время;
cal – показывает календарь.
exit – выйти из терминала
clear – очистить окно терминала
du dir – показывает занятое место в директории dir .

Приложение 2. Примеры PBS скриптов

Подробно о возможностях PBS Torque можно узнать из руководства пользователя.

Ниже приведены несколько примеров использования Torque.

```
#PBS -o $DIR/stdout.log
```

Определяет имя файла, в который будет перенаправлен стандартный поток stdout

```
#PBS -e $DIR/stderr.log
```

Определяет имя файла, в который будет перенаправлен стандартный поток stderr

```
#PBS -l nodes=8:ppn=2:cpp=1
```

Определяет какое количество узлов и процессоров на них необходимо задействовать.

nodes – количество узлов

ppn – число процессоров на узле

cpp – число процессов на процессоре

```
#PBS -l walltime=20:00:00
```

Определяет максимальное время счета задания

```
#PBS -l mem=1000mb
```

Определяет количество необходимой оперативной памяти

```
cat $PBS_NODEFILE | grep -v master | sort | uniq -c |
awk '{printf "%s:%s\n", $2, $1}' >
$PBS_O_WORKDIR/temp.tmp
```

Составляет список узлов в необходимом формате, на которых будет запущена задача и записывает их в файл temp.tmp

```
cd $PBS_O_WORKDIR
/usr/bin/mpirun -m temp.tmp -np 100 ./a.out
```

Запускает на узлах указанных в файле temp.tmp задачу 100 раз.

Пример скрипта:

```
#PBS -o $DIR/stdout.log
#PBS -e $DIR/stderr.log
#PBS -l nodes=50:ppn=2
#PBS -l walltime=20:00:00
#PBS -l mem=1000mb
cat $PBS_NODEFILE | grep -v master | sort | uniq -c |
awk '{printf "%s:%s\n", $2, $1}' >
$PBS_O_WORKDIR/script1.temp.sh.mf
cd $PBS_O_WORKDIR
/usr/bin/mpirun -m script1.temp.sh.mf -np 100 ./a.out
```

Здесь будет запущена параллельная программа a.out на 50 узлах, с каждого узла будет использоваться 2 процессора. Файл вывода стандартного потока stdout – stdout.log, стандартного потока stderr – stderr.log.

$DIR содержит путь к файлам stdout.log и stderr.log, например может принимать значение /home/user_name. Под задачу отведено 20 часов. Необходимое количество памяти 1000 мегабайт.

Для запуска последовательной программы first можно использовать следующий скрипт:

```
#PBS -o $DIR/stdout.log
#PBS -e $DIR/stderr.log
#PBS -l walltime=10:00
#PBS -l mem=100mb
./first
```

При запуске программы через команду qsub заданию присваивается уникальный целочисленный идентификатор.

qdel – утилита для удаления задачи.

В случае, если задача уже запущена, процесс ее работы будет прерван. Синтаксис данной утилиты следующий:

qdel [-W время задержки]идентификатор задачи

Выполнение такой команды удалит задачи с заданными идентификаторами через указанное время. Если часть вычислительных узлов, на которых выполнялась задача, недоступны, то принудительно удалить ее с сервера можно путем добавления ключа -p.

Приложение 3. Переменные окружения планировщика Torque

Далее приводится список переменных окружения Torque и примеры их значений.

```
PBS_JOBNAME=env
PBS_ENVIRONMENT=PBS_BATCH
PBS_O_WORKDIR=/home/test
PBS_TASKNUM=1
PBS_O_HOME=/home/test
PBS_MOMPORT=15003
PBS_O_QUEUE=batch
PBS_O_LOGNAME=test
PBS_O_LANG=en_US.UTF-8
PBS_JOBCOOKIE=3088939E7FAA7F4414578D7A806955
PBS_NODENUM=0
PBS_O_SHELL=/bin/bash
PBS_JOBID=93.master.localdomain
PBS_O_HOST=master.localdomain
PBS_VNODENUM=0
PBS_QUEUE=batch
PBS_O_MAIL=/var/spool/mail/test
PBS_O_PATH=/home/test/bin:/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:
/usr/games:/opt/gnome/bin:/opt/kde3/bin:
/usr/lib/mit/bin:/usr/lib/mit/sbin
```

Список литературы

[1] http://cluster.linux-ekb.info/

[2] http://www.ibm.com/developerworks/ru/library/l-ganglia-nagios-1/#resources

[3] http://www.intel.com/design/servers/ipmi/ipmi.htm

[4] http://www.clusterresources.com/torquedocs/index.shtml

[5] http://www.clusterresources.com/torquedocs21/usersmanual.shtml

[6] http://supercomputer.susu.ru/users/instructions/torque_manual.pdf

[7] http://www.ibm.com/developerworks/linux/library/l-11sysadtips/index.html

Учебное издание

Казанский Николай Львович
Попов Сергей Владимирович
Серафимович Павел Григорьевич

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА НА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ СИСТЕМАХ

Учебное пособие

Компьютерная верстка Я.Е. Тахтаров

Подписано в печать 15.12.2010 г. Формат 60×84 1/16.
Бумага офсетная. Печать офсетная.
Печ. л. 4,62. Тираж 100 экз. Заказ 111.

Учреждение Российской академии наук Институт систем обработки изображений РАН,
Самарский государственный аэрокосмический университет имени академика С.П.Королёва
(национальный исследовательский университет)

Издательство «ВЕК#21»
443099 Самара, Чапаевская 69 а.