Entornos de ejecución de computación paralela

Paralelismo y Sistemas Distribuidos

Grado en ciencia e ingeniería de datos

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

Licencia

https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es

Ejecución tradicional HPC

- Acceso al Sistema
  - Nodos de “login”
  - Nodos de computación
- Configuración del entorno de trabajo
  - Entorno de usuario: Shell , variables de entorno básicas, transferencias de ficheros
  - Módulos de usuario para seleccionar herramientas, versiones especificas de librerías etc
  - Directorios de trabajo
  - Compilación
  - Ejecución en colas

Acceso al sistema

- Los centros de supercomputación tienen, por lo menos, dos tipos de nodos:
  - Nodos de login
  - Nodos de computación
- Nodos de login
  - Son los nodos a los cuales tenemos acceso desde la red externa
  - Se permite la conexión utilizando programas conocidos como **ssh**, putty, etc.
  - Se puede hacer transferencia de datos a/desde estos nodos
  - En muchos centros, por temas de seguridad, no se pueden utilizar herramientas de desarrollo de software como git
  - Normalmente, tienen un software y hardware diferente a lo disponible en los nodos de computo
  - Se utilizan para compilar y hacer test muy básicos que no requieran ni paralelismo ni recursos en exclusiva

Nodos de login

- En el caso de MareNostrum IV tenemos nodos de login igual a nodos de computación
  - Nodos de login: Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz
  - Tres nodos: mn1.bsc.es, mn2.bsc.es, mn3.bsc.es
  - Acceso mediante ssh, putty (cualquier cliente compatible con ssh v2 puede usarse)
  - Se pueden ejecutar aplicaciones graficas y visualizarlas en local mediante la utilizaciones de 'X-servers', por ejemplo
    - Cygwin/X: http://x.cygwin.com (open source)
    - X-Win32 : http://www.starnet.com
  - Se puede hacer transferencia de ficheros mediante scp (secure copy) o sftp (secure ftp) pero siempre conectado desde local a MareNostrum
    - las conexiones desde MareNostrum al exterior no están permitidas

Nodos de login

- Ejemplos (comandos a ejecutar desde vuestro PC o portátil)
  - localsystem$ ssh username@mn1.bsc.es
  - Para copiar un fichero que esta en MN a vuestro PC (portíatil)
    - localsystem$ scp username@mn1.bsc.es:path_origen path_destino_local
    - localsystem$ sftp username@mn1.bsc.es
      - username's password:
      - sftp> get remotefile
  - Para copiar de vuestro PC (portátil) a MareNostrum
    - localsystem$ scp path_origen_local username@mn1.bsc.es:path_destino
    - localsystem$ sftp username@mn1.bsc.es
      - username's password:
      - sftp> put localfile

Nodos de cómputo

- Peak Performance of 11.15 Petaflops
- 384.75 TB of main memory
- 3,456 nodes:
  - 2x Intel Xeon Platinum 8160 24C at 2.1 GHz
  - 216 nodes with 12x32 GB DDR4-2667 DIMMS (8GB/core)
  - 3240 nodes with 12x8 GB DDR4-2667 DIMMS (2GB/core)
- Interconnection networks:
  - 100Gb Intel Omni-Path Full-Fat Tree
  - 10Gb Ethernet
- Operating System: SUSE Linux Enterprise Server 12 SP2

Configuración del entorno de usuario

- Un usuario nuevo tiene un contexto (o entorno) básico
- El contexto de un usuario se define por el valor de las **variables de entorno** y **límites de usuario**
- Los límites de usuario los define el administrador y normalmente no se pueden cambiar por el usuario (hay que solicitarlo)

```
nct00018@login2:~> ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 379109
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 100000
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 2097152
cpu time               (seconds, -t) 300
max user processes              (-u) 379109
virtual memory          (kbytes, -v) 8388608
file locks                      (-x) unlimited
```

Configuración del entorno de usuario

- Variables de entorno
  - Son variables que consultan las herramientas de sistema (+librerías etc) para configurar su comportamiento
  - Se definen desde el intérprete de comandos pero las puedes leer/escribir cualquier software
- Definición/consulta/exportar a otros programas desde bash

```
snct00018@login1:~> cat show_env.sh
#!/bin/bash
echo $MI_VARIABLE
nct00018@login1:~> chmod u+x show_env.sh
nct00018@login1:~> MI_VARIABLE=hola
nct00018@login1:~> ./show_env.sh
nct00018@login1:~> export MI_VARIABLE=hola
nct00018@login1:~> ./show_env.sh
hola
```

Muestra el valor

Define la variable

Define y exporta la variable

Ejemplos: bash

- Variable PATH
  - PATH: lista ordenada de directorios en los cuales buscar los ejecutables

```
nct00018@login1:~> echo $PATH
/apps/modules/bsc/bin:/apps/INTEL/2017.4/impi/2017.3.196/bin64:/apps/INT
EL/2017.4/bin:/usr/local/bin:/usr/bin:/bin:/usr/games:/usr/lpp/mmfs/bin:
/home/nct00/nct00018/.local/bin:/home/nct00/nct00018/binnct00018@login1:
nct00018@login1:~> ls -l show_env.sh
-rwxr--r-- 1 nct00018 nct00 30 feb  7 08:01 show_env.sh
nct00018@login1:~> show_env.sh
-bash: show_env.sh: command not found
nct00018@login1:~> ./show_env.sh
nct00018@login1:~> export PATH=$PATH:.
nct00018@login1:~> show_env.sh
nct00018@login1:~>
```

- El commando “env” muestra todas las variables definidas

Ejemplos:bash

- LD_LIBRARY_PATH: lista ordenada de directorios en los cuales el “loader” busca las librerías dinámicas
  - Sólo aplica a binarios, no scripts (ficheros de texto con comandos que interpreta, no es código ejecutable)

```
nct00018@login1:~> echo $LD_LIBRARY_PATH
/apps/INTEL/2017.4/mkl/lib/intel64:/apps/INTEL/2017.4/impi/2017.3
.196/lib64:/apps/INTEL/2017.4/lib/intel64
```

- HOME: directorio raíz del usuario. Podemos usarlo para hacer referencia a nuestra carpeta (o directorio) raíz

```
nct00018@login1:~> echo $HOME
/home/nct00/nct00018
```

- USER: nuestro nombre de usuario

```
nct00018@login1:~> echo $USER
nct00018
```

Ejemplos intel MPI

- https://software.intel.com/en-us/mpi-developer-reference-linux-environment-variable-reference
- Para seleccionar el/los compiladores

```
nct00018@login1:~> env | grep I_MPI
I_MPI_F77=ifort
I_MPI_F90=ifort
I_MPI_CC=icc
I_MPI_CXX=icpc
I_MPI_ROOT=/apps/INTEL/2017.4/impi/2017.3.196
```

- Utilizadas al empezar el programa
  - I_MPI_HYDRA_HOST_FILE: proporciona una lista de hosts específica
  - I_MPI_HYDRA_ENV: controla la propagación de variables de entorno
  - I_MPI_HYDRA_BOOTSTRAP: indica como crear los procesos
  - I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS, permite pasar parámetros al bootstrap server
    - Ej: I_MPI_HYDRA_BOOTSTRAP=slurm I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARG="slurm args"

Ejemplos intel MPI

- Variables que modifican la selección de recursos o la creación de procesos
  - **I_MPI_DEBUG** : selecciona el nivel de detalle que muestra la librería de Intel MPI sobre la ejecución de la aplicación (def=0)
  - I_MPI_DEBUG_OUTPUT: indica donde mostrar el output (def=stdout)
  - Variables relacionadas con el *pinning* de procesos a procesadores
    - Pinning es “atar” procesos (o threads) en procesadores

Ejemplos OpenMP

- Configura el comportamiento de la librería de OpenMP
  - **OMP_NUM_THREADS**= define el máximo número de threads a usar por cada ”parallel region”
  - OMP_SCHEDULE: define el “scheduling” de bucles a utilizar en tiempo de ejecución. Muy útil si la aplicación no tiene un patrón de acceso a datos constante y conocido
- Variables de la librería de Intel KMP: empiezan con KMP
  - KMP_AFFINITY: define el criterio de afinidad a la hora de atar threads en procesadores → puede entrar en conflicto con MPI!!

Configuración del entorno: Módulos de usuario

- El entorno de trabajo en sistemas como MareNostrum incluye una configuración básica que se extiende mediante la carga de ”módulos de usuario”
- Un modulo es un fichero que modifica algunas variables de entorno para dar visibilidad a un software que está instalado en el sistema
  - PATH
  - MANPATH
  - LD_LIBRARY_PATH
  - Variables de entorno que necesita el software en concreto
- Los módulos tienen varios comandos sencillos para:
  - Ver que módulos hay: **module avail**
  - Cargar un módulo: **module load *module_name***
  - Ver que módulos tengo cargados: **module list**
  - Eliminar un módulo: **module unload *module_name***
  - Eliminar todos los módulos que tengo cargado: **module purge**
  - Ver la info de un módulo: **module whatis *module_name***

Módulos cargados

Ejemplo

```
nct00018@login2:~> module list
Currently Loaded Modules:
  1) intel/2017.4   2) impi/2017.4   3) mkl/2017.4   4) bsc/1.0

nct00018@login2:~> module avail
/apps/modules/modulefiles/compilers ------------------------------------------------
---------
   gcc/4.8.5    gcc/7.1.0        go/1.9.2             intel/2017.6    intel/2018.1
intel/2018.4    java/7u80         java/8u151
   gcc/4.9.4    gcc/7.2.0 (D)    intel/2017.1         intel/2017.7    intel/2018.2
java/latest     java/8u131 (D)
   gcc/5.4.0    gcc/8.1.0        intel/2017.4 (L,D)   intel/2018.0    intel/2018.3
java/6u45       java/8u144

/apps/modules/modulefiles/environment ----------------------------------------------
----------
   bsc/1.0      (L)    impi/2017.1        impi/2018.0    impi/2018.4
openmpi/1.10.4        openmpi/3.0.0    transfer/1.0
   fabric/1.4.2 (D)    impi/2017.4 (D)    impi/2018.1    tmod/icase
openmpi/1.10.7 (D)    openmpi/3.1.1
   fabric/1.5.0        impi/2017.6        impi/2018.2    mvapich2/2.3b
openmpi/2.0.2         opt/flags
   fabric/1.5.3        impi/2017.7        impi/2018.3    mvapich2/2.3rc1 (D)
openmpi/2.1.3         stack/intel
```

Módulos disponibles

Descargar un módulo

Ejemplo

```
nct00018@login2:~> echo $LD_LIBRARY_PATH
/apps/INTEL/2017.4/mkl/lib/intel64:/apps/INTEL/2017.4/impi/2017.3.196/lib64:/apps/INTEL/
2017.4/lib/intel64

nct00018@login2:~> module unload impi/2017.4
remove impi/2017.4 (PATH, MANPATH, LD_LIBRARY_PATH)

nct00018@login2:~> echo $LD_LIBRARY_PATH
/apps/INTEL/2017.4/mkl/lib/intel64:/apps/INTEL/2017.4/lib/intel64

nct00018@login2:~> module load intel/2018.4

nct00018@login2:~> echo $LD_LIBRARY_PATH
/apps/INTEL/2018.4.057/lib/intel64:/apps/INTEL/2017.4/mkl/lib/intel64

nct00018@login2:~> module load impi/2018.4
load impi/2018.4 (PATH, MANPATH, LD_LIBRARY_PATH)

nct00018@login2:~> echo $LD_LIBRARY_PATH
/apps/INTEL/2018.4.057/impi/2018.4.274/lib64:/apps/INTEL/2018.4.057/lib/intel64:/apps/IN
TEL/2017.4/mkl/lib/intel64
```

Cargar un módulo

Directorios

- En los entornos HPC normalmente tenemos
  - Zonas de disco visibles desde toda la máquinas gestionadas por algún sistema de ficheros paralelo
    - GPFS (General Parallel File System) es el más típico
    - Se realizan copias de seguridad de algunas de estas zonas de disco
      - /gpfs/scratch accesible pero no se realizan backups
  - Zonas de disco privadas (locales) a los nodos de computación accesibles a los usuarios
    - Zonas accesibles a modo de zonas temporales
    - Solo válidas durante la ejecución del job
    - Accesibles a través de $TMPDIR
  - Zonas de disco que no son accesibles para el usuario
  - El espacio está limitado por usuario y grupos
    - MareNostrum: bsc_quota

Compilación

- Podemos elegir entornos open source como gcc, g++ o propietarios como icc
- Las aplicaciones tradicionales HPC usan básicamente dos modelos de programación para generar programas paralelos
  - OpenMP
    - El **programador añade directivas** a un programa secuencial para expresar las regiones que pueden ejecutarse en paralelo
    - El **compilador transforma las directivas en llamadas al runtime** específico de OpeMP
    - Sólo extrae paralelismo dentro de un nodo ya que se basa en threads y memoria compartida
  - MPI
    - El programador debe crear el paralelismo y hacer la transferencia de datos explícitamente entre procesos **mediante el API de MPI**
    - Permite crear paralelismo en múltiples nodos
  - Típicamente se combina MPI+OpenMP

Compilación

- OpenMP
  - OpenMP es un estándar de modelo mediante directivas. Existen diferentes librerías que ofrecen este estándar
  - Hay que activar el flag de OpenMP o no se procesan las directivas
  - gcc: -fopenmp
  - icc **-qopenmp** (ifort para programas fortran)
- MPI
  - MPI es un API, hay diferentes implementaciones que ofrecen este API.
  - Hay que utilizar un programa propio de cada versión que extiende el entorno de compilación+enlace con las librerías de cada versión
  - mpicc → Normalmente utiliza todo el entorno de gcc
  - mpiicc → Entorno Intel (mpiifort para programas fortran)

Ejecución

- OpenMP
  - No necesita de ningún comando adicional para ejecutar una aplicación con OpenMP ya que los threads se crean en el contexto de proceso UNIX.
  - Mediante variables de entorno se configuran opciones como el número de threads (por proceso)
- MPI
  - Necesita de un comando propio para ejecutar estos programas que se encarga de crear los procesos de forma remota en coordinación con el sistema de colas.
  - Estos comandos aceptan opciones como número total de procesos=tasks, lista de nodos (o simplemente número de nodos), procesos por nodo, interfaces de red que queremos utilizar, etc.
  - mpirun : Entorno gcc típicamente
  - mpiexec.hydra: Entorno Intel

Como pedir recursos para ejecutar nuestros programas en un entorno HPC

SISTEMAS DE COLAS

Los sistemas de colas

- Los sistemas de colas gestionan los recursos computacionales disponibles
- Para poder ejecutar, hemos de:
  1. Pedir recursos (dentro de nuestros límites)
  2. Ejecutar nuestros programas usando estos recursos
     - Podemos ejecutar N programas asociados a una única petición de recursos
- SLURM es un sistema de colas open source
- Dependiendo de la instalación, el administrador define particiones o QoS
  - Particiones son grupos de nodos que tienen asociados unos límites
    - Máximo de nodos
    - Tiempo máximo de ejecución
  - QoS (Quality of Service) es un concepto más nuevo. Define límites asociados al job.
    - Prioridad del job
    - Límites
    - Una partición asociada
- Los Jobs se ejecutan a medida que hay recursos disponibles. Para garantizar unos recursos el administrador puede crear reservas de recursos

Sistemas de colas:comandos

- Algunos comandos básicos de SLURM son:
  - https://slurm.schedmd.com/quickstart.html
  - sinfo: para conseguir información de las particiones (web slurm)

adev0: **sinfo**

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
debug*	up	30:00	2	down*	adev[1-2]
debug*	up	30:00	3	idle	adev[3-5]
batch	up	30:00	3	down*	adev[6,13,15]
batch	up	30:00	3	alloc	adev[7-8,14]
batch	up	30:00	4	idle	adev[9-12]

  - squeue: para conseguir información del estado de las colas, jobs encolados, tiempo de espera estimado, etc (web slurm)

adev0: **squeue**

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
65646	batch	chem	mike	R	24:19	2	adev[7-8]
65647	batch	bio	joan	R	0:09	1	adev14
65648	batch	math	phil	PD	0:00	6	(Resources)

Sistemas de colas:comandos

- scontrol: Comando para reporter información más detallada sobre nodos, particiones, configuración, etc. Comando para administradores principalmente (web slurm)

adev0: **scontrol show partition**
PartitionName=debug TotalNodes=5 TotalCPUs=40
RootOnly=NO Default=YES OverSubscribe=FORCE:4 PriorityTier=1
State=UP MaxTime=00:30:00 Hidden=NO MinNodes=1 MaxNodes=26
DisableRootJobs=NO AllowGroups=ALL Nodes=adev[1-5] NodeIndices=0-4

adev0: **scontrol show job**
JobId=65672 UserId=phil(5136) GroupId=phil(5136) Name=math
Priority=4294901603 Partition=batch BatchFlag=1 AllocNode:Sid=adev0:16726
TimeLimit=00:10:00 ExitCode=0:0 StartTime=06/02-15:27:11 EndTime=06/02-15:37:11
JobState=PENDING NodeList=(null) NodeListIndices= NumCPUs=24
ReqNodes=1 …
PreSusTime=0 Command=/home/phil/math WorkDir=/home/phil

Sistemas de colas:comandos

- En MN estos comandos no se pueden ejecutar directamente, hay unos comandos propios del BSC
  - bsc_queues: devuelve las “colas” disponibles para el grupo del usuario y sus límites
  - bsc_quota: devuelve es estado actual del consumo de los recursos asignados y los límites
  - bsc_acct: devuelve información de ”accounting” del proyecto
  - bsc_load: : devuelve la carga actual de los nodos de computación

Ejecución en SLURM

- Comandos para pedir recursos: sbatch, salloc, srun
- sbatch: recibe un script como parámetro con una lista de requerimientos. El script puede contener una ejecución secuencial pero típicamente incluye el comando srun para ejecutar programas paralelos.
- salloc: similar a sbatch pero solo implica reserva de recursos
  - no es obligatorio incluir un script para ejecutar, se puede indicar después lo que queremos hacer
  - Script= fichero con comandos a ejecutar, no es un binario.
- En todos los casos, se puede pedir recursos mediante argumentos de los comandos o mediante directivas dentro de los scripts.
- Típicamente: sbatch + srun dentro del script
- Cada reserva de recursos

Ejecución en SLURM

```
#!/bin/bash
#SBATCH --job-name=bt-mz
#SBATCH --output=bt-mz_%j.out
#SBATCH --error=bt-mz_%j.err
#SBTACH --nodes=1
#SBATCH --ntasks=12
#SBATCH --cpus-per-task=4
#SBATCH --qos=debug

#MPI +  OpenMP
export NAS_PATH=$HOME/sesion2/psd/sesion2/NPB3.3.1-MZ/NPB3.3-MZ-MPI/bin
export OMP_NUM_THREADS=4

# Ejecutara primero bt y luego sp, los valores de srun son por defecto
srun $NAS_PATH/bt-mz.C.12
srun $NAS_PATH/sp-mz.C.12
```

Ejecución en SLURM

- Cuando es OpenMP no necesitamos srun para ejecutar el programa ya que estamos limitados al propio nodo donde esta el proceso.

```
#!/bin/bash
#SBATCH --job-name=bt-mz
#SBATCH --output=bt-mz_%j.out
#SBATCH --error=bt-mz_%j.err
#SBATCH --nodes=1
#SBATCH --qos=debug

#Solo OpenMP
export NAS_PATH=$HOME/sesion2/psd/sesion2/NPB3.3.1-MZ/NPB3.3-MZ-OMP/bin
export OMP_NUM_THREADS=48
$NAS_PATH/bt-mz.C.x
```

Ejecución en SLURM

```
#!/bin/bash
#SBATCH --job-name=bt-mz
#SBATCH --output=bt-mz_%j.out
#SBATCH --error=bt-mz_%j.err
#SBTACH --nodes=1
#SBATCH --ntasks=12
#SBATCH --cpus-per-task=4
#SBATCH --qos=debug

#MPI +  OpenMP
export NAS_PATH=$HOME/sesion2/psd/sesion2/NPB3.3.1-MZ/NPB3.3-MZ-MPI/bin
export OMP_NUM_THREADS=4

# Ejecutara primero bt y luego sp, los valores de srun son por defecto
srun $NAS_PATH/bt-mz.C.12
srun $NAS_PATH/sp-mz.C.12
```

Ejecución en SLURM

```
#!/bin/bash
#SBATCH --job-name=bt-mz
#SBATCH --output=bt-mz_%j.out
#SBATCH --error=bt-mz_%j.err
#SBTACH --nodes=1
#SBATCH --qos=debug

#MPI +  OpenMP
export NAS_PATH=$HOME/sesion2/psd/sesion2/NPB3.3.1-MZ/NPB3.3-MZ-MPI/bin

export OMP_NUM_THREADS=4
srun–ntasks=12 –cpus-per-task=4 $NAS_PATH/bt-mz.C.12
export OMP_NUM_THREADS=2
srun–ntasks=12 –cpus-per-task=2 $NAS_PATH/sp-mz.C.12
export OMP_NUM_THREADS=1
srun–ntasks=48 –cpus-per-task=1 $NAS_PATH/bt-mz.C.48
```

Ejecución en SLURM

- Con sbatch pedimos recursos, con srun ejecutamos programas

```
#!/bin/bash
#SBATCH --job-name=bt-mz
#SBATCH --output=bt-mz_%j.out
#SBATCH --error=bt-mz_%j.err
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --qos=debug

#MPI +  OpenMP
export NAS_PATH=$HOME/sesion2/psd/sesion2/NPB3.3.1-MZ/NPB3.3-MZ-MPI/bin
export OMP_NUM_THREADS=4

# Ejecutara los dos a la vez, cada uno en un nodo
srun --nodes=1 $NAS_PATH/bt-mz.C.12 &
srun --nodes=1 $NAS_PATH/sp-mz.C.12
```

Ejecución en SLURM

- Existen muchas otras opciones (normalmente las opciones de sbatch, salloc y srun son las mismas)
  - --reservation=reservation_name
  - --time=minutes
  - --partition=partition_name
  - --nodelist=hosts
  - --nodefile=file
  - --exclude=hosts
  - --constraint="CONS1&CONS2"
    - Ejemplo: --constraint="highmem", --constraint="GPU"

Cancelar trabajos

- scancel es el comando para cancelar trabajos, podemos cancelar un trabajo concreto (por jobid) o podemos aplicar otros criterios como cancelar todos los trabajos de un usuario, por nombre, etc (de la web de slurm)
  - --user=user_name
  - --name=job_name

```
adev0: sbatch test
srun: jobid 473 submitted

adev0: squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
473          batch       test jill       R 00:00     1              adev9

adev0: scancel 473

adev0: squeue

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
```