# Projeto de um Cluster Didático para Programação Paralela e Distribuída (Parte II)

Paulo Shirley
Universidade Aberta,
pos@uab.pt

**Resumo**

O aumento do poder de cálculo permite resolver problemas mais rapidamente ou problemas de maiores dimensões antes inacessíveis. Este aumento tem um grande impacto a todos os níveis, sejam eles de desenvolvimento, investigação ou de prestação de serviços. A Computação de Alto Desempenho (HPC - High Performance Computing) implementada através de um conjunto de computadores pessoais dedicados ligados por uma rede privada, vulgarmente denominado Beowulf cluster, surge como uma solução viável e relativamente económica para o acesso a um maior poder computacional. Este trabalho consiste na 2ª parte da descrição da conceção, planeamento e implementação de um cluster experimental que replica o ambiente e operacionalidade de clusters de maiores dimensões, permitindo o contato com este tipo de tecnologia e a sua utilização para fins didáticos em programação paralela e distribuída, entre outros.

**Palavras-chave**: cluster, HPC, programação paralela, MPI, cálculo científico

**Title:** Design of a Didactic Cluster for Parallel and Distributed Programming (Part II)

**Abstract**

Increasing computing power allows one to solve problems faster or solve larger previously inaccessible problems. This increase has a major impact at all levels, be it development, research or service delivery. High Performance Computing (HPC) implemented through a set of dedicated personal computers connected by a private network, called a Beowulf cluster, emerges as a viable and relatively inexpensive solution for access to increased computing power. This work consists of the 2nd part of the description of the design, planning and implementation of an experimental cluster that replicates the environment and operation mode of larger clusters, allowing the contact with this type of technology and its use for didactic purposes in parallel and distributed programming, among others.

**Keywords**: cluster, HPC, parallel programming, MPI, scientific computing

## 1. Introdução

Na parte I do artigo [Shirley 2017] foi apresentada uma perspetiva geral da arquitetura de um cluster (agregado) de computadores pessoais ligados entre si por uma rede privada e da sua instalação e operação. Descreveram-se os componentes de hardware do cluster e parte dos componentes de software do servidor responsável pela instalação automatizada dos nós do cluster, designado por servidor de software. Os componentes descritos na parte I correspondem aos passos 1 a 6 da tabela 1, reproduzida aqui por conveniência.

**Tabela 1.** Passos para a instalação do Servidor de Software.

1- Instalação do sistema operativo
2- Estrutura de diretorias para o servidor FTP
3- Configuração do servidor FTP
4- Configuração do cliente NFS
5- Configuração do servidor PXE (DHCP + TFTP)
6- Configuração do servidor NTP
7- Compilação e configuração do gestor de recursos Slurm
8- Compilação e configuração da biblioteca MPI
9- Configuração do Firewall
10- Ficheiros de configuração dos nós

Nesta segunda parte do artigo, na secção 2 descrevem-se os componentes de software correspondentes aos passos 7 a 10 da tabela 1, na secção 3 são dados exemplos de utilização do cluster e na secção 4 são apresentadas as conclusões finais.

## 2. Componentes de Software (continuação)

Continuação da descrição dos passos para a instalação do servidor de software, iniciada na secção 4 da parte I do artigo.

### 2.1. Compilação e configuração do gestor de recursos Slurm

O slurm (Simple Linux Utility for Resource Management) [slurm] é um software de código aberto que proporciona um sistema de gestão de recursos de clusters e escalonamento (scheduling) de trabalhos (jobs), tolerante a falhas e altamente escalável para clusters Linux de grandes e pequenas dimensões, sendo usado em grande parte dos 500 maiores supercomputadores do mundo [top500]. Como gestor de recursos e de carga de trabalho do cluster, o slurm possui três funções principais. Primeiro, atribuir acesso exclusivo e / ou não exclusivo a recursos (nós computacionais) aos utilizadores por algum tempo para que eles possam executar trabalhos (normalmente programas paralelos). Segundo, fornece uma estrutura para iniciar, executar e monitorizar trabalhos no conjunto de nós computacionais alocados. Terceiro, arbitrar a contenção de recursos gerindo filas de espera de trabalhos pendentes. Funcionalidades e opções adicionais podem ser implementadas através de um esquema baseado em Plugins.

A instalação deste software consiste na parte mais complexa e trabalhosa de todo o software necessário à operação do cluster. Felizmente, para uma funcionalidade relativamente básica de operação as instruções dadas aqui deverão ser suficientes. No entanto, havendo muitas opções e funcionalidades adicionais possíveis é recomendável a leitura adicional dos guias rápidos de administração, instalação e utilização [slurm AG][slurm UG].

O software é fornecido em código fonte num ficheiro vulgarmente denominado "tarball", que consiste num ficheiro de arquivo .tar comprimido, por exemplo slurm-17.02.11.tar.bz2, que pode ser descarregado a partir do sítio [slurm]. Para obter os pacotes finais .rpm para instalação do slurm são necessários três passos. O primeiro consiste em pré-instalar outros pacotes de software de que o slurm necessita (dependências) executando como utilizador root os comandos da figura 1a. Poderá acontecer algum dos pacotes indicados já estar instalado, caso em que o respetivo comando será simplesmente ignorado. Neste passo é conveniente consultar o guia rápido de administração e instalação [slurm AG] para verificar se existe alguma funcionalidade adicional considerada fundamental e instalar também as respetivas dependências. Dado que o guia recomenda não considerar para a primeira instalação do slurm a contabilização (accounting) de dados de utilização do cluster, nenhuma base de dados é necessária / instalada.

O segundo passo consiste na criação das estrutura de diretorias necessárias à compilação e criação de pacotes de software tipo .rpm. A criação das diretorias é feita por e na diretoria home de um utilizador regular (não root) conforme comando da figura 1b. Este utilizador será o mesmo que irá executar o comando de compilação.

O terceiro passo consiste na compilação propriamente dita do software slurm contido no tarball executando como utilizador regular (não root) na diretoria onde está o tarball o comando da figura 1c. No final do processo, os pacotes .rpm criados estão na diretoria ~/rpmbuild/RPMS/i686 e mais tarde serão copiados para a diretoria de um repositório local.

```
yum -y groupinstall perl-runtime
yum -y install perl-ExtUtils-MakeMaker openssl-devel pam-devel
yum -y install gcc gcc-c++ gcc-gfortran make
yum -y kernel-devel kernel-headers rpm-build redhat-rpm-config
yum -y install readline readline-devel ncurses ncurses-devel
yum -y rrdtool freeipmi redhat-lsb
yum -y install munge munge-devel munge-libs
```

**(a)**

```
mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
```

**(b)**

```
rpmbuild –tb slurm-17.02.11.tar.bz2
```

**(c)**

**Figura 1. (a)** Comandos para instalar as dependências do slurm; **(b)** Comando para criar estrutura de diretorias de suporte à criação de pacotes .rpm. **(c)** Comandos para compilar e criar os pacotes .rpm para instalação do slurm.

Nota: a partir da versão slurm 17.11 inclusive o ficheiro slurm.spec que se encontra no tarball e que contém as instruções para a criação dos pacotes rpm foi revista e passou a assumir por defeito que a compilação é efetuada no CentOS 7. Para continuar a possibilitar a compilação e manter a compatibilidade com sistemas anteriores é fornecido na diretoria contrib do tarball o ficheiro alternativo slurm.spec-legacy que segue a linha dos anteriores. No entanto, para o usar é necessário descompactar o tarball, substituir o ficheiro slurm.spec pela versão legacy e voltar a compactar o tarball. Essas versões não foram testadas no âmbito deste trabalho.

### 2.2. Compilação e configuração da biblioteca MPI

Um programa paralelo pode ser visto como um conjunto de programas, ou processos do ponto de vista do sistema operativo, a serem executados em várias unidades de processamento (atualmente usualmente denominadas núcleos ou cores) pertencentes a computadores (nós) diferentes. O objetivo deste conjunto de programas é colaborar e cooperar entre si para resolver um determinado problema para o que regra geral necessitam de comunicar uns com os outros. Dois programas que estejam a ser executados em computadores diferentes podem comunicar entre si por troca de mensagens, funcionalidade essa que é atualmente fortemente suportada pela existência de uma especificação de uma biblioteca padrão denominada MPI (Message Passing Interface) definida pelo MPI Fórum [MPI Fórum]. Um tutorial de introdução sobre programação com a biblioteca MPI é dado em [Barney 2018].

Existem várias implementações da especificação da biblioteca MPI, tendo sido escolhida neste trabalho a biblioteca OpenMPI [openmpi]. Esta biblioteca existe como um conjunto de pacotes de software no repositório base do Linux CentOS, no entanto a sua compilação não foi configurada a pensar na sua utilização num cluster em conjunto com um gestor de recursos como o slurm, pelo que é necessário recompilar os pacotes .rpm correspondentes ao OpenMPI.

A componente em falta no pacote openmpi é o suporte da interface pmi (Process Management Interface) [Balaji P. et al][PMI-2 API] que permite que quando um programa MPI é executado a biblioteca MPI e o gestor de recursos comuniquem entre si estabelecendo o ambiente necessário à execução do programa paralelo, nomeadamente estabelecendo qual o número de processos e quais as unidades de processamento em que vão ser executados. Para a recompilação e criação dos pacotes .rpm do openmpi o processo é semelhante ao da criação dos pacotes do slurm, sendo efetuado em quatro passos.

Dado que o pacote openmpi é um pacote que já existe no repositório base oficial do CentOS 6, o primeiro passo consiste em obter o pacote com o código fonte, os chamados pacotes .src.rpm, que podem ser descarregados a partir do sítio [CentOS Vault]. No repositório estão disponíveis várias versões, sendo usada a mais recente para o CentOS 6 que é a 1.10, a que corresponde o ficheiro openmpi-1.10.2-2.el6.src.rpm.

O segundo passo consiste em descompactar o pacote .src.rpm, o que resulta no povoamento com o seu conteúdo das subdiretorias de ~/rpmbuild. O terceiro passo consiste em alterar o ficheiro openmpi.spec que contem a informação de configuração para a compilação. O quarto passo consiste em executar o comando rpmbuild para a criação dos pacotes .rpm. A figura 2 mostra os comandos necessários e a linha a incluir e respetiva localização no ficheiro openmpi.spec.

Tal como no caso do software slurm, os pacotes .rpm criados encontram-se localizados na diretoria ~/rpmbuild/RPMS/i686.

```
# descompactar código fonte
rpm –i openmpi-1.10.2-2.el6.src.rpm
# alterar openmpi.spec e acrescentar a linha "--with-pmi"
#   junto com as outras do mesmo tipo
vim ~/rpmbuild/SPECS/openmpi.spec
# compilar e criar pacotes .rpm
rpmbuild –bb ~/rpmbuild/SPECS/openmpi.spec
```

**Figura 2.** Comandos para (re)compilar e criar os pacotes .rpm do openmpi com suporte para pmi.

### 2.3. Configuração do Firewall

A configuração do firewall dos nós é diferente para acesso à rede privada do cluster e para a rede externa. Para acesso à rede privada não existem restrições pelo que o respetivo porto de rede de todos os nós é configurado com acesso livre (trusted). Para acesso à rede externa pelos portos de rede do servidor de software (eth0) e do nó de interface com o utilizador (eth0), estes são configurados por uma questão de segurança para serem acedidos a partir do exterior apenas pelos protocolos SSH e opcionalmente pelo protocolo ICMP. Por outro lado, o acesso ao exterior é livre, sendo feito exclusivamente apenas a partir do servidor de software e do nó de interface com o utilizador, sendo a fundamentação desta política o possibilitar aos utilizadores do cluster o acesso a servidores externos para obterem dados para processamento no cluster. No entanto, a configuração do firewall é um assunto sensível dadas as suas implicações a nível da segurança, pelo que devem ser levadas em conta também as políticas de segurança do local onde vai ser instalado o cluster.

A configuração do firewall dos nós é feita no ficheiro kickstart de instalação automatizada dos nós. No servidor de software a configuração de ambos os portos de rede deve ser feita ou revista manualmente.

### 2.4. Ficheiros de configuração dos nós

Os ficheiros de configuração dos nós são os necessários à instalação automatizada dos nós computacionais e do nó de interface com o utilizador. São colocados na diretoria /var/ftp/pub/centos6-32/cfg associada ao serviço ftp. A tabela 2 mostra a lista dos ficheiros.

**Tabela 2.** Ficheiros de configuração dos nós.

1- hosts
2- ntp.conf
3- local-repos.repo
4- openmpi.sh
5- munge.key
6- slurm.conf
7- slurm-install.sh
8- ksnode.cfg
9- ksnode-ui.cfg

O ficheiro hosts é dado na parte I em [Shirley 2017, figura 2].

O ficheiro ntp.conf é igual ao do servidor de software com as diferenças nas linhas mostradas na figura 3. As linhas iniciadas por "restrict" e por "server" são comentadas (anuladas) e é acrescentada uma nova linha "server" indicando que o software server é o servidor ntp na rede privada dos nós do cluster.

```
# Hosts on local network are less restricted.
#restrict 172.17.0.0 mask 255.255.254.0 nomodify nopeer
  noquery

# Use public servers from the pool.ntp.org project.
# Please consider joining the pool
(http://www.pool.ntp.org/join.html).
#server 0.centos.pool.ntp.org iburst
#server 1.centos.pool.ntp.org iburst
#server 2.centos.pool.ntp.org iburst
#server 3.centos.pool.ntp.org iburst
server 172.17.0.254 iburst
```

**Figura 3.** Troço com as diferenças do ficheiro ntp.conf do servidor de software para os nós do cluster (clientes).

O ficheiro local-repos.repo define os repositórios de pacotes de software .rpm a serem utilizados por cada nó para a sua instalação, atualizações posteriores e ainda software novo que se pretenda instalar no cluster. O ficheiro indica os repositórios criados no servidor de software e disponibilizado aos nós do cluster pelo protocolo FTP. A figura 4 mostra o conteúdo do ficheiro.

```
# /etc/yum.repos.d/local-repos.repo  p/ rede do cluster
#
# Substitui os repos oficiais na Internet por repos locais
#   no Servidor de Software
# Remover primeiro todos os ficheiros /etc/yum.repos.d/*.repo

# Local Base
[localbase]
name=Local Base
baseurl=ftp://172.17.0.254/centos6-32/repos/base
gpgcheck=0
enabled=1

# Local Extras
[localextras]
name=Local Extras
baseurl=ftp://172.17.0.254/centos6-32/repos/extras
gpgcheck=0
enabled=1

# Local Others
[localothers]
name=Local Others
baseurl=ftp://172.17.0.254/centos6-32/repos/others
gpgcheck=0
enabled=1

# Local Updates
[localupdates]
name=Local Updates
baseurl=ftp://172.17.0.254/centos6-32/repos/updates
gpgcheck=0
enabled=1
```

**Figura 4.** Configuração dos repositórios de software locais para os nós do cluster.

O ficheiro openmpi.sh tem como função configurar as variáveis de ambiente globais PATH e LD_LIBRARY_PATH necessárias à operação da biblioteca MPI. A figura 5 mostra o conteúdo do ficheiro.

```
# /etc/profile.d/openmpi.sh  p/ rede do cluster
#
# vars de ambiente PATH e LD_LIBRARY_PATH para openmpi-1.10

if ! echo ${PATH} | /bin/grep -q /usr/lib/openmpi-1.10/bin ;
then
   PATH=/usr/lib/openmpi-1.10/bin:${PATH}
```

```
fi
if ! echo ${LD_LIBRARY_PATH} | /bin/grep -q /usr/lib/openmpi-
  1.10/lib ;
then
   LD_LIBRARY_PATH=/usr/lib/openmpi-1.10/lib:${LD_LIBRARY_PATH}
fi
export PATH LD_LIBRARY_PATH
```

**Figura 5.** Configuração das variáveis de ambiente globais para a biblioteca MPI.

O gestor de recursos slurm recomenda e é utilizado neste trabalho o software munge para implementar mecanismos de autenticação de comunicações entre componentes do slurm. O ficheiro munge.key é a chave utilizada nos processos de autenticação e é gerada uma vez e depois utilizada em todos os nós, sendo gerada preferencialmente pelo comando mostrado na figura 6. No comando mostrado é utilizado o dispositivo /dev/random na geração da chave, no entanto este comando pode bloquear se o sistema operativo decidir que não existe entropia suficiente no sistema para a geração da chave. Nessa situação, o comando pode ser executado noutro computador "mais ativo" ou alternativamente utilizado (mas menos seguro) o dispositivo /dev/urandom.

```
dd if=/dev/random bs=1 count=1024 > munge.key
```

**Figura 6.** Geração da chave munge.key utilizada em autenticação.

O ficheiro slurm.conf é o ficheiro principal de configuração da operação de todo o cluster. Nele são definidos os nós computacionais que constituem o cluster e o nó que os controla e faz a sua gestão (do ponto de vista do slurm), a política seguida pelas filas de espera de trabalhos submetidos pelos utilizadores e muitas mais opções de configuração descritas na (vasta) documentação do software [Slurm docs]. Para ajudar a criar uma file de configuração o software disponibiliza um configurador "configurator.html" que funciona como um questionário onde se selecionam as opções de configuração e no final é criado um ficheiro slurm.conf, que ainda necessitará de alguns ajustes mas que constitui um bom ponto de partida para a versão final.

A figura 7 mostra um exemplo muito simplificado mas funcional de um ficheiro de configuração do slurm, onde foram retiradas todas as linhas em comentário, linhas essas que tinham como significado assumir as respetivas opções por defeito. A man page de slurm.conf contém uma descrição completa de todas as opções e parâmetros de configuração. Das linhas mostradas destacam-se as seguintes:

- "`ControlMachine=juno`". O slurm opera essencialmente com dois daemons, slurmctld que corre num único nó e controla o cluster, e slurmd que corre em todos os nós que executem trabalho e/ou aceitem comandos do utilizador. Esta opção indica o nome do nó que executa o daemon slurmctld.

- "`MpiDefault=pmi2`". Esta opção indica que independentemente da implementação da biblioteca MPI utilizada no cluster, a comunicação entre o slurm e a biblioteca MPI é feita através da interface pmi (versão 2).

- "`MpiParams=ports=12000-12999`". Esta opção destina-se à biblioteca openmpi, indicando os portos a utilizar para comunicação entre os seus componentes de software.

- "`SchedulerType=sched/backfill`". Esta opção define o tipo de escalonamento de trabalhos por defeito FIFO mas com a possibilidade de trabalhos de menor ou igual prioridade (por exemplo de reduzido tempo de execução e recursos) passarem à frente na fila de espera desde que a sua execução não atrase o início de trabalhos à sua frente na fila.

- "`SelectType=select/cons_res`". Esta opção identifica o tipo de algoritmo de seleção de recursos que é utilizado. Neste caso indica que os recursos contidos num nó, tais como núcleos do processador ou memória RAM, podem ser escolhidos como os recursos solicitados nos pedidos de submissão de trabalhos.

- "`SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK`". A opção "`CR_Core`" indica que o recurso preferencial que pode ser solicitado na submissão de um trabalho é o núcleo físico de um processador (core). Com esta configuração, o cluster pode ser visto como uma lista de núcleos onde podem ser executados processos, independentemente se pertencem ao mesmo processador ou nó. A opção "`CR_CORE_DEFAULT_DIST_BLOCK`" indica que os processos são atribuídos aos núcleos por blocos, ou seja, só é atribuído um núcleo de um novo processador se os processadores que estão a ser usados já têm os seus núcleos todos atribuídos, pelo que se tenta minimizar o número de processadores usados para cada trabalho (salvo instruções específicas em contrário, como por exemplo um processo por nó no caso de processos multitarefa).

- "`# COMPUTE NODES`". Nesta secção do ficheiro indicam-se os nomes dos computadores que constituem os nós computacionais do cluster e as suas características técnicas ou recursos. Na descrição de listas de nomes de computadores podem ser usadas gamas (ranges) para simplificar descrições de grandes quantidades de computadores como "nodes[1-4]" ou especificar nomes individuais separados por vírgulas como "node[13,15,17,20]". Os nós computacionais podem por conveniência serem agrupados em partições. No exemplo dado criaram-se três partições, a partição "single" com os computadores (nós) com processadores com um único núcleo, a partição "dual" com os computadores com processadores com dois núcleos (dual core) e por fim uma partição "total" com todos os computadores do cluster que é considerada a escolha por defeito. Ao submeter um trabalho, um utilizador pode escolher uma partição específica para a execução do trabalho ou no caso da sua omissão, o trabalho é executado na partição por defeito. Cada partição pode especificar um tempo máximo para a execução de um trabalho, neste exemplo 24 horas, ao fim do qual os processos do trabalho são terminados e iniciada a execução do próximo trabalho na fila de espera.

```
# /etc/slurm/slurm.conf  p/ rede do cluster
#
# full slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
```

```
#
ControlMachine=juno
AuthType=auth/munge
CacheGroups=0
CryptoType=crypto/munge
DisableRootJobs=NO
MpiDefault=pmi2
MpiParams=ports=12000-12999
PluginDir=/usr/lib/slurm
ProctrackType=proctrack/pgid
ReturnToService=1
SlurmctldPidFile=/var/run/slurm/slurmctld.pid
SlurmctldPort=6817
SlurmdPidFile=/var/run/slurm/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/var/spool/slurm/slurmd.spool
SlurmUser=slurm
StateSaveLocation=/var/spool/slurm/slurm.state
SwitchType=switch/none
TaskPlugin=task/none
#
# TIMERS
InactiveLimit=0
KillWait=30
MinJobAge=300
SlurmctldTimeout=120
SlurmdTimeout=300
Waittime=0
#
# SCHEDULING
FastSchedule=1
SchedulerType=sched/backfill
SchedulerPort=7321
SelectType=select/cons_res
# SelectTypeParameters with several options coma separated!
SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK
#
# LOGGING AND ACCOUNTING
AccountingStorageType=accounting_storage/none
AccountingStoreJobComment=YES
ClusterName=juno
JobCompType=jobcomp/none
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/none
SlurmctldDebug=3
SlurmctldLogFile=/var/log/slurm/slurmctld.log
SlurmdDebug=3
SlurmdLogFile=/var/log/slurm/slurmd.log
#
# COMPUTE NODES
NodeName=node[4-11]        CPUs=1 Sockets=1 CoresPerSocket=1
```

```
  ThreadsPerCore=1 State=UNKNOWN
NodeName=node[13,15,17,20] CPUs=2 Sockets=1 CoresPerSocket=2
  ThreadsPerCore=1 State=UNKNOWN
PartitionName=single Nodes=node[4-11]  MaxTime=24:00:00 State=UP
PartitionName=dual   Nodes=node[13,15,17,20] MaxTime=24:00:00
  State=UP
PartitionName=total  Nodes=node[4-11,13,15,17,20] Default=YES
  MaxTime=24:00:00 State=UP
```

**Figura 7.** Configuração do gestor de recursos slurm.

O ficheiro slurm-install.sh agrupa todos os comandos necessários à instalação do slurm num nó. A sua instalação não consiste só na instalação dos respetivos pacotes .rpm, mas também na criação no sistema do utilizador de nome slurm e na criação de diretorias e ficheiros específicos com permissões de leitura e escrita pelo utilizador slurm. Como se pode ver pelo exemplo do ficheiro slurm.conf dado na figura 7, são referidas diretorias e ficheiros que necessitam de serem criados explicitamente. Dada a extensão dos comandos necessários, estes foram agrupados num ficheiro que se mostra na figura 8.

```
# script slurm-install.sh  p/ instalação do slurm num nó

# diretoria e ficheiro de configuração
[ ! -d /etc/slurm ] && mkdir /etc/slurm
cp /root/cfg/slurm.conf  /etc/slurm/

# utilizador slurm (no home, no login)
useradd -M -s /sbin/nologin slurm

# ficheiros log em /var/log/slurm/
SLURMDIR=/var/log/slurm
mkdir $SLURMDIR; chmod 775 $SLURMDIR; chown slurm $SLURMDIR
SLURMFILE=$SLURMDIR/slurmctld.log
touch $SLURMFILE; chmod 664 $SLURMFILE; chown slurm $SLURMFILE
SLURMFILE=$SLURMDIR/slurmd.log
touch $SLURMFILE; chmod 664 $SLURMFILE; chown slurm $SLURMFILE

# ficheiros pid em /var/run/slurm/
SLURMDIR=/var/run/slurm
mkdir $SLURMDIR; chmod 775 $SLURMDIR; chown slurm $SLURMDIR
SLURMFILE=$SLURMDIR/slurmctld.pid
touch $SLURMFILE; chmod 664 $SLURMFILE; chown slurm $SLURMFILE
SLURMFILE=$SLURMDIR/slurmd.pid
touch $SLURMFILE; chmod 664 $SLURMFILE;

# state dirs em /var/spool/slurm/
SLURMDIR=/var/spool/slurm
mkdir $SLURMDIR; chmod 775 $SLURMDIR; chown slurm $SLURMDIR
SLURMDIR=/var/spool/slurm/slurm.state
mkdir $SLURMDIR; chmod 775 $SLURMDIR; chown slurm $SLURMDIR
SLURMDIR=/var/spool/slurm/slurmd.spool
```

```
mkdir $SLURMDIR; chmod 775 $SLURMDIR; chown slurm $SLURMDIR

# instalar pacotes .rpm
yum -y install slurm-plugins
yum -y install slurm slurm-devel slurm-munge
yum -y install slurm-perlapi slurm-pam_slurm slurm-contribs
```

**Figura 8.** Comandos para a instalação do slurm num nó do cluster.

O ficheiro ksnode.cfg é um ficheiro kickstart [redhat 2014] que automatiza a instalação de um nó providenciando as respostas às perguntas que seriam feitas durante uma instalação manual e além disso pode executar comandos adicionais para detalhes mais particulares da instalação em causa. O ficheiro ksnode.cfg destina-se à instalação de um nó computacional. Para o nó de interface com o utilizador é utilizado outro ficheiro específico. A figura 9 mostra o conteúdo do ficheiro ksnode.cfg com os comandos e instruções de instalação.

```
# Ficheiro kickstart ksnode.cfg para instalação de nó
#  computacional pela rede com pxe em eth0

cmdline
keyboard  pt-latin1
lang      en_US.UTF-8
timezone  Europe/Lisbon
install   # network
url --url=ftp://172.17.0.254/centos6-32/repos/base
logging   --level=info
firstboot --disable
poweroff  # after install, allow boot reconfig
auth      --useshadow  --passalgo=sha512
rootpw    --iscrypted  $1$/vNMYAPQ$xy1aBDNAAiE3wBUjtokqYa5
# network, hostname is place holder
firewall --disabled   # only for nodes
network  --bootproto=dhcp --device=eth0 --onboot=on --noipv6
  --hostname=imahost

# HD
zerombr
clearpart --all
bootloader --location=mbr  --append="vga=773"  # vga 1024x768
  256colors
part /boot --fstype=ext2 --asprimary --size=1000
part swap --asprimary --size=3000
part / --fstype=ext4 --asprimary --size=20000

%packages --nobase
# bare minimum
@core
man
man-pages
openssh-server
```

```
ftp
yum
authconfig
system-config-firewall-base

%post --log=/root/post.log
# get all cfg files by ftp to /root/cfg
mkdir /root/cfg;  cd /root/cfg
ftp -in 172.17.0.254 <<End-Of-Session
user ftp ''
type binary
cd /centos6-32/cfg
mget *
bye
End-Of-Session

# set /etc/hosts
cp /root/cfg/hosts  /etc/

# set local repos
rm -f /etc/yum.repos.d/*   # remove all default repos
# set plugins=0 and gpgcheck=0 in /etc/yum.conf
sed -i 's/gpgcheck=1/gpgcheck=0/' /etc/yum.conf
sed -i 's/plugins=1/plugins=0/'   /etc/yum.conf
cp /root/cfg/local-repos.repo  /etc/yum.repos.d/
yum makecache

# set openmpi environment variables
cp /root/cfg/openmpi.sh  /etc/profile.d/
# fix man pages path
sed -i 's@MANPATH\t/usr/share/man.*@&\nMANPATH
/usr/share/man/openmpi-1.10-i386@' /etc/man.config

# add users
useradd -u 1000 -c "Paulo Shirley" pos   # set password later

# update system
yum -y update

# install relevant packages
yum -y groupinstall perl-runtime
yum -y install perl-ExtUtils-MakeMaker yum-utils openssl-devel
  nfs-utils nfs-utils-lib
yum -y install gcc gcc-c++ gcc-gfortran make vim kernel-devel
  kernel-headers rpm-build pam pam-devel
yum -y install readline readline-devel ncurses ncurses-devel
  rrdtool freeipmi redhat-lsb redhat-rpm-config
yum -y install openmpi-1.10 openmpi-1.10-devel
yum -y install munge munge-devel munge-libs
yum -y install ntp ntpdate
```

```
# nfs client
mkdir /data
echo "172.17.0.253:/data  /data  nfs  rw,hard,intr  0 0" >>
  /etc/fstab
chkconfig rpcbind on
chkconfig netfs on

# set ntp client and net clock sync
cp /root/cfg/ntp.conf  /etc/
echo 'juno2' >>   /etc/ntp/step-tickers
chkconfig ntpd on

# munge
[ ! -d /etc/munge ] && mkdir /etc/munge
cp /root/cfg/munge.key  /etc/munge/
chown munge: /etc/munge/munge.key
chmod 400 /etc/munge/munge.key
chkconfig munge on

# slurm
. /root/cfg/slurm-install.sh
chkconfig slurm on

# void default hostname
sed -i 's/imahost//' /etc/sysconfig/network

%end
```

**Figura 9.** Ficheiro kickstart ksnode.cfg para instalação por rede de um nó computacional.

O ficheiro ksnode-ui.cfg é um ficheiro kickstart que automatiza a instalação do nó interface com o utilizador. É muito semelhante ao ficheiro ksnode.cfg da figura 9 pelo que apenas se vai salientar as diferenças entre os dois ficheiros. As diferenças são a nível do firewall e porto de rede eth0 para a rede externa, a nível do disco rígido existe mais uma partição que ocupa o espaço restante do disco e é atribuída à diretoria /data, o nó não é um cliente nfs mas sim um servidor nfs que exporta a diretoria /data para todos os nós do cluster. A diretoria /data funciona como uma segunda diretoria /home, tendo cada utilizador também uma subdiretoria em /data com o seu nome. É nesta diretoria que os utilizadores do cluster devem colocar os seus programas e dados para processamento pelo cluster, dado que a diretoria é acessível por todos os nós do cluster. A figura 10 mostra as linhas relevantes para obter o ficheiro ksnode-ui.cfg, linhas a substituir ou a acrescentar a ksnode.cfg, tendo em conta as diferenças mencionadas

```
# firewall, apenas ssh para a rede externa
firewall --enable --trust=eth1 --ssh

# rede privada
network --device=eth1 --bootproto=dhcp --onboot=on --noipv6
  --hostname=juno
```

```
# rede externa (substituir por valores apropriados)
network --device=eth0 --bootproto=static --onboot=on --noipv6
  --ip=192.168.5.23 --netmask=255.255.255.0
  --gateway=192.168.5.254 --nameserver 192.168.5.10

# HD
part /data --fstype=ext4 --asprimary --size=500 --grow

# add users
mkdir /data/pos
chown pos:pos /data/pos

# nfs server (/data)
echo "/data  172.17.0.0/23(rw,root_squash,sync,subtree_check)"
  > /etc/exports
# turn on nfs
chkconfig rpcbind on
chkconfig nfs on
```

**Figura 10.** Principais diferenças entre os ficheiros ksnode.cfg e ksnode-ui.cfg.

Para finalizar a instalação do servidor de software, falta povoar o repositório local others na diretoria /var/ftp/pub/centos6-32/repos/others. Este repositório vai conter os pacotes do software munge, obtidos por descarregamento da Internet do repositório epel do CentOS e os pacotes do software slurm obtidos anteriormente por compilação. Por último, é necessário substituir no repositório local base os pacotes do software openmpi existentes pelos obtidos também anteriormente por compilação. A figura 11 mostra os comandos necessários a esta finalização.

```
# criar repositório others
DIRFTP=/var/ftp/pub/centos6-32
mkdir $DIRFTP/repos/others/Packages
# munge software
yumdownloader --destdir $DIRFTP/repos/others/Packages
  munge munge-devel munge-libs
# slurm software
cp ~/rpmbuild/RPMS/i686/slurm-17.02*.rpm
  $DIRFTP/repos/others/Packages
createrepo -q $DIRFTP/repos/others

# atualizar repositório base com openmpi
cp -f ~/rpmbuild/RPMS/i686/openmpi-1.10*.rpm
  $DIRFTP/repos/base/Packages
createrepo -q --update $DIRFTP/repos/base
```

**Figura 11.** Comandos para criar o repositório others e atualizar o base.

É ainda de referir que o ficheiro AC1101FD mencionado na secção 4.5.1 na parte I do artigo [Shirley 2017] para instalação do nó de interface com o utilizador e o ficheiro

AC1101 para instalação de um nó computacional mostrado na mesma secção na figura 9a, diferem na linha que se mostra na figura 12, devido ao ficheiro kickstart e o porto de rede serem diferentes para a instalação do nó de interface com o utilizador.

```
append initrd=initrd.img ks=ftp://172.17.0.254/centos6-32
  /cfg/ksnode-ui.cfg ksdevice=eth1
```

**Figura 12.** Alteração à linha no ficheiro AC1101 para se obter o ficheiro AC1101FD.

Nesta altura a instalação do servidor de software está concluída. Para ficar operacional deve ser feita a reinicialização do sistema operativo (reboot). Após a reinicialização os nós podem começar a ser instalados, devendo o primeiro nó a ser instalado ser o nó de interface com o utilizador. Para instalar nós o procedimento é editar o ficheiro /etc/dhcp/dhcpd.conf e atribuir a cada nó número i a instalar o IP 172.17.1.i, após o que se deve reinicializar o serviço DHCP com o comando "service dhcp restart" e ligar os respetivos nós. Quando os nós se desligarem quer dizer que terminaram a sua instalação. Deve-se então voltar a editar o ficheiro dhcpd.conf e alterar os IP para 172.17.0.i, reinicializar o serviço DHCP e ligar novamente os nós. Se tudo correr bem, após a instalação de todos os nós o cluster deve agora estar operacional. Na seção seguinte são dados exemplos de utilização do cluster que também podem ser utilizados para testar o seu bom funcionamento.

## 3. Exemplos de utilização do cluster

Nesta seção são dados exemplos de utilização do cluster. A configuração do cluster utilizada é a dada pelo ficheiro de configuração slurm.conf da figura 7. O cluster é constituído por: 8 computadores com 1 núcleo cada (single core), nomes node4 a node11 consecutivos (abreviado por node[4-11]) e agrupados numa partição de nome "single"; 4 computadores com 2 núcleos cada (dual core) e nomes node13, node15, node17 e node20 (abreviado por node[13,15,17,20]) agrupados numa partição de nome "dual"; uma partição de nome "total", que é a partição por defeito para submissão de trabalhos e que agrupa todos os computadores do cluster. Neste cenário idealizado, os computadores com 1 núcleo estão fora de serviço pelo que o cluster conta com um total de 8 cores disponíveis para executar trabalho.

Existem sete comandos básicos para operar e submeter trabalhos para execução no cluster. A tabela 3 mostra uma perspetiva geral dos comandos com o nome do comando e uma breve descrição da sua funcionalidade. Os comandos cujos nomes começam pela letra 's' são comandos do gestor de recursos slurm. Para uma descrição detalhada de cada comando podem ser consultadas as respetivas man pages.

**Tabela 3.** Lista de comandos básicos para operar com o cluster.

| Comando | Descrição |
|---|---|
| mpicc | Compilar programas MPI. |
| sinfo | Informação sobre nós e partições do cluster. |
| srun | Executar programas paralelos MPI com n processos ou executar em paralelo n cópias de um programa sequencial. |
| sbatch | Submeter um trabalho em lote (batch job) definido num ficheiro (script) para uma fila de espera. |
| scancel | Cancelar um trabalho numa fila de espera. |
| squeue | Listar os trabalhos que se encontram numa fila de espera. |
| scontrol | Comando de administração. Além de atos administrativos também pode ser usado para obter informações sobre o cluster tais como características dos nós e estado de operação. |

Nos exemplos que se seguem considera-se que o utilizador de nome pos fez login remoto no nó de interface com o utilizador (juno) a partir de outra máquina Linux com o comando "ssh pos@juno". A seguir mudou a sua diretoria corrente para uma diretoria de trabalho pertencente à hierarquia /data, que é a que tem visibilidade em todos os nós do cluster.

### 3.1. Informações sobre o cluster

A figura 13 mostra alguns exemplos com o comando sinfo para obter informação sobre o cluster. A opção –N permite obter informação com uma linha por nó e a opção –p especificar uma partição. O estado down significa que o nó não está ativo e o estado idle significa que o nó está ativo mas sem trabalho atribuído. A opção –e dá uma informação mais compacta agrupando nós com características iguais. A opção –s é ainda mais compacta dando informações por partição e o estado dos respetivos nós. As letras (A/I/O/T) significam o estado dos nós (Allocated/Idle/Other/Total). Allocated significa que o nó está ativo e a executar trabalho. Idle o nó está ativo mas não está a executar trabalho, Others o nó está num estado que não é nem Allocated nem Idle. Total indica o número de nós (não núcleos ou cores) da partição. Para o exemplo dado, o estado other significa down. Pode ainda verificar-se que o tempo limite de execução de um trabalho é um dia, ou seja, 24 horas. O asterisco na partição total significa que é a partição por defeito.

```
[pos@juno ~]$ cd /data/pos/rcc
[pos@juno rcc]$ pwd
/data/pos/rcc
[pos@juno rcc]$ sinfo -N -p total
NODELIST   NODES PARTITION STATE
node4          1    total* down*
node5          1    total* down*
node6          1    total* down*
node7          1    total* down*
node8          1    total* down*
node9          1    total* down*
```

```
node10         1    total* down*
node11         1    total* down*
node13         1    total* idle
node15         1    total* idle
node17         1    total* idle
node20         1    total* idle
[pos@juno rcc]$ sinfo -e
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
single       up 1-00:00:00     8  down* node[4-11]
dual         up 1-00:00:00     4   idle node[13,15,17,20]
total*       up 1-00:00:00     8  down* node[4-11]
total*       up 1-00:00:00     4   idle node[13,15,17,20]
[pos@juno rcc]$ sinfo -s
PARTITION AVAIL  TIMELIMIT NODES(A/I/O/T) NODELIST
single       up 1-00:00:00        0/0/8/8 node[4-11]
dual         up 1-00:00:00        0/4/0/4 node[13,15,17,20]
total*       up 1-00:00:00       0/4/8/12 node[4-11,13,15,17,20]
[pos@juno rcc]$
```

**Figura 13.** Exemplos do comando sinfo com várias opções.

### 3.2. Submissão de trabalhos

A figura 14 mostra exemplos de submissão de trabalhos para processamento em lote (batch jobs) através de ficheiros (scripts) que incluem comandos para executar e opções de execução. A submissão de um trabalho definido num script é feita com o comando sbatch, sendo-lhe atribuído de imediato um número de identificação (batch job ID). Todos os dados de saída gerados pela execução desse trabalho são coligidos num único ficheiro de nome "slurm-ID.out" disponibilizado na diretoria corrente após o término da execução do trabalho.

A especificação de opções de execução pode ser feita em linhas especiais do script identificadas com a palavra "#SBATCH" no seu início, como por exemplo no script01.sh da figura 14 a opção "-n 5" (define n=5).

```
[pos@juno rcc]$ cat script01.sh
#!/bin/bash
# script para comando sbatch
# definir opções por defeito com linhas iniciadas por #SBATCH
#SBATCH -n 5      # número de processos
# executar programa paralelo com o comando srun
srun hostname
echo done!
[pos@juno rcc]$ sbatch script01.sh
Submitted batch job 184
[pos@juno rcc]$ cat slurm-184.out
node13
node13
node15
```

```
node17
node15
done!
[pos@juno rcc]$ cat script02.sh
#!/bin/bash
#SBATCH -n 5      # número de processos
# executar programa paralelo com o comando srun
srun sleep 4
echo done!
[pos@juno rcc]$ sbatch script02.sh
Submitted batch job 185
[pos@juno rcc]$ squeue
  JOBID PARTITION     NAME USER ST   TIME  NODES NODELIST
    185     total script02  pos  R   0:02      3 node[13,15,17]
[pos@juno rcc]$ squeue
  JOBID PARTITION     NAME USER ST   TIME  NODES NODELIST
[pos@juno rcc]$ cat slurm-185.out
done!
[pos@juno rcc]$
```

**Figura 14.** Exemplos dos comandos sbatch e srun aplicados a programas sequenciais.

A execução do script é feita num nó selecionado pelo slurm. Se no script existirem comandos do tipo "srun [opções] prog [args]" isso significa a execução de um programa paralelo com n processos, ou seja, são executadas n cópias de "prog" cada uma executada numa unidade de processamento diferente (por defeito um núcleo). Se "prog" for um programa sequencial, cada cópia executa independentemente das outras, como é o caso de "prog=hostname" no script01.sh da figura 14, que produz como dados de saída o nome do computador onde é executado. Se "prog" for um programa MPI, então cada cópia pode comunicar com as outras através das funções disponibilizadas pela biblioteca MPI, que graças à interface pmi, tem o ambiente de execução MPI configurado automaticamente pelo slurm.

Os comandos num script que não são derivados de srun são executados apenas uma vez, ou seja, só é criado um processo, como é o caso do comando echo no script01.sh da figura 14 que só produz uma linha de saída de dados.

No exemplo do script02.sh da figura 14 é utilizado "prog=sleep 4", um programa que simplesmente espera 4 segundos, o tempo suficiente para após a submissão do trabalho executar o comando squeue para consultar os trabalhos que se encontram nas filas de espera. O comando produz uma lista que contem os trabalhos que estão atualmente a ser executados juntamente com o tempo de execução já gasto e uma lista de trabalhos à espera que fiquem disponíveis os recursos necessários (por exemplo processadores, núcleos e/ou memória RAM) para serem executados.

A figura 15 mostra um exemplo simples de um programa MPI, uma versão paralela do usual programa "Olá mundo!", em que cada processo além da mensagem imprime o

número de ordem (rank) do processo e o número total de processos (size) do programa paralelo.

```
[pos@juno rcc]$ cat ola-mpi.c
/* ola-mpi.c
 * programa "Olá mundo!" em MPI
*/
#include <mpi.h>
#include <stdio.h>

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

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf("Olá mundo!  Eu sou o processo %d de %d\n", \
       rank, size);

    MPI_Finalize();
    return 0;
}
```

**Figura 15.** Versão MPI do programa "Olá mundo!".

Neste programa são utilizadas a função MPI_Init() para inicializar o processo perante a biblioteca MPI, as funções MPI_Comm_rank() e MPI_Comm_size() para o processo que as invoca saber o seu número de ordem e o número total de processos, e por fim a função MPI_Finalize() para indicar que o processo vai terminar. Caso os vários processos que constituem o programa paralelo MPI quisessem comunicar entre si tinham à sua disposição as funções MPI_Send() e MPI_Recv(), entre muitas outras, para enviarem e receberem mensagens entre si.

Um exemplo de compilação do programa MPI ola-mpi.c e respetiva execução com prog=ola-mpi é dado no script03.sh da figura 16. Como se pode verificar pelos dados de saída gerados, um programa paralelo MPI com n processos identifica-os com números de ordem de 0 a n-1.

```
[pos@juno rcc]$ mpicc -Wall -o ola-mpi ola-mpi.c
[pos@juno rcc]$ cat script03.sh
#!/bin/bash
#SBATCH -n 5      # número de processos
# executar programa paralelo MPI com o comando srun
srun ola-mpi
[pos@juno rcc]$ sbatch script03.sh
Submitted batch job 186
[pos@juno rcc]$ cat slurm-186.out
Olá mundo!  Eu sou o processo 1 de 5
```

```
Olá mundo!  Eu sou o processo 3 de 5
Olá mundo!  Eu sou o processo 4 de 5
Olá mundo!  Eu sou o processo 0 de 5
Olá mundo!  Eu sou o processo 2 de 5
[pos@juno rcc]$
```

**Figura 16.** Exemplo dos comandos sbatch e srun aplicados a programas paralelos MPI.

## 4. Conclusões

Este artigo complementa e termina a parte I apresentada em [Shirley 2017] sobre o projeto e montagem de um cluster (agregado) de computadores pessoais ligados entre si por uma rede privada ethernet. Esta arquitetura proporciona uma solução viável e relativamente económica para dois objetivos importantes: o acesso a uma plataforma de programação paralela e a um maior poder computacional.

A descrição dada foi baseada propositadamente numa versão com um sistema operativo de 32 bits, a distribuição Linux CentOS 6 de 32 bits, para que possa ser útil e alcançar o máximo de público alvo. Embora a tendência atual seja a de computadores com processadores de 64 bits, os computadores menos recentes com processadores de 32 bits e/ou com 3GB ou menos de memória RAM ainda abundam e podem beneficiar diretamente das instruções dadas neste trabalho. No entanto, as descrições dadas são flexíveis e muito facilmente adaptáveis à versão de processador e sistema operativo de 64 bits, assim como também facilmente adaptáveis a um número de nós superior a 250.

Em contraste com os computadores menos recentes estão os desenvolvimentos tecnológicos atuais em termos de processadores com o aumento do número de núcleos (cores), à data de escrita já existem processadores x86_64 com 32 núcleos e a tendência é continuarem a aumentar. A associação de computadores em cluster vem tornar o número total de núcleos disponíveis para processamento ainda maiores. Uma visita ao [top500] mostra que clusters com dezenas de milhares de núcleos já são relativamente comuns. Claramente o paralelismo já não pode ser ignorado na programação de sistemas, quer em larga escala quer em pequena escala, pois o processador mono núcleo é uma espécie em extinção e a programação sequencial uma abordagem que fundamentalmente desperdiça recursos, exceto para casos muito simples. Assim, o desenvolvimento e implementação de plataformas e ambientes de programação paralela assume um papel importante e um elevado valor didático ao permitir o contacto com este tipo de tecnologia e ao contribuir para a difusão da sua aprendizagem.

Por outro lado, o aumento do poder computacional trás também grandes benefícios à investigação, possibilitando que vá mais longe, ao permitir resolver problemas de maior dimensão e/ou mais rapidamente, proporcionando perspetivas mais abrangentes dos problemas e encurtando o tempo para obter respostas.

**Agradecimentos**

Agradeço à Universidade Aberta as condições que me proporcionou para a realização deste projeto, que foi um projeto interno da Universidade. Agradeço em particular aos meus colegas da SIFT/DCeT o apoio dado ao projeto desde o início e a cedência de computadores do Laboratório de Informática para o efeito, aos Serviços de Informática a cedência de material informático diverso e apoio técnico na configuração da rede e acesso remoto, à Divisão de Serviços Técnicos pelo apoio na gestão do espaço e mobiliário da sala técnica onde se realizou este projeto.

**Referências**

Barney B., "Message Passing Interface (MPI) (Tutorial)", Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/mpi/ [2018]

[CentOS Vault] http://vault.centos.org/

Granjal J. (2013), "Gestão de Sistemas e Redes em Linux", FCA.

intel (1999), "Preboot Execution Environment (PXE) Specification Version 2.1", www.pix.net/software/pxeboot/archive/pxespec.pdf [2008-09-29]

MPI Forum, "Message-Passing Interface Standard", http://mpi-forum.org/docs/

openmpi, "Open MPI: Open Source High Performance Computing, a High Performance Message Passing Library", https://www.open-mpi.org/

Balaji P. et al. (2010), "PMI: A Scalable Parallel Process-Management Interface for Extreme-Scale Systems". In: Keller R., Gabriel E., Resch M., Dongarra J. (eds) Recent Advances in the Message Passing Interface. EuroMPI 2010. Lecture Notes in Computer Science, vol 6305. Springer, Berlin, Heidelberg.
https://www.mcs.anl.gov/papers/P1760.pdf

PMI-2 API, "Design of version 2 of the Process Management Interface API used within MPICH", Argonne National Laboratory.
http://wiki.mcs.anl.gov/mpich2/index.php/PMI_v2_API, [2012-11-10]

pxelinux, "A Syslinux derivative, for booting from a network server using a network ROM conforming to the Intel PXE (Pre-Execution Environment) specification", https://www.syslinux.org/wiki/index.php?title=PXELINUX [2016-07-16]

Quinn Michael J. (2004), "Parallel Programming in C with MPI and OpenMP", McGraw-Hill Higher Education.

redhat (2011), "Red Hat Enterprise Linux 6 Managing Confined Services".

redhat (2012), "Red Hat Enterprise Linux 6 Developer Guide".

redhat (2013), "Red Hat Enterprise Linux 6 Deployment Guide".

redhat (2014), "Red Hat Enterprise Linux 6 Installation Guide".

Shirley P. (2017) "Projeto de um Cluster Didático para Programação Paralela e Distribuída (Parte I)", Revista de Ciências da Computação, nº12, Universidade Aberta. http://hdl.handle.net/10400.2/7686

slurm, "Workload Manager", https://slurm.schedmd.com [2018-10-24]

slurm AG, "Quick Start Administrator Guide", https://slurm.schedmd.com/archive/slurm-17.02.11/quickstart_admin.html

slurm UG, "Quick Start User Guide", https://slurm.schedmd.com/archive/slurm-17.02.11/quickstart.html

slurm docs, "Coletânea de documentos sobre os mais variados tópicos do software slurm", https://slurm.schedmd.com/archive/slurm-17.02.11/documentation.html

Sloan Joseph D. (2005), "High Performance LINUX Clusters", O'Reilly Media.

top500, "TOP 500 Supercomputers List", https://www.top500.org/

wikipedia, "Beowulf cluster", https://en.wikipedia.org/wiki/Beowulf_cluster [2017-11-17]

**Paulo Shirley**, Professor Auxiliar no Departamento de Ciências e Tecnologia (DCeT), Secção de Informática, Física e Tecnologia (SIFT). Coordenador da Licenciatura em Informática no triénio 2014–2016 e Vice-Coordenador do Mestrado Tecnologias e Sistemas Informáticos Web no biénio 2014–2015. Licenciado em Engenharia Electrotécnica e de Computadores em 1988 pelo IST-UTL. Obteve os graus de Mestre (perfil de Controlo e Robótica) e de Doutor em Eng. Electrotécnica e de Computadores, pelo IST-UTL em 1993 e 2003 respetivamente. Tem como áreas de interesse, a intersecção da Informática (Computer Science) com a área do Controlo Automático, nomeadamente a área da "Computação de Alto Desempenho" aplicada a problemas de otimização.

( esta página par está propositadamente em branco)