The Minerva Cluster User's Guide

Version 1.0

LaCED

Laboratório de Computação de Elevado Desempenho

Instituto Superior de Engenharia de Coimbra

Instituto de Investigação Aplicada

Instituto Politécnico de Coimbra

Miguel Couceiro and Luís Santos

March 15, 2017

Table of Contents

Introduction 1

Description of The Minerva Cluster 3

Hardware Resources 4

Hardware Layout 5

Storage Layout 6

Some Remarks on Compute Nodes Configuration 7

Software Resources 8

Theoretical vs. Measured Performance 11

How to Request Access to the Minerva Cluster 13

Rules for the username 14

Generating SSH Public/Private Key Pair Files 14

Unix/Linux OS 14

Windows 16

Accessing the Minerva Cluster 19

Text Mode Access 19

Unix/Linux 19

Windows 20

Disconnecting from Text Mode Access 22

Graphical Mode Access 22

Unix/Linux 22

Windows 24

Disconnecting from Graphical Mode Access 26

Running Programs in Interactive Mode 27

Command Line Interface 27
Setting Environment Variables 28
Running Software 29
Matlab 30
COMSOL Multiphysics 30
Running Programs in Non-Interactive mode 33
SLURM Partitions 33
Submitting Jobs 34
Some SLURM Options for the *sbatch* Command 36
License Reservation 39
Matlab Jobs 41
COMSOL Multiphysics Jobs 44
Task Farming Jobs 45
Monitoring and Cancelling Jobs 47
Job Scheduling 49

LaCED

INTRODUCTION

An HPC cluster is a computing system consisting of high performance components (CPUs, memory, network connections, and storage) that allows users to request a set of resources needed to run computationally intensive jobs, and that runs the jobs when the requested resources are available. The aforementioned jobs may consist of several tasks that are executed sequentially (without any king of parallelism), of tasks that can run in parallel on a single computer, as those not using Massage Passing Interface (MPI), or of tasks that can run in parallel spread over several computers interconnected by some sort of network intercommunication, as those using MPI. Regardless of the type of jobs submitted to the cluster, users do not have to be concerned with the time that a given job will take to complete in order to submit the next one. Instead, users submit all the jobs they want to run to a central workload manager, which, after scheduling the jobs based on a set of established priority rules, sends them to compute nodes when the requested resources are available.

From the above, it is clear that:

- An HPC cluster is not appropriate to run:
  - A single sporadic job that does not take too long to complete in a desktop or laptop computer (let us say, less than 30 minutes), because the time needed for the resources to be available can be much higher than that required to run the job on a simple desktop or laptop computer;
  - Any job requiring some sort of user interface or intervention during the computation process, because users do not have access to the jobs when they are running.

- An HPC cluster is appropriate to run:
  - A single job that takes too long (let us say, more than 5 hours) to complete on common desktop or laptop computers;
  - A set of jobs that takes too long to complete (let us say, 20 jobs each of which takes 15 minutes to complete) on common desktop or laptop computers;
  - Jobs that require a lot of computational resources not usually available on desktop or laptop computers (let us say, 10, 20 or 100 CPU cores, 50 GB of memory, etc.);

DESCRIPTION OF THE MINERVA CLUSTER

The Minerva cluster is a High Performance Computing (HPC) cluster composed of a management node (commonly referred to as head node) and twenty compute nodes, all interconnected through an InfiniBand (IB) FDR[1] network and a 1 Gbit/s Ethernet network. The IB network is used for computations and for sharing the storage between the head node and the compute nodes of the cluster (using NFS[2] over IPoIB[3]), while the1 Gbit/s Ethernet network is used for management and for sharing the storage with non-computing nodes (through NFS over Ethernet).The cluster also has a physical server that provides three virtualized nodes: one for routing the Ethernet communications, one for administration proposes, and one that serves as a login node and that is used by users to interact with the cluster. (For details on the hardware available, please consult the section *Hardware Resources*.)

All the nodes of the cluster run under the Linux Operating System (OS), namely the CentOS distribution, version 7.3. For resource management, the cluster uses the Simple Linux Utility for Resource Management, also known as SLURM Workload Manager[4], version 16.05.2. For management of the users' environment variables, the cluster uses the Environment Modules software, which is responsible for loading the environment variables required to run a given program/computation. Moreover, the cluster has a set of software installed. (For details on the software available, please consult the section *Some Remarks on Compute Nodes Configuration*.)

---

1 InfiniBand (IB) is a very high throughput and low latency computer-networking communication standard used in HPC, and capable of Remote Direct Memory Access (RDMA). In the Minerva Cluster, the IB network operates with 4 aggregate links at FDR speed (4xFDR). Each FDR link has a speed of 14 Gb/s, leading to a point-to-point speed of 56 Gb/s. (The effective rate for data transfer in 4xFDR links is equal to 54 Gb/s.)

2 NFS stands for Network File System, and is one of many protocols used to transparently share filesystems in a network. By "transparently share", one means that users in a client host access files on a server host as if they were in the client host.

3 Internet Protocol (IP) over InfiniBand (IB).

4 Detailed documentation can be found in https://slurm.schedmd.com/.

Hardware Resources

- 1× Head and Storage Node (Dell PowerEdge R720)
  - 2 Intel Xeon E5-2680v2 CPUs (10 cores each) @ 2.80 GHz, 15 MB cache, 8.0 GT/s QPI
  - 160 GB of registered and ECC DDR-3 memory (10×16 GB RDIMM, 1866MT/s, Standard Voltage, Dual Rank, ×4 Data Width)
  - 1 RAID Controller with 512 MB Non Volatile Cache
  - 146 GB on a RAID1 array (2x146 GB Hot-Plug SAS HDDs, with 15,000 rpm and 6 Gbit/s)
  - 1 HBA PCIe SAS controller with 2 external connectors with 4 ports each, with a transfer rate of 6 Gbit/s per port, connected to a storage unit (bellow)
  - 1Broadcom 5720 quad port 1 Gbit/s Ethernet work daughter card
  - 1 Broadcom 57810 dual port 10 Gbit DA/SFP+ Ethernet converged network adapter
  - 1 HCA single port InfiniBand FDR 56 Gbit/s adapter card(Mellanox ConnectXR-3 VPI MCX353A-FCBT)
  - 2 Hot-Plug power supplies in redundant mode
  - 1 Dell iDRAC7 management board

- 1× Storage Unit (Dell PowerVault MD3220), attached to the HBA PCIe SAS controller on the Management and Storage Node
  - 2 SAS connectors with 4 ports each, with a transfer rate of 6 Gbit/s per port
  - 2 RAID controllers, with 4 GB of aggregate cache memory
  - 28.8 TB of raw capacity (24×1.2 TB Hot-Plug SAS HDDs, with 10,000 rpm and 6 Gbit/s)
  - 2Hot-Plug power supplies in redundant mode

- 1× Storage Expansion Unit (Dell PowerVault MD1220), attached to the Dell PowerVault MD3220 Storage Unit
  - 28.8 TB of raw capacity (24×1.2 TB Hot-Plug SAS HDDs, with 10,000 rpm and 6 Gbit/s)
  - 2 Hot-Plug power supplies in redundant mode

- 20× Compute Nodes (Dell PowerEdge R720)
  - 2 Intel Xeon E5-2695v2 CPUs (12 cores each) @ 2.40 GHz, 30 MB cache, 8.0 GT/s QPI
  - 192 GB of registered and ECC DDR-3 memory(12×16 GB RDIMM, 1866MT/s, Standard Voltage, Dual Rank, ×4 Data Width)
  - 1 Integrated RAID Controller (PERC H710), with 512 MB Non Volatile Cache
  - 146 GB on a RAID1 array (2x146 GB Hot-Plug SAS HDDs, with 15,000 rpm and 6 Gbit/s)
  - 1 Broadcom 5720 quad port 1 Gbit/s network daughter card
  - 1 HCA single port InfiniBand FDR 56 Gbit/s adapter card (Mellanox ConnectXR-3 VPI MCX353A-FCBT)
  - 2 Hot-Plug power supplies in redundant mode
  - 1 Dell iDRAC7 management board

- 1× server for virtualized nodes (Dell PowerEdge R720)
  - 2 Intel Xeon E5-2660v2 CPUs (10 cores each) @ 2.20 GHz, 25 MB cache, 8.0 GT/s QPI
  - 128 GB of registered and ECC DDR-3 memory (8×16 GB RDIMM, 1866MT/s, Standard Voltage, Dual Rank, ×4 Data Width
  - 1 RAID Controller with 512 MB Non Volatile Cache
  - 300 GB on a RAID1 array (2×300 GB Hot-Plug SAS HDDs, with 15,000 rpm and 6 Gbit/s)
  - 1 Broadcom 5720 quad port 1 Gbit/s network daughter card
  - 1 Broadcom 57810 dual port 10 Gbit/s DA/SFP+ converged network adapter
  - 2 Hot-Plug power supplies in redundant mode
  - 1 management board Dell iDRAC7
- 1× InfiniBand switch (Mellanox MSX6036F-1SFR)
  - 36 non-blocking QSFP ports, each with a latency less than 200 ns and a data transfer rate of 56 Gbit/s
  - Aggregate data transfer rate of 4.032 TB/s
  - InfiniBand/Ethernet gateway
  - 2 Hot-Plug power supplies in redundant mode
- 2× Ethernet Switches, 1 Gbit/s (Dell PowerConnect 5548)
  - 48 ports, each at 1 Gbit/s
  - 1 GB of RAM and 16 MB of flash memory
  - 2 stacking HDMI ports
- 2× Uninterrupted Power Supplies (AEC NST5400030015)
  - Three-phase units with 30 kVA each
  - 40 batteries in of 9 Ah in each unit, plus 40 external batteries of 9 Ah per each unit
  - Operation in redundant mode
- 4× Power Distribution Units (Racktivity ES6124-32)
  - Three-phase units with real time, true RMS measurements at outlet level (power, apparent power, current, power factor, kVAh, consumption)
  - Individual power-outlet switching through the network

Hardware Layout

Figure 1 depicts the hardware and network layout of the Minerva cluster. Users can only access the login node through the public network.

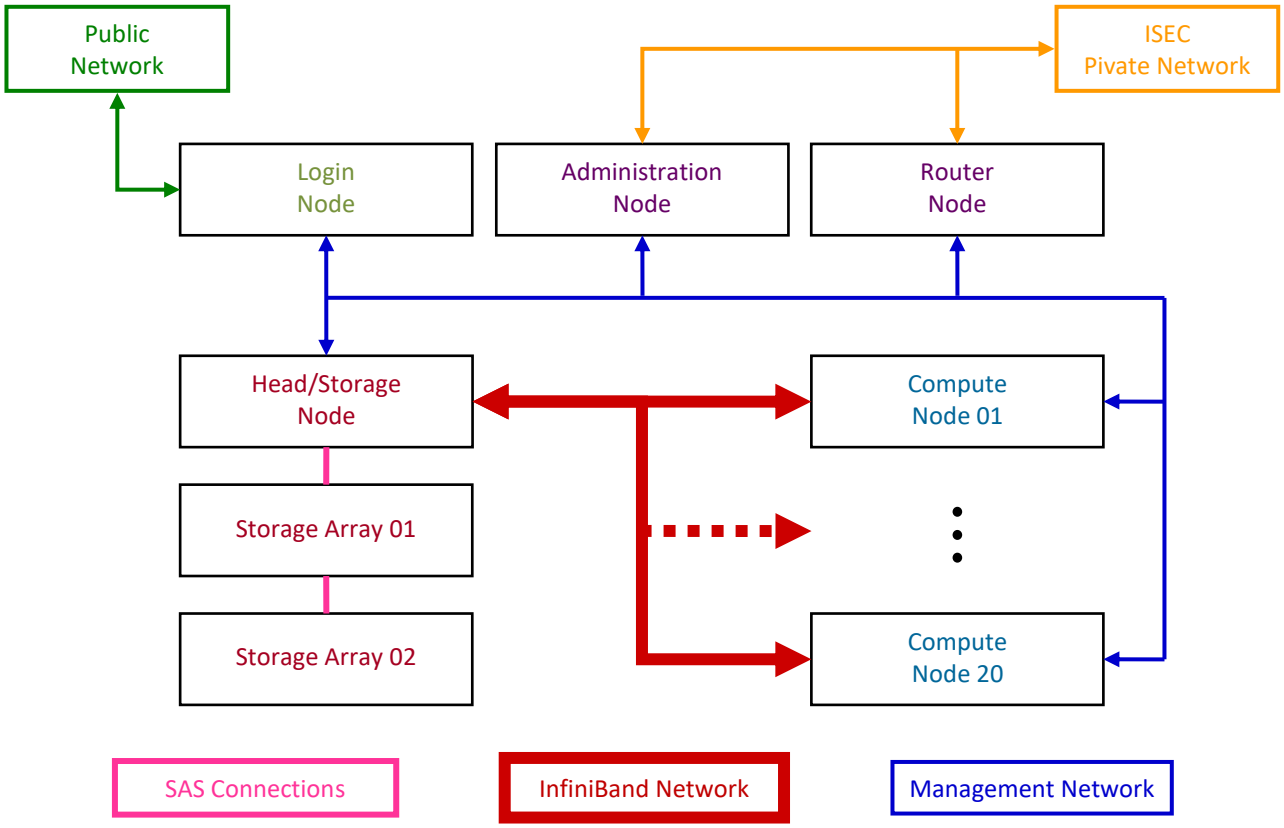

Figure 1: Hardware and network layout of the Minerva cluster

Storage Layout

Figure 2 depicts the layout of the storage space used in the Minerva Cluster. Each compute node has a local scratch space of ~ 93 GiB, mounted on a RAID1 system, which can be used to store temporary files. This space has no imposed quotas and is accessible through the directory "/scratch/local/*username*/", where *username* is your username in the cluster. This means that this scratch space cannot be accessed from within the login node, only through commands issued from tasks running on compute nodes. Moreover, a process running on a given compute node cannot access the local scratch space of other compute nodes.

The cluster also has a global scratch space of ~ 7.9 TiB mounted on a RAID0 system located at the two storage arrays, which can be used to store temporary files. This space has no imposed quotas, is shared by the head node to the compute nodes through the IB network (using NFS over IPoIB), to the login node through the Ethernet network (using NFS) and can be accessed through the directory "/scratch/global/*username*/", were *username* is your username in the cluster.

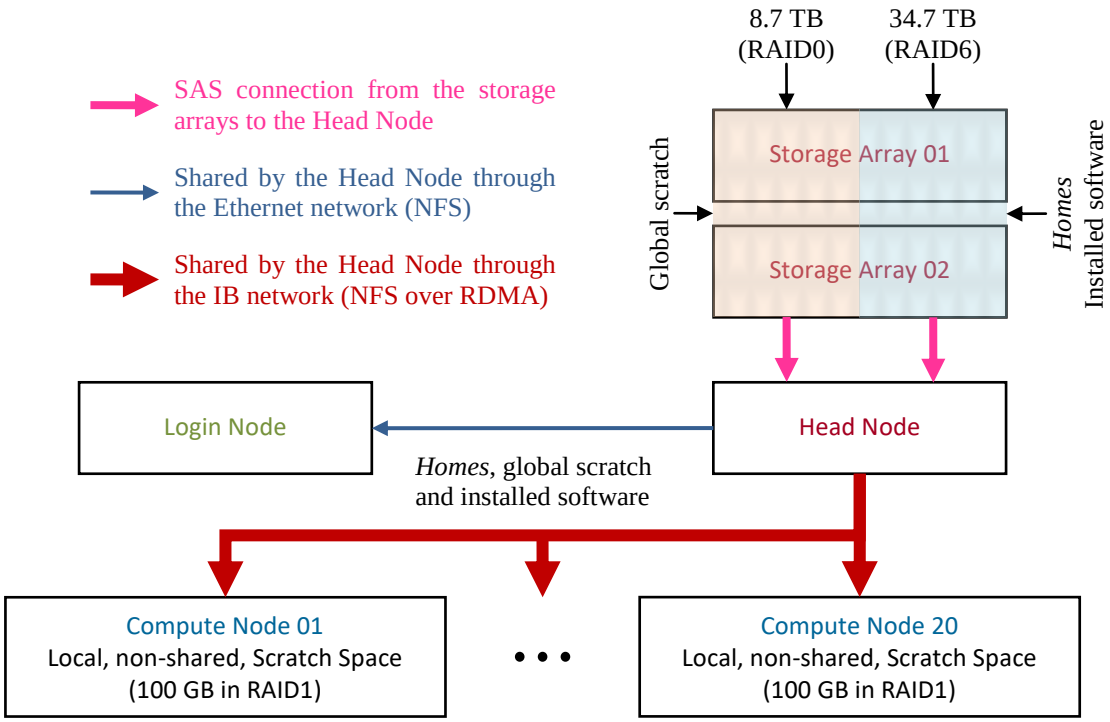

Figure 2: Storage layout of the Minerva Cluster.

The storage array also has a total of ~ 31.5 TiB of storage space for installed software and *home*[5] directories, which is mounted on a RAID6 system. These directories are exported by the head node to the login node through the Ethernet network (using NFS), and to the Compute Nodes through the IB network (using NFS over IPoIB).

Some Remarks on Compute Nodes Configuration

The compute nodes of the Minerva cluster have two physical CPUs (with twelve processing cores each)that communicate through a point-to-point interconnect named Intel Quick Path Interconnect (QPI).Concerning the memory, compute nodes have a total of 192 GiB equally distributed between two memory banks, each being closer to one of the physical CPU sockets.

Memory that is closer to a given CPU socket is said to be local to that CPU and its cores and is accessed by the memory controller of that CPU. Memory that is farther from a given CPU socket is said to be remote to that CPU and its cores, and is accessed by requesting access to the other CPU through the Intel QPI. This means that the access to remote memory is slower than the access

[5] In Unix/Linux systems, the term *home* directory is used to refer to the directory assigned to a given user for storing personal files. This directory is private, and in the Minerva Cluster only the user and *root* (the super user) have access to it.

to local memory. This configuration is known as Non-Uniform Memory Access (NUMA).

Figure 3 depicts the NUMA configuration of the compute nodes, with two NUMA nodes, along with the core numbering of each NUMA node and the Intel QPI between the two CPUs. In all compute nodes, NUMA node 0 is local to even numbered cores and remote to odd numbered cores, while NUMA node 1 is local to odd numbered cores and remote to even numbered cores.

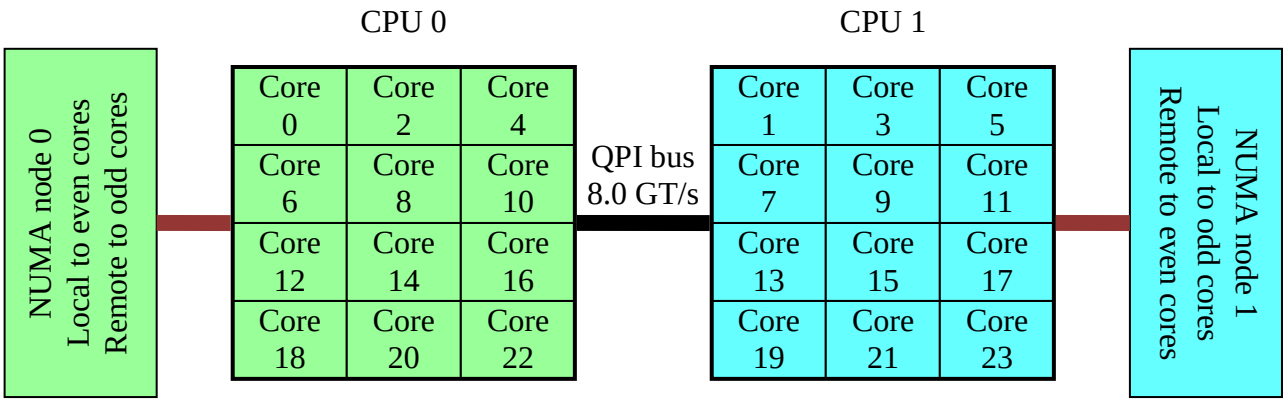

Figure 3: Layout of the NUMA architecture of the Compute Nodes. The core numbering represented in each CPU is the logical numbering set by the OS after startup.

Software Resources

The cluster uses the Linux OS (distribution CentOS, version 7.3), and has installed the following software:

- SLURM Workload Manager (resource manager, scheduler and accounting)
- Environment Modules (user environment manager)
- Compilers
  - GCC (versions 4.8.5, 4.9.3, 5.4.0, and 6.1.0)
    - C/C++, Fortran, Objective-C/C++, and Java
    - Not subjected to license availability
  - Intel Composer XE (version 2016.3.210)
    - C/C++ and Fortran
    - 2 network floating licenses
    - For academic use only

- Message Passing Interface (MPI) for parallel computing
  - OpenMPI
    - Version 2.0.1
    - Available for all versions of the aforementioned compilers
  - MPICH2
    - Version 3.2
    - Available for all versions of the aforementioned compilers
  - MVAPICH2
    - Version 2.2
    - Available for all versions of the aforementioned compilers

- Libraries
  - GMP (GNU Multiple Precision Arithmetic Library), version 6.1.0
  - ISL (GNU Integer Set Library), versions 0.11.1, 0.12.2, 0.14, and 0.16
  - MPFR (GNU Multiple-Precision Floating-point computations with correct Rounding), version 3.1.4
  - CLooG (library to generate code for scanning Z-polyhedra), versions 0.18.0 and 0.18.1
  - Intel DAAL (Data Analytics and Acceleration Library), version 2016.3.310
  - Intel IPP (Integrated Performance Primitives), version 2016.3.210
  - Intel TBB (Threading Building Blocks), version 2016.3.210
  - Intel MKL (Math Kernel Library – a BLAS library optimized for Intel processors), version 2016.3.210
  - Mellanox MXM (Messaging Accelerator – for IB communications), version 3.5.3092
  - Mellanox FCA (Fabric Collective Accelerator – for MPI software), version 2.5.2431
  - Mellanox HCOL (Fabric Collective Accelerator – for MPI software), version 3.6.1228
  - Mellanox KNEM (High-Performance Intra-Node MPI Communication), version 1.1.2.90

- Matlab (version R2016a)
  - 2 network floating licenses
  - Subject to academic licensing restrictions
  - The following toolboxes are available
    - Parallel Computing Toolbox (2 licenses)
    - Distributed Computing Server (license for up to 64 workers)
    - Matlab Coder (1 license)
    - Matlab Compiler (1 license)
    - Simulink (2 licenses)
    - Simscape (2 licenses)
    - Simulink Verification and Validation (2 licenses)
    - Partial Differential Equations (2 licenses)

  - Statistics (2 licenses)
  - Curve Fitting (2 licenses)
  - Optimization (2 licenses)
  - Global Optimization (2 licenses)
  - Neural Networks (2 licenses)
  - Fuzzy Logic (2 licenses)
  - Signal Processing (2 licenses)
  - DSP System (2 licenses)
  - Image Processing (2 licenses)
  - Computer Vision (2 licenses)
  - Mapping (2 licenses)
  - Bioinformatics (2 licenses)

- COMSOL Multiphysics (version 5.2a)
  - 1 network floating license for the base software and all modules
  - Subject to academic licensing restrictions
  - The following modules are available
    - Electrical – AC/DC
    - Multipropose – Optimization
    - Multipropose – Particle Tracing
    - Multipropose – Material Library
    - Interfacing – LiveLink for Matlab

Besides the aforementioned software, additional software packages can be installed, as long as the following conditions are all met:

- The software is for the Linux OS and compatible with the CentOS 7.3 distribution;
- The software is open source and free of charges, or being commercial, the user, project or institution requesting it pays the fees and the license maintenance when applicable (in this latter case, the software will be made available only to the users designated by the user, project or institution paying the fees);
- The software is appropriate for cluster computing, which means that it must be able to run without user intervention, namely, without user interfaces of any kind. (This does not mean that the software cannot have a user interface.)

Theoretical vs. Measured Performance

The theoretical peak performance $\left(P_{peak}\right)$ of an HPC cluster is measured in FLOPS[6], or more commonly, in GFLOPS (Giga-FLOPS) or TFLOPS (Tera-FLOPS), and can be computed (in GFLOPS)by

$$P_{peak} = Nodes \times \frac{sockets}{node} \times \frac{cores}{socket} \times clock\ speed\ [\mathrm{GHz}] \times \frac{\mathrm{FLOP}}{core}$$

The value obtained with the above equation does not take into account the performance losses due to network communications and memory usage. To account for that, performance tests with appropriate software must be conducted. The software commonly used to conduct those tests is the HPL package, which is a portable implementation of the Linpack's Highly Parallel Computing benchmark.

For the Minerva Cluster, the above equation holds a theoretical peak performance of

$$P_{peak} = 20 \times 2 \times 12 \times 2.4\,[\mathrm{GHz}] \times 8 = 9,216\ \mathrm{GFLOPS} = 9.216\ \mathrm{TFLOPS}$$

the performance obtained with the HPL package, using all the 480 cores at 90% memory occupancy, having been $P_{peak} = 8.041\ \mathrm{TFLOPS}$, representing an efficiency of 87.3%, which can be considered to be in accordance with expected values[7].

---

[6] FLOPS stands for Floating Point OPerations per Second, and is a measure of computer performance.

[7] For FDR InfiniBand, the theoretical loss due to network communications is estimated to be of 10%. (See, http://www.mellanox.com/related-docs/solutions/deploying-hpc-cluster-with-mellanox-infiniband-interconnect-solutions.pdf.)

How to Request Access to the Minerva Cluster

All researchers (teachers and students) of the Polytechnic Institute of Coimbra can access the Minerva Cluster for research (non-commercial) purposes, and they will have access to all the software described in the section *Software Resources*. Other users from outside IPC may also use the cluster, but fees may apply.

Users interested in using the cluster must send an e-mail to *admin@laced.isec.pt*, requesting approval for an account. The aforementioned e-mail must contain a small description of the project for which the computational resources are required, as well as the resources needed, including: storage space, maximum computing time needed for each running job to complete, total requested computation time, and additional software (if any) that may be necessary to install. Users may also provide additional information that they consider pertinent.

For an estimate of the maximum computing time needed for each job to complete, users can give the average time needed for the job to complete in a desktop or laptop computer (when possible).For evaluating the total computation time to be requested, users must multiply the number of jobs (*njobs*) to be submitted, by the number of cores needed for each job (*cpjob*) and by the average time that each job takes to complete (*atpjob*), that is,

$$requested\ time = njobs \times cpjob \times atpjob$$

For instance, if a given user estimates that it will submit 100 jobs (*njobs* = 100), each requiring (on average) 12 cores (*cpjob* = 12) for 12 hours (*atpjob* = 12), then the total requested time should be something like 14,400 hours. (Take into account that the total computational time available on the Minerva Cluster for non-leap years is equal to 2,204,800 hours.)

After following the abovementioned procedure to request access to the cluster, users will be informed if access was granted, which special resources will be allowed to use, and also with further instructions to activate the account. Along with these instructions, a *username* will also be requested for accessing the cluster, and the user's public SSH[8] key for accessing the cluster. (For security reasons, the cluster can only be accessed by using SSH.)

[8] SSH stands for Secure SHell, which "is a cryptographic network protocol for operating network services securely over an unsecured network." (*in Wikipedia*, https://en.wikipedia.org/wiki/Secure_Shell).

Rules for the username

The *username* must be short (no longer than 30 characters), must start with a lowercase standard character [a-z], followed by any combination of lowercase characters [a-z], underscores [_] and numbers [0-9].Uppercase characters, spaces, accentuated characters, cedillas, dots, and special characters will not be accepted.

Generating SSH Public/Private Key Pair Files

The method used to generate a pair of public/private SSH key pair files depends on the OS used. The following section describes how to create SSH public/private key pair files in Unix/Linux and in Windows. Those that already know how to create SSH public/private key pair files may skip this. Nevertheless, please read footnote 9 on page 14.

Unix/Linux OS

Open a terminal and change to your home directory (type "cd", without the quotation marks and press the *enter* key). Create a directory named ".ssh" (type "mkdir–p .ssh" without the quotation marks and press the *enter* key). Now, type

```
ssh-keygen –t rsa –b 2048
```

and press the *enter* key. The "-t" option allows for specifying the key type, which must be "rsa". The "-b" option allows to specify the length of the key (in bits), which must be equal to at least 2048. After issuing the above command, a message will appear asking for the directory to where the public/private key pair files should be saved. Enter "/home/*username*/.ssh/id_minerva" without the quotation marks, where *username* is your account name and *id_minerva* is the name of the RSA private key file. A second message will be displayed asking for a passphrase. It is advisable to introduce a strong passphrase[9], which can contain standard lowercase and uppercase characters ([a-z] and [A-Z]), numbers, spaces, hyphens, dots, underscores, and symbols like #, $,

[9] The passphrase is like a password for unlocking the SSH authentication mechanism, and not the password of the user account in the Minerva cluster. When authenticating through SSH using a password (which is not allowed in the Minerva Cluster), the password is sent (encrypted) through the network so that the remote server can verify the authenticity of the login. When using the SSH public/private key authentication mechanism, the passphrase is not sent through the network, being only used locally (on the computer from which the user is logging) to unlock the private key in order for the connection to be established. So, the passphrase is just an additional security measure that avoids unauthorized logins by someone that has gained access to the private key file of a given user.

and &, but should not contain accented characters and cedillas. After typing the passphrase (no character will be output to the display while typing the passphrase), just press the *enter* key. A new message will appear asking for passphrase confirmation. Just type the passphrase again, and press the *enter* key. Figure 4 depicts the output produced by the abovementioned procedure.

Now, go to the *.ssh* directory (type "cd .ssh" without the quotation marks and press the *enter* key). Type "ls -lah" without the quotation marks and press the *enter* key. Search for a file named "config". If the file exists, open it with your preferred text editor. If the file does not exist, create it (type "touch config" without the quotation marks and press the *enter* key) and open it with your preferred text editor. At the beginning of the file, add the following:

```
Host Minerva
    HostName        minerva.laced.isec.pt
    User            username
    IdentityFile    ~/.ssh/id_minerva
```

where *username* must be replaced by your username in the cluster. The last entry above is critical. You will not be able to connect if you do not specify the identity file exactly as shown above.

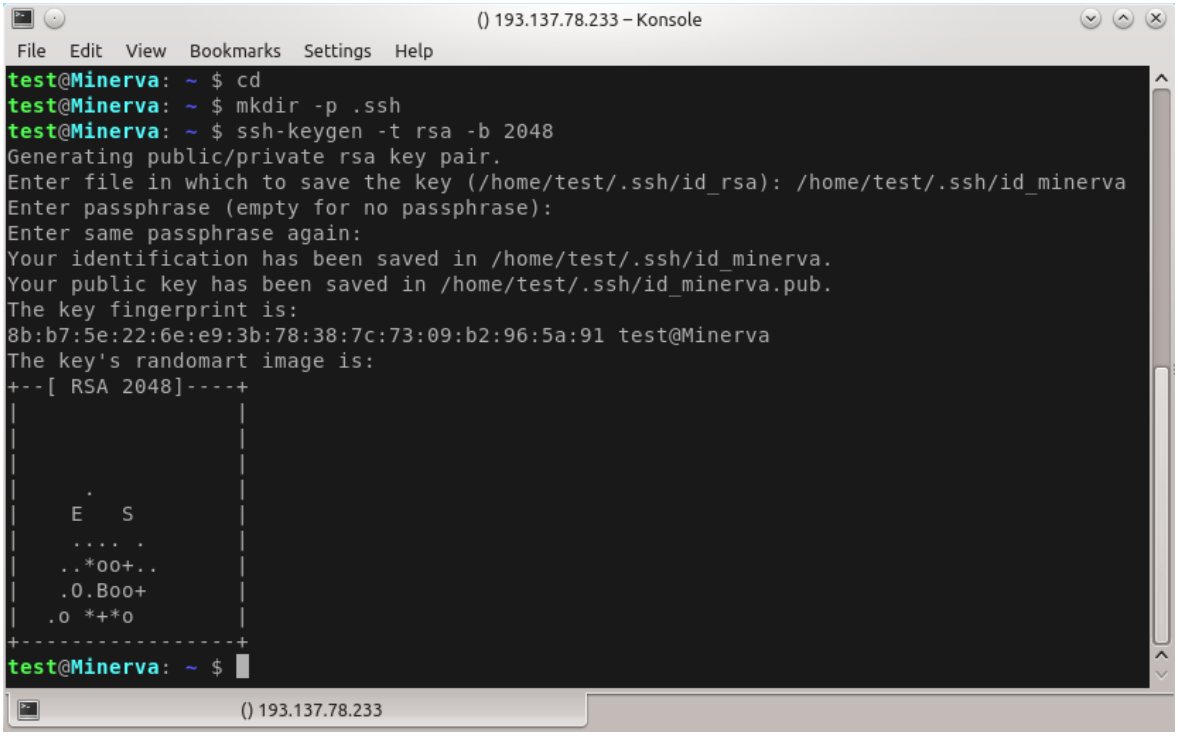

Figure 4: Generating the SSH public/private key pair in a Linux console.

Windows

To generate the public/private SSH key pair files in Windows, the PuTTY package is needed. Download it from https://the.earth.li/~sgtatham/putty/latest/x86/putty.zip. (It is necessary to download the ZIP compressed file containing at least the PUTTY.exe and the PUTTYGEN.exe files. The latter one is used for generating the public/private SSH key pair files, while the former will be used to access the cluster.)

After downloading the "putty.zip" file, extract it and run the PUTTYGEN.exe program (see Figure 5). Select the "SSH-2 RSA" option (center bottom of the "PuTTY Key Generator" program window), set the key size to at least 2048 (right bottom of the "PuTTY Key Generator" program window) and press the "*Generate*" button. After pressing the "*Generate*" button, the "PuTTY Key Generator" program window will look like Figure 6. Follow the instructions, namely, randomly move the mouse over the blank area of the PUTTYGEN program window until the completion bar reaches 100%. A third window is then displayed (see Figure 7) in which a passphrase can be

Figure 5: Main window of the "PuTTY Key Generator" program.

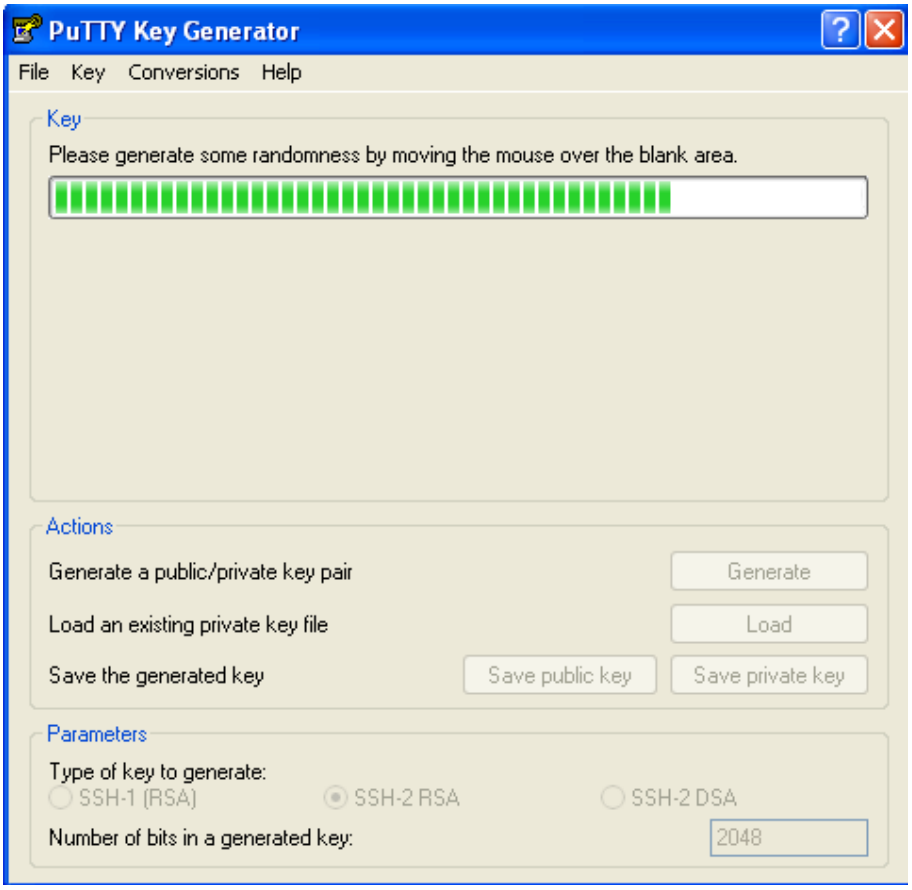

Figure 6: "PuTTY Key Generator" program window, depicting the instructions to move the mouse over the blank area in order to generate some randomness.

entered. Enter a passphrase and retype it. (Please refer to section *Unix/Linux OS* on page 14 for more information concerning the passphrase.)

After completing the aforementioned process, it is necessary to save the public and private keys to files. For that, press the "*Save public key*" button and save the SSH public key in a secure location, giving it the name "id_minerva.pub". Press the "*Save private key*" button and save the SSH private key file to a secure location, giving it the name "id_minerva.ppk" (it is not necessary to type the extension, since the "PuTTY Key Generator" program will add it automatically).Please remember that it is important to keep the private key in a secure location. See footnote 9, on page 14.

If a private key created in Unix/Linux is already available, the PuTTY private key can be generated with the "PuTTY Key Generator" program, by selecting the "Import Key" entry in the "Conversions" menu, and following the instructions.

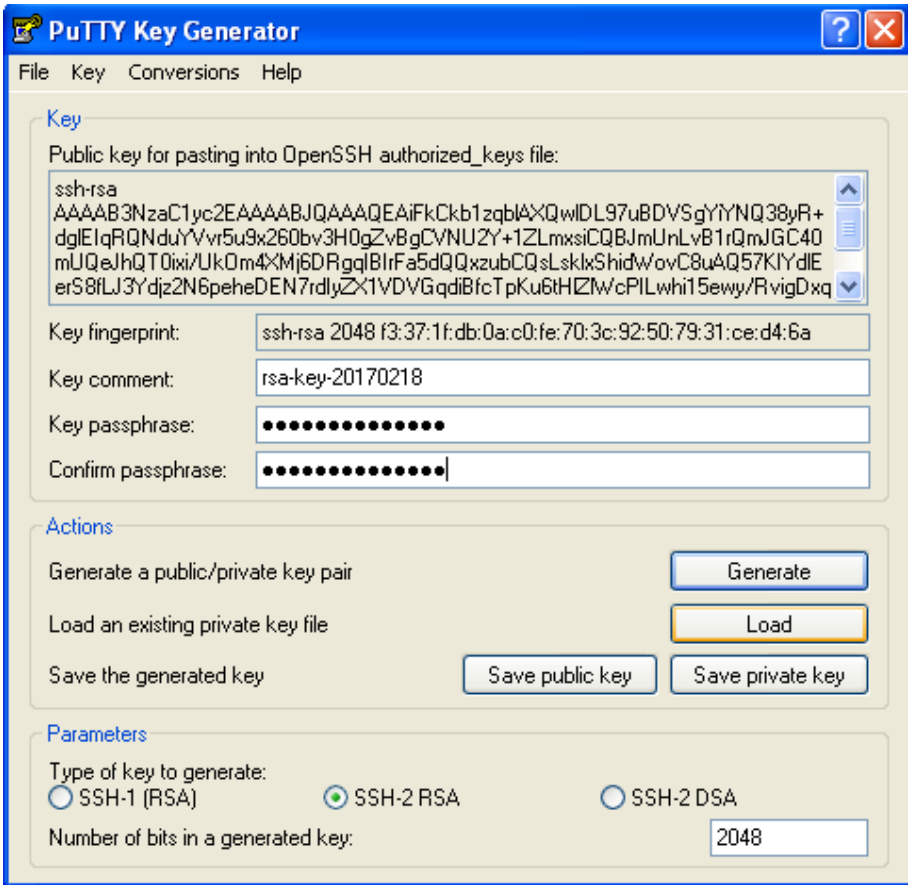

Figure 7: "PuTTY Key Generator" asking for a passphrase.

ACCESSING THE MINERVA CLUSTER

The cluster can be accessed both in text and graphical modes. However, due to bandwidth constraints, the access in graphical mode will eventually be restricted to users which unequivocally demonstrate that need. Moreover, although the access is allowed in graphical mode, the first access to the cluster must be in text mode[10]. So, the next section will explain how to access the cluster in text mode, both from Unix/Linux and from Windows.

Just one more note before continuing. The cluster has the public IP 173.137.78.233. This IP is mapped by a Domain Name System (DNS) server to the Fully Qualified Domain Name (FQDN) *minerva.laced.isec.pt*. Some Internet Service Providers (ISPs) and some institutions may block some DNS traffic. If this is your case, the access through the FQDN will fail with a message like "Could not resolve hostname minerva.laced.isec.pt", and you will have to use the IP address instead of the FQDN. Alternatively, configure your internet connection to use a public DNS server.

Text Mode Access

Unix/Linux

In Unix/Linux systems, the access in text mode is straightforward. Assuming that you have defined (in your local host) the "config" file as in page 15, just open a console and type (and then press the *enter* key)

```
ssh Minerva
```

or type (and then press the *enter* key)

```
ssh minerva.laced.isec.pt
```

Then, once asked, type your passphrase and press the *enter* key. Now you should have been logged in to the cluster and should have a text window with a command prompt that looks like "*username*@Minerva: ~ $", where *username* is your username.

[10] The graphical login requires that a password is defined for the user logging in. Since user accounts are created without a password, you must first login to the cluster in text mode to create the password, since the access in text mode does not require, neither use, a password. This will not be necessary if you only intend to login in text mode or if you were not allowed to login in graphical mode.

The first time you connect to the cluster from a given host, a long message concerning the host key will appear. Just answer “yes”.

Windows

In Windows systems, it is necessary to use the PuTTY software already mentioned above. Open PuTTY and, from the list box placed on the left side of the PuTTY program window (see Figure 8), select “Connection -> SSH -> Auth”. In the private key file text box, select the “Browse” button and search for the private key file (“id_minerva.ppk”). The option “Allow agent forwarding” can also be selected. Leave the remaining options as they are or set them as in Figure 8.

Go back to the list box placed on the left side of the PuTTY program window and select “Session” (the first option) as in Figure 9. In the “Host Name (or IP address)” text box type the username and the host FQDN as *username*@minerva.laced.isec.pt (without the quotation marks),

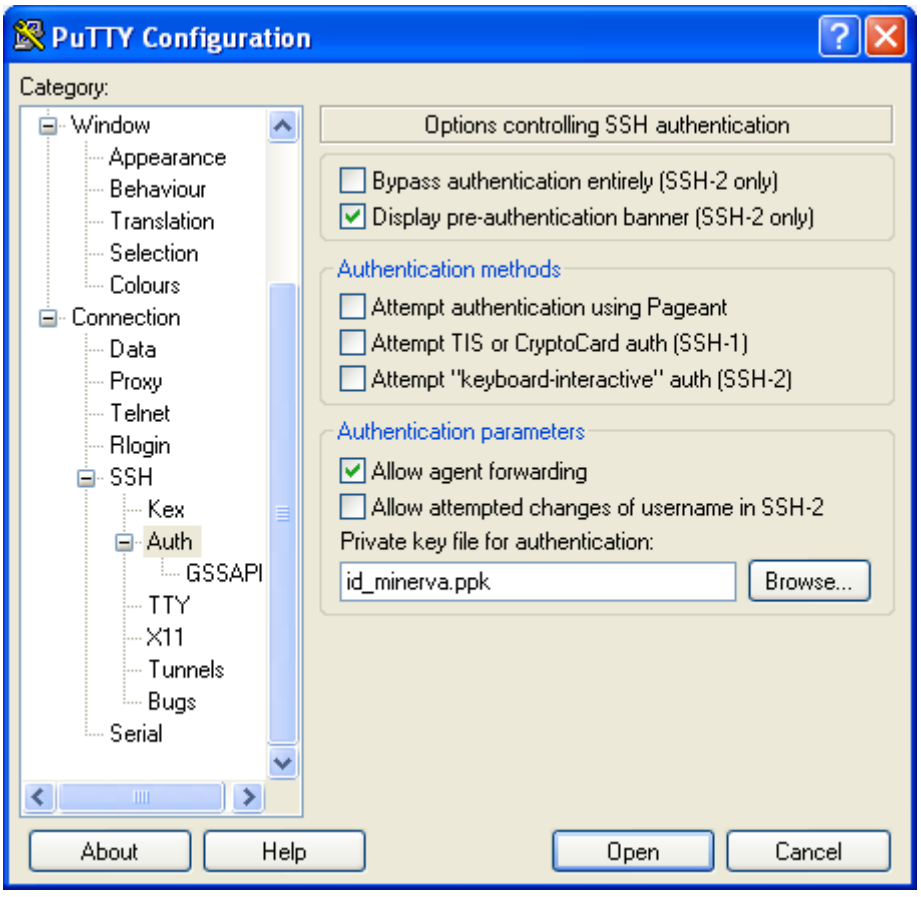

Figure 8: Configuring PUTTY to use SSH public/private key authentication.

replacing *username* with your username[11]. The "Port" text box should display 22. Leave it as is, or change it to 22 if a different port is displayed. On the "Connection type" radio buttons (below the "Host Name (or IP address)" text box), select SSH. In the "Saves Sessions" text box, type the name you wish to give to this connection (for instance, Minerva), and then press the "Save" button. Now, in the "Saved Sessions" list box, you should have an entry named "Minerva". To connect, just double-click the connection name that you created.

Then, once asked, type your passphrase and press the *enter* key. Now you should have been logged in to the cluster, and should have a text window with a command prompt that looks like "*username*@Minerva: ~ $", where *username* is your username.

The first time you connect to the cluster from a given host, a long message concerning the host key will appear. Just answer "yes".

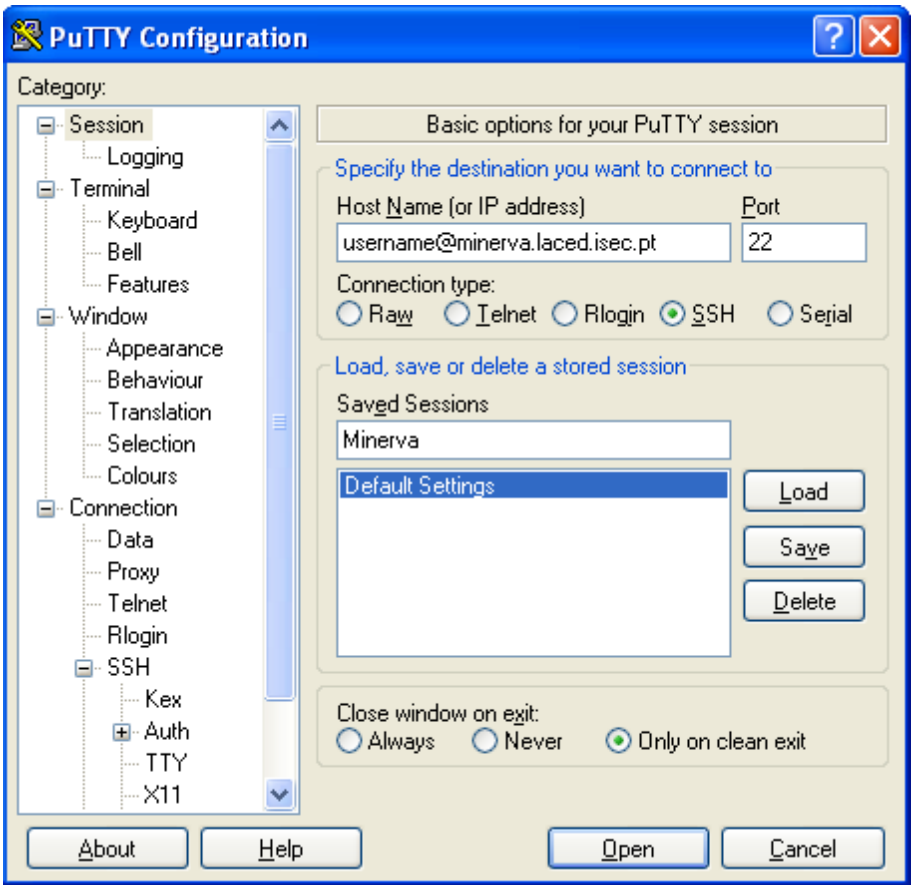

Figure 9: Configuring PUTTY to connect to the Minerva Cluster.

[11] You may also introduce only "minerva.laced.isec.pt". In this case, the window that will be displaced will ask you for the username.

Disconnecting from Text Mode Access

To disconnect from the text mode session, type *exit* at the command prompt and press the *enter* key, or just close the connection window.

Graphical Mode Access

To access the cluster in graphical mode, users must have a password defined in the Minerva Cluster. Since user accounts are created without a password, the first thing to do is to access the cluster in text mode and define one. For that, login to the cluster using one of the procedures mentioned in the section Mode above. After gaining access in text mode, set a password by issuing the command "passwd" (without the quotation marks) and follow the instructions[12]. After setting the password, you may close the opened connection.

Now that a password is defined in the cluster, it is necessary to have a program[13] that can connect to remote computers using the Remote Desktop Protocol (RDP), which is a proprietary protocol developed by Microsoft. However, and as a security measure the connection is only possible through an SSH tunnel, which must be created before connecting in graphical mode.

The next sections describe the procedures to be used in Unix/Linux and Windows systems, both to create an SSH tunnel and to login in graphical mode.

Unix/Linux

First open a console and create an SSH tunnel to the cluster:

ssh –f *username*@minerva.laced.isec.pt –L 5100:localhost:3389 -N

where *username* must be replaced by your username in the cluster. The "–f" option requests that ssh go to background before the command is executed. The "-L 5100:localhost:3389"[14] specifies that the port 5100[15] in the local host (the computer from which you are logging) be forwarded to the port 3389 on the remote host (the Minerva Cluster). The "-N" options instruct SSH not to

[12] The password can be equal to the passphrase that you have chosen for the SSH public/private key.

[13] The program to use depends on several factors, one of which is the OS present in the computer from which the user is connecting to the cluster. Windows systems come with a program for connecting to remote computers using RDP, which is usually in the program menu with the name of "Remote Desktop Connection". For Unix/Linux systems, there are several programs that can be used (for instance, KRDC for the KDE desktop).

[14] Instead of the word localhost, you can use the IP address 172.0.0.1.

[15] The local port 5100 should work for all users. If it doesn't, users may specify a different port of their choice.

execute a command on the remote host. Now that the tunnel is created, open the RDP client program and establish an RDP connection to “localhost:5100”. The exact procedure depends on the RPD client being used. In what follows, KRDC (for the KDE desktop) will be used to illustrate the procedure. (For other RPD clients, please set accordingly.)

Launch KRDC from the "K Menu -> Internet -> KRDC" or open a terminal and execute "krdc". In the "Connect to" dropdown button select "rdp". In the "Connect to" text box enter "*username*@localhost:5100", where *username* is your username in the Minerva Cluster, as in Figure. Now, you must have a dialog box, similar to the one shown in Figure, to set some options for the RDP connection. From the "Desktop resolution" drop down box, select the desired resolution (for instance, "Current Screen Resolution", if you want to use full screen mode). From the "Color depth" drop down box, select “True Color (24 Bit)”and uncheck the "RemoteFX: Enhanced visual experience" check box[16]. Press the “Ok” button to establish the connection. When

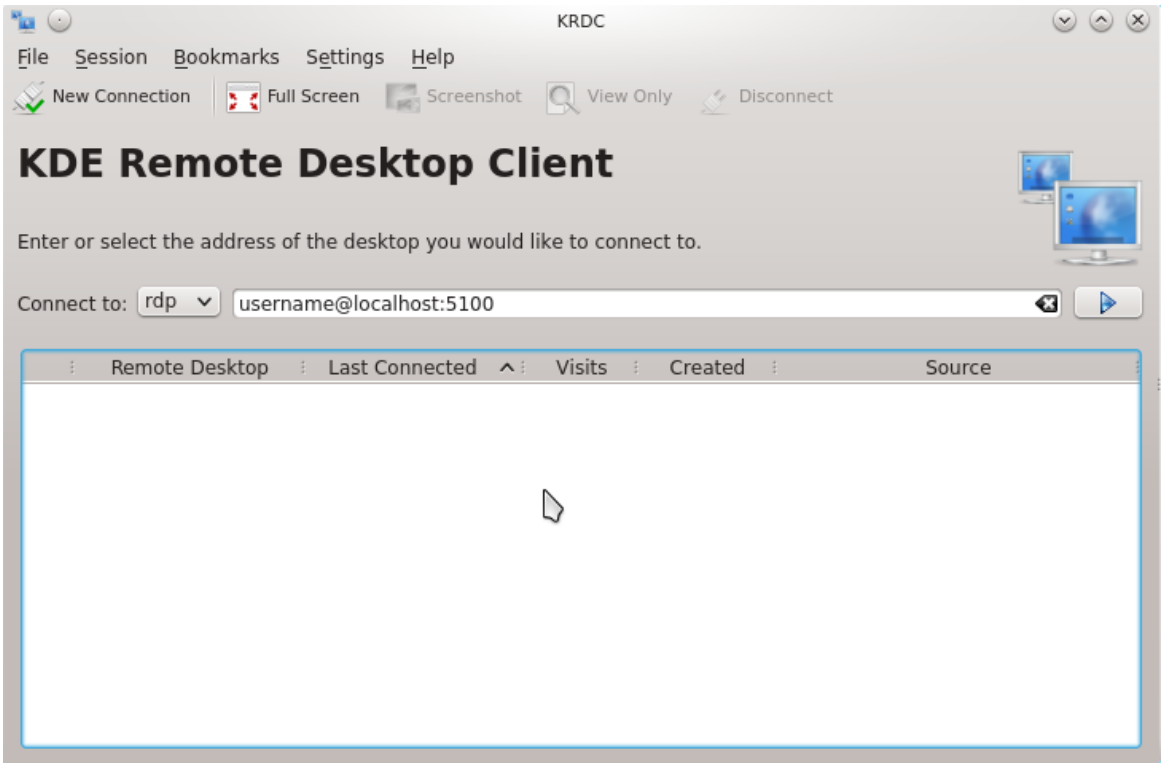

Figure 10: Establishing a connection to the cluster through KRDC.

[16] The connection will fail if either the color depth is higher than 24 bits, or the “RemoteFX: Enhanced visual experience” option is checked.

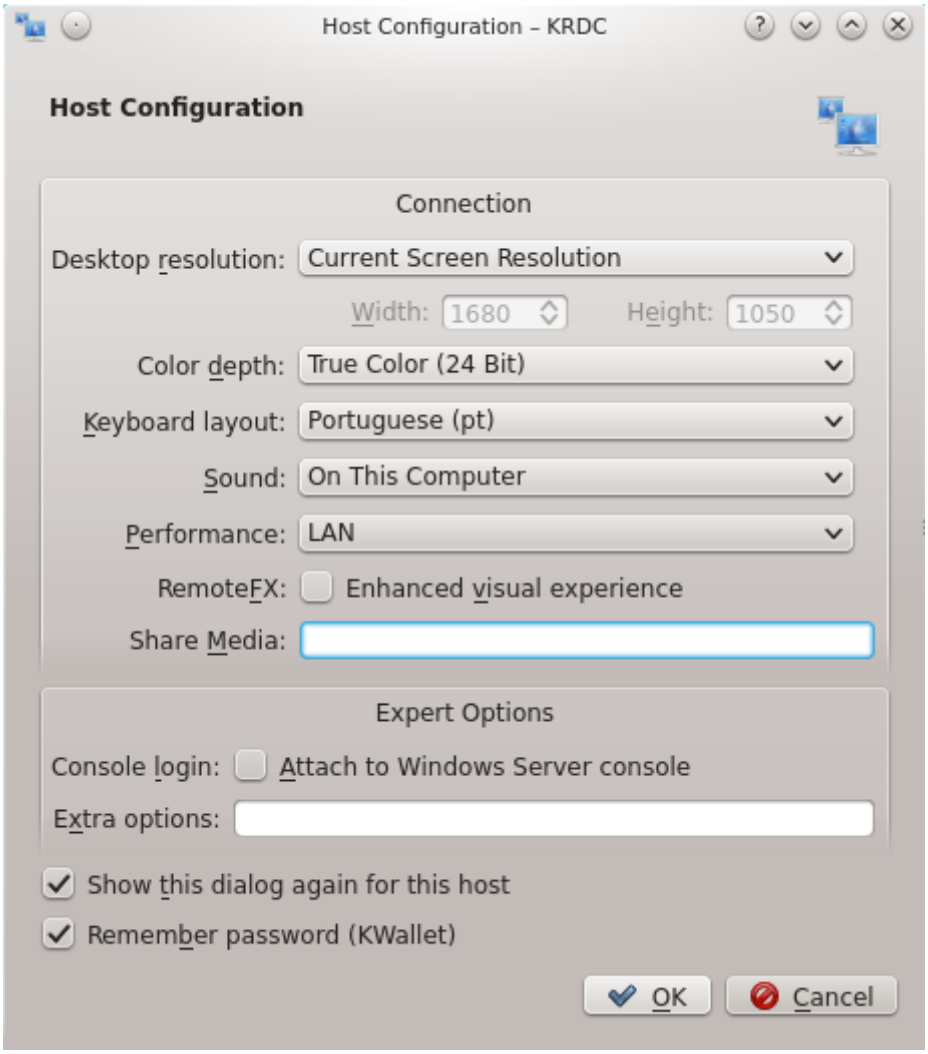

Figure 11: KRDC dialog box to set some options for the RDP connection.

asked, introduce your password and possibly the username (if in the “Connect to” text box of Figure 10 you entered only “localhost:5100”). If the connection fails, confirm if the SSH tunnel has been created: open a terminal window, issue the command "ps aux | grep ssh" and look for a line containing the command given to create the SSH tunnel. If the line exists, verify the connection options given to KRDC, otherwise create the tunnel and try connecting again.

Windows

First, open PuTTY to create an SSH tunnel to the cluster. For that, select the connection named “Minerva” (the one that you saved when creating the profile for text mode access) from the “Saved Sessions” list box and press the “Load” button (do not double-click the “Minerva” profile, neither press the “Open” button). Now, go to the list box placed on the left side of the

PuTTY program window, and select “Connection->SSH->Tunnels”. The dialog box will look like the one presented on Figure 12. In the “Add new forwarded port” section, enter 5100 in the “Source port” text box, and “localhost:3389” in the “Destination” text box. Leave the remaining options as in Figure 12, and press the “Add” button. A line as the one shown in Figure 12 will appear in the “Forwarded ports” list box. Go back to the “Session” option of the list box on the left side of the PuTTY program window and save the configuration as “Minerva – SSH tunnel” for future use. Double-click the saved configuration (“Minerva – SSH tunnel”) and login to the cluster in text mode to create the SSH tunnel.

Now that the SSH tunnel is created and active, open the “Remote Desktop Connection” program of Windows (Figure 13). Type “localhost:5100” in the “Computer” text box, type your username in the “User name” text box (if the text box is available) and then press the “Connect” button. The connection will be established and you will be asked to enter your username (if the username text box was not available) and your password to the cluster.

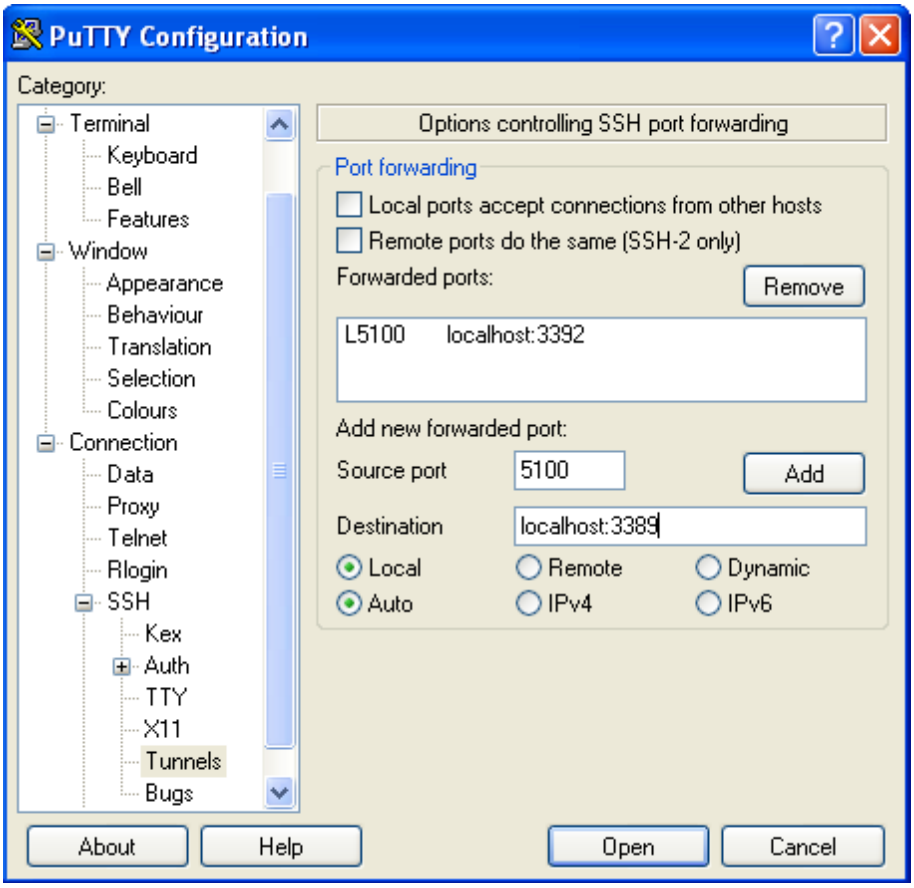

Figure 12: Configuring PuTTY to create an SSH tunnel.

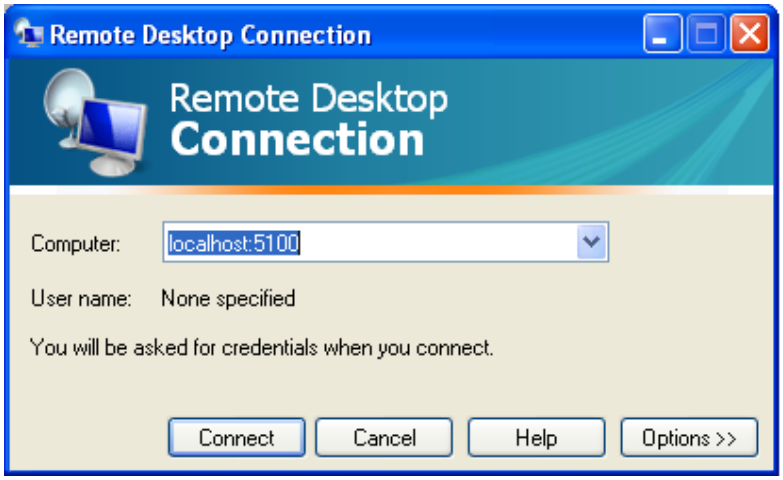

Figure 13: The Remote Desktop Connection interface.

Disconnecting from Graphical Mode Access

Two options exist to disconnect from graphical mode.

In the first method, go to the "Kickoff Application Launcher "icon (the icon in bottom left corner of the graphical login window) and press the left mouse button. A small window with five icons at the bottom will appear. The rightmost icon is "Leave". Select it and the pop up window will show the "Log Out" button. Click it to close the session. If you close the session in this way, the current opened windows and programs will be closed, and will not remain available for future use. In future sessions you will have to launch them again if you want to use them.

A second method to close the session consists of closing the session in the program used to connect through RDP, or simply closing the RDP program. Programs used to connect through RDP typically have a ribbon that hides on top of the desktop. Move the mouse to the top middle portion of the desktop to bring that ribbon down, and look for a button for disconnecting the session. If you close the session in this way, the desktop will be preserved for future connections.

RUNNING PROGRAMS IN INTERACTIVE MODE

HPC clusters are usually configured so that users can run processes in the so called interactive mode, in which users can interact with programs. This allows, for instance:

- Running software for compiling, debugging and profiling programs;
- Running software with user interfaces;
- Monitoring the state of jobs submitted to the cluster.

Running software interactively is usually accomplished in one of two ways: by submitting interactive jobs to the cluster and by running software in the login node. For now, the Minerva cluster allows only the latter mode of execution of interactive jobs. However, keep in mind that running jobs interactively, particularly in the login node, is not the preferred mode of operation of an HPC cluster and should be used with great discretion, besides being subject to severe constraints[17]. This is so because login nodes have scarce resources[18]shared by all users connected to the cluster at a given time.

The next sections provide information on how to run programs in interactive mode in the Minerva cluster. Please read them carefully and in sequence.

Command Line Interface

All software in the cluster must be executed through the command line interface, viz. from a Linux terminal. Users connected in text mode will already have a Linux shell to run software. Users that are logged in graphical mode must open a Linux terminal. For that, users may click the console icon (the third one from the left) in the quick launch bar located at the bottom of the graphical user interface. Alternatively, users may press the F12 key, which will bring a dropdown command line interface. (This latter method may not work if the F12 key is assigned to some action in the users host OS and/or if the program used for logging in graphical mode is not configured to grab all possible keys.)

---

[17] In the case of the Minerva cluster, processes started by users in the Login Node that are found to consume unreasonable resources for an unreasonable time, will be killed without notice. Moreover, users that, after being alerted, are repeatedly found consuming unreasonable resources for unreasonable times will have the access cancelled. (The notion of reasonable is left to the judgment of the cluster administration.)

[18] In the case of the Minerva cluster, the Login Node has only 4 CPU cores and 8 GiB of memory.

Setting Environment Variables

The environment variables necessary to run some installed software are not set by default. This is due to the fact that several versions of the same software can be installed in the cluster (e.g., the four versions of GCC referred to in section*Software Resources*), or the cluster may have installed different packages that provide the same functionalities (e.g., the several C/C++ and Fortran compilers, or the three MPI implementations referred to in section*Software Resources*), or a given software may have been installed with different options (e.g., the Intel Compiler referred to in section*Software Resources*), or linked with different compilers (e.g., the MPI libraries referred to in section*Software Resources*). So, users must first set the environment variables needed to execute the software they want.

To manage the environment variables needed by each installed program or software package, the cluster uses the Environment Modules software. This software makes use of the so called *modulefiles*, where the environment variables needed to run a given program or to use a given library are defined. Besides, each *modulefile* may define a list of required (conflicting) *modulefiles*, so that if a user loads a *modulefile_1* that requires (conflicts with) a *modulefile_2* that is not loaded (is loaded), the module program will not load the *modulefile_1* and informs the user that the *modulefile_2* is required by (conflicts with) the *modulefile_1*. Moreover, a given *modulefile* can also implement the mechanism necessary to load all *modulefiles* that it requires. This latter is the way in which the *modulefiles* of the Minerva cluster are preferably defined. However, some deliberate exceptions may exist.

The interface for the Environment Modules software is provided by a program named "module", which has several subcommands, some of which must be followed by a parameter. The command is issued as

module *subcommand parameter*

The most used subcommands[19] are "avail", "help", "load", "unload", "list", and "purge", and are summarized inTABLE 1.

[19] The full list of allowed subcommands and options is extensive and out of the scope of the current manual. Interested users may refer to the documentation page of the Environment Modules software, located at http://modules.sourceforge.net/tcl/module.html.

TABLE 1: Most used module subcommands. The parameter modulefile *(if needed) is the full name of the module file, as returned by the "avail" subcommand.*

Subcommand	Parameter	Comment
avail		Returns a list of all module files available
help	*modulefile*	Returns the help text defined in the module file
load	*modulefile*	Loads a module file
unload	*modulefile*	Unloads a module file
list		List all loaded modulefiles
purge		Unloads all loaded module files

As an example, the "module avail" command will return several module files, one of which is "Compilers/GCC/4.9.1". To load this module file issue the command

module load Compilers/GCC/4.9.1

To unload the "Compilers/GCC/4.9.1" module file issue the command

module unload Compilers/GCC/4.9.1

To see the details of the "Compilers/GCC/4.9.1" module file issue the command

module help Compilers/GCC/4.9.1

As a final notice, the environment variables loaded by the Environment Modules software are set only in the shell in which they were loaded. So, if you have two shells opened (*A* and *B*) and load a module file in shell *A*, the environment variables will be set only for shell *A*.

Running Software

To run a given program in interactive mode, you must load the needed modules and then

issue the command to run the software. In the following, only the steps needed to run COMSOL and Matlab will be addressed. (It is assumed that users that want to compile source code, with or without using of the MPI libraries, know the steps needed to accomplish those tasks.)

Matlab

For running Matlab in interactive mode, load the "Programs/matlab-R2016a" module file:

```
module load Programs/matlab-R2016a
```

Once the Matlab module is loaded, issue the command

```
matlab
```

If you logged in text mode and an error message concerning the graphical interface is displayed, issue the command

```
matlab –nodesktop –nosplash –nojvm
```

When the *matlab* command is issued, the system creates a 60 minutes license reservation[20] and executes the Matlab software. After that time, the system will automatically close the Matlab software without a notice. This is so because only two Matlab licenses are available for all users of the cluster. If Matlab licenses are not available, users will be informed and asked to try again later.

COMSOL Multiphysics

For running COMSOL Multiphysics in interactive mode you must be logged in graphical mode. Once logged in graphical mode, load the "Programs/comsol-5.2a" module file by issuing the command

```
module load Programs/comsol-5.2a
```

Once the COMSOL module is loaded, issue the command

```
comsol
```

When the *comsol* command is issued, the system creates a 240 minutes license reservation[20]

[20] Exceptionally, a longer reservation can be created for a specific day and to start at a specific time. (Requests for reservations should be sent to the e-mail address admin@laced.isec.pt.)

and executes the COMSOL software. After that time, the system will automatically close the COMSOL software without notice. This is so because only one COMSOL license is available for all users of the cluster. If the COMSOL license is not available, users will be informed and asked to try again later.

RUNNING PROGRAMS IN NON-INTERACTIVE MODE

The preferred and main mode of operation of the Minerva cluster is by submitting jobs[21] to the SLURM Workload Manager, hereinafter referred to as SLURM, which, as stated in the overview section of the SLURM web page, is a cluster management and job scheduling system for Linux clusters with three key features:

- Has the responsibility for allocating access to cluster resources to users for a given time duration so as they can perform their work;
- Provides a framework for starting, executing, and monitoring work on the set of allocated resources;
- Arbitrates contention for resources by managing a queue of pending work.

In what follows, the basic concepts of SLURM needed to use the Minerva cluster will be addressed. If you do not have experience using HPC clusters with SLURM, read them carefully and in sequence and refer to the SLURM documentation web page whenever you need to. Otherwise, please read the *License Reservation* if you need to run jobs requiring software licenses.

SLURM Partitions

SLURM uses the concept of partitions for grouping resources that can be allocated to jobs being submitted, and may include constraints such as maximum number of nodes that can be allocated by a job, maximum memory that a job can use, maximum execution time, and users or group of users that can use the partition.

In the Minerva cluster, four partitions are defined: *generic* (the default partition), *shortterm*, *longterm* and *highmem*. Each of these partitions define the maximum number of compute nodes that can be allocated by jobs submitted to the partition, the default and maximum memory per CPU that can be used by jobs submitted to the partition, the default and maximum execution time allowed for a single job submitted to the partition, a priority for the partition, and the groups of users that are allowed to use the partition. TABLE 2resumes all the definitions of the abovementioned partitions.

[21] A job is a computational work that is submitted to the cluster and for which the user requests computational resources (such as Compute Nodes, CPU cores and memory). Each job may be subdivided into several tasks, each of which may run in one or more CPU cores.

TABLE 2: SLURM partitions defined in the Minerva cluster. The default values are applied if the user does not specify a value for the resource.

Resources		Partitions			
		generic	**shortterm**	**longterm**	**highmem**
Maximum Number of Nodes per Job		10	20	5	5
Memory per CPU [MiB]	Default	2048	2048	2048	4096
	Maximum	7920	7920	7920	98000
Allocation Time [hours]	Default	48	12	96	24
	Maximum	96	24	192	96
Priority		50	100	25	25
Groups of Users Allowed		all	all	slurmspecial	slurmspecial

Submitting Jobs

To submit jobs to the cluster you must use the SLURM *sbatch* command. The easiest way to accomplish this is by creating a text script file[22] and pass it as an argument to *sbatch*:

sbatch *scriptFile*

Where *scriptFile* is the name of the text script file defining all that is necessary to run the job, including, among many others, the resources needed to run the job, the maximum time for execution, and the partition to which the job must be submitted. Moreover, the script file must also contain the commands that would be given if the job was run in interactive mode, namely, the commands necessary to load the environment variables needed by the job being submitted and commands needed to run the job. For this, read the section *Setting Environment Variables* of the chapter *Running Programs in Interactive Mode*.

[22] The file can be created with a text editor of your choice (the cluster has a very good one named *kate* with syntax highlighting, which can be used only in graphical access mode). However, if you have to generate a considerable number of submission files for performing similar computations, it is preferable to automate the process of creating the script file by recurring to programming (e.g, a bash script or a Matlab script).

The script text file has the following structure:

```
#!/bin/bash

#SBATCH --time=24:00:00
#SBATCH --partition=generic
#SBATCH --licenses=name@HeadNode:N
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --mem-per-cpu=4096
#SBATCH --mem_bind=local
#SBATCH --output=/home/username/Documents/job.out
#SBATCH --error=/home/ username/Documents/job.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=someone@somewhere

cd $SLURM_SUBMIT_DIR

module load modulefile_1
module load modulefile_2
...

./full_path_to_the_command_to_execute_your_code
```

The first line, which must start with a shebang (#!) followed by the full path of a Linux shell interpreter (/bin/bash in the example). This line instructs the OS which interpreter should be used to execute the script. In the above example the interpreter is bash (/bin/bash), but it could be a different one.

For those used with bash scripting, the lines containing "#SBATCH" may be confusing, since lines started with an hash (#) are considered comments and are not executed by the bash interpreter. However, the *sbatch* command interprets those lines as containing options. Please refer to section *Some SLURM Options* below for the meaning of the options given above.

Following the options you can type any valid Linux command, including changing to directories, exporting environment variable, and the command used to execute your code.

In the example above the first non-empty line following the SLURM options will change the directory of the batch script to that from which the batch script was submitted (the SLURM_SUBMIT_DIR environment variable). (At the end of the *sbatch* documentation page you may find a list of all environment variables set by SLURM in the batch script shell.)

The two lines following the above mentioned one, will load all the modules needed by your job. This is necessary because the environment variables of the shell from where the job is submitted are not exported to the bash script environment. So, if your job needs some modules to be loaded and you do not load them in the batch script, the job will fail to execute.

The last line of the batch script example given above is the line in which you effectively instruct SLURM to do what you want him to do. So, this line must contain the exact command that you would give from the command line to execute the program you want. If the full path to the program executable file is not defined in the environment of the batch script, then you must give the full path to the executable file, preceded by a dot. Moreover, the full path to the command must end with the name of a Linux executable file.

Some SLURM Options for the *sbatch* Command

The full set of options available for the *sbatch* command is out of the scope of this manual. Users are encouraged to consult them in the *sbatch* documentation page. In what follows only the options given above (which are probably more than those that you will ever need to set) will be mentioned:

- **#SBATCH --time=24:00:0**

  Maximum time for which the the resources should be allocated.

  The resources will be released as soon as the job ends or the execution time reaches the specified value, in which case the job (if still running) will be killed without notice.

  With *D* for days, *HH* for hours, *MM* for minutes, and *SS* for seconds, the valid time formats specifications are: *D-HH*:*MM*:*SS, D-HH*:*MM*, *D-HH*, *HH*:*MM*:*SS*, *MM*:*SS*, *MM*.

  If this option is omitted, the default time of the requested partition will be used (see TABLE 2).

- **#SBATCH --partition=generic**

  Partition to which the job is to be submitted.

  One of the four partitions in TABLE 2 can be used. Remember that two of them require special permission, which you may not have.

  If this parameter is omitted, the *generic* partition will be used.

- **#SBATCH --licenses=*license1*@headnode:*N1*,*license2*@headnode:*N2***

  Comma separated list of software licenses that are required to run the job.

  license1 and *license2* are the internal names by which SLURM recognises the licenses to be reserved and *N1* and *N2* are the number of licenses requested for each license name. When requesting a license, this latter parameter is mandatory, even if it is equal to one. The "@headnode" identifies the server that contains the information concerning the available licenses and must be given exactly as is.

  If this parameter is omitted, no licenses will be allocated to the job. For further information on software licenses, please refer to section *License Reservation*. This section not only explains why it is so important to request licenses for those jobs that need them, but also lists all the available licenses in the cluster.

- **#SBATCH --ntasks=1**

  Total number of CPU cores that must be reserved for running the job.

  For jobs that require only one CPU core, omit this option or set it to one. For parallel jobs that do not use MPI, set this option to the number of CPU cores used by the job (between 2 and 24) and set the "--nodes" option below to one, otherwise your job may be scheduled across several nodes and will run slowly as all tasks will run in the CPU cores allocated in a single compute node (the master node), the remaining allocated resources being wasted. For parallel jobs that use MPI, set this option to the total number of cores used by the job (between 2 and the maximum number of cores allowed on the partition to which you are submitting the job, which is equal to 24 times the partition maximum number of allowed nodes, as given in TABLE 2).

  If this option is omitted, one core will be used.

- **#SBATCH --nodes=1**

  Number of Compute Nodes that must be reserved for running the job.

  For jobs requesting a single CPU core (--ntasks=1), omit this option or set it to one. For parallel jobs requiring more than one CPU core but that do not use MPI, set this option to one (see the "--ntasks" option). For parallel jobs that use MPI, omit this option or set it to the number of desired nodes, between 1 and the maximum number of allowed nodes on the partition to which you are submitting the job (see

TABLE 2), but less than or equal to the total number of requested CPU cores.

If this option is omitted, no default value is assumed and SLURM may evenly spread the number of requested CPU cores across the maximum number of nodes allowed.

- **#SBATCH --mem-per-cpu=4096**

  Minimum amount of memory per allocated CPU core that is required to run the job. The default unit is MiB, but a different unit can be given by adding one of the following suffixes: “k” or “K” for kiB, “m” or “M” for MiB, “g” or “G” for GiB.

  If you specify a value higher than the maximum allowed memory per CPU defined for the partition to which you are submitting the job (see TABLE 2), then the job will be rejected at submission time. (Notice that if your job allocates more memory per CPU core than the maximum allowed for the requested partition, the job will be killed without notice.)

  If this option is omitted, the default memory per CPU for the partition to which you are submitting the job will be used (see TABLE 2).

- **#SBATCH –mem_bind=local**

  Scheme that should be employed to bind tasks to memory.

  Recommended values are “none” or “local”. The latter ensures that the memory allocated to a task is located in the same NUMA node as the CPU core in which the task is running, while the former allows SLURM to allocate memory that is considered remote to the CPU core in which the task is running. (Please refer to section *Some Remarks on Compute Nodes Configuration* on page 7 in which a brief explanation of NUMA memory locality is given.)

  If this option is omitted, the default value of “none” will be used.

- **#SBATCH --output=*/full_path_to_directory/job.out***

  Full path of the filename to which the standard output of the batch script shell should be redirected.

  Replace “*/full_path_to_directory*” by the full path to a valid directory for which you have write permissions (typically located in your home directory) and “*job.out*” by a unique filename that allow you to easily relate the file with the submitted job.

If these parameter is omitted, SLURM will redirect the standard output and the standard error of the batch script to a file named "slurm-%ID.out" (%ID will be replaced by the job ID) located in the directory from which the job was submitted.

- **#SBATCH --error=*/full_path_to_directory/job.err***

  This option is similar to the "--output" option given above, but for the standard error of the batch script shell.

  If this option is omitted, the standard error of the batch script shell will be written to the same file to which the standard output of the batch script is written.

- **#SBATCH --mail-type=*option1,option2,…***

  Comma delimited list of message types that will be sent to the e-mail address specified by the option "--mail-user" (below).

  Valid options are NONE (no message will be sent), BEGIN (job execution has started), END (job execution has ended), FAIL (job execution has failed), REQUEUE (job has been requeued), ALL (BEGIN,END,FAIL,REQUEUE), TIME_LIMIT (job has reached the allowed time limit for execution), and TIME_LIMIT_XX (job as reached *XX*% of the allowed time limit for execution, with *XX* equal to 50, 80, or 90%).

  If this option is omitted, no message will be sent.

- **#SBATCH --mail-user=*someone@somewhere***

  E-mail address to which messages specified by "--mail-type" should be sent.

  If this option is omitted, no e-mail messages will be sent.

License Reservation

When a given program that requires a valid license starts to execute, the software verifies the existence of available licenses by running its license manager mechanism, which is usually accomplished by specialized independent software for license management. If a license is not available the program ends prematurely, otherwise the program continues to execute normally. So, to be able to truly perform license management in the cluster, SLURM should have the capability to communicate with license management software, which it has not. However, SLURM has a mechanism that, if used properly, significantly reduces the risk of sending jobs to execution that will terminate prematurely due to license availability issues. This mechanism involves defining

licenses as consumable resources. These resources are defined by a unique name identifying the software and the total and available number of licenses present at a given time.

When a job requesting for licenses (with the "--licenses" option) reaches the top of the submission queue, SLURM verifies internally the license availability. If the number of available licenses is less than the total number of requested licenses, the job is sent to the compute nodes for execution and the number of licenses in use is increased. If the number of requested licenses is not available, the job will be held at the top of the submission queue in a state of pending due to resource availability, until all the requested resources (including licenses) are available. When that happens, the job is sent to the compute nodes for execution. So, if you do not request licenses for jobs requiring them, one of two things may happen:

- When the job is sent to the compute nodes no license is available and the job terminates prematurely.
- When the job starts execution a license is available and the software starts its normal execution. Since SLURM did not know that a license is in use, it does not update the internal license availability count. Now, if another user submits a job to the cluster and requests for a license, SLURM thinks that the license is available and sends his job for execution. However, the job will end prematurely because all the licenses are in use.

The license resources available on the Minerva cluster are resumed on *TABLE 3*.

TABLE 3: Licenses available for reservation.

Software Package	Resource Name	Number
Matlab	matlab	2
Matlab Distributed Computing Server	matlab-dsc	64
COMSOL Multiphysics	comsol	1
Intel Composer XE	intel-composer-xe	2

Matlab Jobs

Although Matlab can be used to run code in batch mode (without requiring a user interface), this is not allowed in the Minerva cluster, except in very special situations which must be analysed case by case. This is so because only two Matlab licenses are available in the cluster. As such, if users were allowed to submit Matlab batch jobs, a maximum of two non-parallel jobs were allowed to run in the cluster at any given time, and a maximum of 64 out of the 480 cores available in the cluster could be used at any given time for Matlab parallel batch jobs. Since Matlab is widely used in several research areas, allowing running Matlab jobs in batch mode would be a tremendous waste of the cluster's computing power. This does not means that users cannot run Matlab jobs in the cluster, but that to do so users must compile their Matlab code[23] (Matalb available on the cluster has the Matlab Coder and Matlab Compiler toolboxes) and then submit the Matlab generated executable files to the cluster to be run with the Matlab runtime engine, which is not subject to licensing constraints.

Compiling Matlab Code

Compiling Matlab code to be run as a standalone application is very simple, and can be performed both from the Linux command line[24] and the Matlab command prompt by issuing the command

mcc –m *matlab_code_file*

where *matlab_code_file* (without the .m extension) can be a script or a function, which can accept parameters. Moreover, your function may call other Matlab functions (either Matlab native functions or functions developed by you). If this is the case, the path to the directories where are located all the functions needed by your code must be in the Matlab path or the compiler will not be able to find them.

Once your Matlab code is compiled, you can submit it to the cluster as explained above by loading the Matlab Runtime module, and then execute the standalone generated file. So, your submission file could be something like (set all the parameters that you need as explained in section *Some SLURM Options for the sbatch Command*)

[23] Remember that the Matlab installed on the cluster has the Matlab Coder and the Matlab Compiler toolboxes, with which you can create libraries and standalone applications that can be run with the Matlab Runtime engine.

[24] Do not forget to load the module required to run Matlab (Programs/Matlab-R2016a).

```
#!/bin/bash

#SBATCH --time=24:00:00
#SBATCH --partition=generic
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --mem-per-cpu=4096
#SBATCH --mem_bind=local
#SBATCH --output=/home/username/Documents/job.out
#SBATCH --error=/home/ username /Documents/job.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=someone@somewhere

cd $SLURM_SUBMIT_DIR

module load Programs/matlab-R2016a-Runtime

./full_path_to_the_matlab_standalone_application [opt1 opt2 opt3 …]
```

where the *full_path_to_the_matlab_standalone_application* includes the full directory path and the name of your Matlab standalone executable, and *opt1*, *opt2*, *opt3*, … (without the square brackets),are possible input arguments required by the Matlab function that originated the standalone Matlab executable. Since the input arguments parameters are passed to the Matlab standalone application as text, your Matlab code must convert them to the appropriate type. Moreover, input arguments as arrays, strings, or any other type not consisting of isolated numbers or words must be passed enclosed in quotation marks. What was said applies only to the function that you compile.

From the above, you may wonder if you have to recode your Matlab function in order to be compiled and run as a standalone application. The answer is yes, but not too much. You can do this easily in two ways, one of which implies recurring to the Matlab function *isdeployed*, which returns one if the code is running as a standalone application, or zero otherwise, and the other may or may not use the aforementioned function. Both methods will be briefly explained.

Suppose you have a function named *SomeThing* that accepts a number and a matrix as input arguments. One way to prepare that function to be run both inside Matlab and as a standalone application is to code it as

```
function SomeThing (myNum, myArray)

    if (isdeployed)
        myNum      = str2num(myNum);
        myArray    = str2num(myArray);
    end

    …(do whatever the function was supposed to)

end
```

Now your function is ready to run both as a standalone application and natively in Matlab. One other way to accomplish the same goal without changing the original function would be by defining a second function (named *SomeThing_deployed*, for instance) which converts the input arguments to the appropriate types and then calls the original function:

```
function SomeThing_deployed (myNum, myArray)

    if (isdeployed)
        myNum      = str2num(myNum);
        myArray    = str2num(myArray);
    end

    SomeThing(myNum, myArray);

End
```

Running Parallel Matlab Jobs

For running parallel Matlab jobs you have first to develop your code for making use of the Matlab Parallel Toolbox. This can be performed on your own computer for testing proposes (as long as you have a valid Matlab license with the aforementioned toolbox), or using the Matlab available on the cluster for testing proposes.

Once your code is fully developed and tested for running in parallel, it is necessary to configure it for submission to SLURM clusters. This requires defining a configuration file for generic schedulers. If you need this file and further instructions to run Matlab parallel jobs, please contact the cluster administration through *admin@laced.isec.pt*.

COMSOL Multiphysics Jobs

Although COMSOL Multiphysics allows submitting simulations from the graphical interface to HPC clusters with SLURM, in the Minerva cluster this is not possible. So, the only way to submit a COMSOL job to the cluster is by running the job in batch mode.

For jobs not requiring parallel computation (i.e., that will use a single CPU core), you can submit the job with a batch script similar to the one shown below

```
#!/bin/bash

#SBATCH --time=24:00:00
#SBATCH --partition=generic
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --mem-per-cpu=4096
#SBATCH --mem_bind=local
#SBATCH --output=/home/username/Documents/job.out
#SBATCH --error=/home/ username /Documents/job.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=someone@somewhere

cd $SLURM_SUBMIT_DIR

module load Programs/comsol-5.2a

comsol batch  -job job_name -inputfile /full_path_to_input_file \
                                    -outputfile /full_path_to_output_file
```

where *job_name* is the name of the COMSOL batch job you want to run and is needed if your COMSOL simulation defines batch jobs, otherwise you may remove the "-job *job_name*" option. The *full_path_to_input_file* is the full path to the COMSOL file containing the simulation you want to run and *full_path_to_output_file* is the full path to the output file in which you want that COMSOL saves the results of your simulation. (The backslash at the end of the last but one line of example given above can be omitted if you give all the parameters in a single line, otherwise you must end each line but the last with a backslash.)

For running your simulation in parallel, please refer to the COMSOL Multiphysics manual. Very briefly, add cluster computing to your study, choose solvers that can take advantage of parallel computing, set the options of the batch script given above so that the appropriate number

of CPU cores are requested for your job, and replace the last line of the batch script example given above by

```
comsol -clustersimple batch  -inputfile /full_path_to_input_file \
                             -outputfile /full_path_to_output_file
```

More advanced options can be given. If you need help on using COMSOL Multiphysics for running parallel jobs, please contact the cluster administration through admin@laced.isec.pt.

Task Farming Jobs

Suppose that you have a considerable number of independent but similar jobs requiring a single CPU core, all of which take, in theory, the same time to complete. In order to materialize, let us say that you have 100 independent jobs consisting in running the same software with different input parameters, as, for instance, in Single Program Multiple Data (SPMD) problems, or parameter sweep problems. To submit those jobs you need to create 100 identical submission scripts requesting the exact same resources, loading the exact same environment modules, and running the exact same program, but differing in the parameters passed to that program, and then submit the 100 jobs to the cluster. You could automate the process of creating the 100 submission scripts, create a shell script containing one submission command per line, and then execute the shell script to submit 100 independent jobs. Nevertheless, you would end up with 100 submission scripts one script for submitting all the jobs at once, 100 jobs in the submission queue. Instead of the aforementioned procedure, you can use task farming to accomplish the same goal, with the advantage of having to create only two files (one with the submission script and the other consisting of a file containing the list of tasks), and submitting and monitoring a single job.

The Minerva cluster has a task farming program[25], which is a simple parallel program that takes a list of independent tasks from an ASCII text file, distributes the first *N* tasks through the *N* CPU cores reserved by your job, and sits waiting for tasks to complete. As soon as a running task (set of running tasks) completes execution, the task farming program takes the next task (set of tasks) and send it (them) for the CPU cores that were released. The process continues until all the tasks are executed, at which time the job is finished.

[25] Kindly provided free of charges by Miguel Afonso Oliveira, HPC Systems Support Analyst at the Imperial College London.

Taking our example of 100 independent but similar tasks given above, the submission script could be something like

```
#!/bin/bash

#SBATCH --time=48:00:00
#SBATCH --partition=generic
#SBATCH --ntasks=20
#SBATCH --output=/home/username/Documents/job.out
#SBATCH --error=/home/ username /Documents/job.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=someone@somewhere

cd $SLURM_SUBMIT_DIR

module load taskfarm

mpiexec taskfarm /full_path_to_the_taks_list_file
```

Notice that the submission script requests 20 CPU cores. As such, for running the 100 tasks of our hypothetical example, the job should request at least 5 times more computation time than the average time taken by a single task to complete. Notice also that a single file containing the standard output of the batch script and possibly a single file containing the standard error of the batch script, will be created, instead of 100 files for the standard output and possibly 100 files for the standard error, as would be the case if you have submitted 100 independent jobs instead of a single task farming job.

Assuming an SPMD model in which the program to be run is called *someProgram*, locate in "*/some_full_path/*" and accepts a single numeric value that goes from 1 to 100, the file containing the tasks list would be something like

```
./some_full_path/someProgram 1;
./some_full_path/someProgram 2;
./some_full_path/someProgram 3;
./some_full_path/someProgram 4;
    ⋮
./some_full_path/someProgram 97;
./some_full_path/someProgram 98;
./some_full_path/someProgram 99;
./some_full_path/someProgram 100;
```

Each line of the task file must end with a semicolon and can contain more than one command, as long as all the commands are separated by a semicolon and a space. To materialize, if before running the *someProgram* program you need to change to a given directory, like "/home/*username*/dir_*X*", where *username* is you username in the cluster and *X* could take values between 1 and 4, then the file containing the tasks would look like

cd /home/*username*/dir_4; ./*some_full_path*/*someProgram* 1;
cd /home/*username*/dir_2; ./*some_full_path*/*someProgram* 2;
cd /home/*username*/dir_4; ./*some_full_path*/*someProgram* 3;
cd /home/*username*/dir_1; ./*some_full_path*/*someProgram* 4;
⋮
cd /home/*username*/dir_3; ./*some_full_path*/*someProgram* 97;
cd /home/*username*/dir_2; ./*some_full_path*/*someProgram* 98;
cd /home/*username*/dir_3; ./*some_full_path*/*someProgram* 99;
cd /home/*username*/dir_4; ./*some_full_path*/*someProgram* 100;

A final notice is required. The task farming program available on the Minerva cluster requires that the number of CPU cores requested by the job be even and that the number of tasks be multiple of the number of requested cores. The task farming program will start by verify if your job complies with these two requirement and if not terminates with a message informing that the number of cores must be even or a message informing that the number of tasks is not a multiple of the number of cores.

Monitoring and Cancelling Jobs

SLURM has a set of tools that allows users to monitor and cancel their jobs in the cluster, each of which has a more or less extensive list of options that can be used. The complete list of tools and their options is out of the scope of the current manual. In what follows only the most relevant tools and their options will be given. For more detailed information of the options available for each command, please consult the manual of the command by issuing the command "man *command*", where *command* should be replaced by the name of the command.

One of the most useful tools for monitoring jobs is the *squeue* command. This command allows viewing the order in which your processes were placed in submission queue. The two main options are "-a" for viewing the complete list of jobs, and "-u *username*", where *username* is your username in the cluster, for viewing the jobs that you submitted.Figure 14shows an example of

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
37856 shortterm test.tas couceiro PD       0:00     20 (Resources)
37857 shortterm test.tas couceiro PD       0:00     20 (Priority)
37855 shortterm test.tas couceiro  R       0:09     20 CompNode[001-020]
```

Figure 14: Default output of the *squeue* command.

the output of the command "squeue –a".

By default, the output of the *squeue* command will produce a list with eight columns containing the ID of the job, the partition to which the job was submitted, the name of the batch script given as input to the *sbatch* command, the user that submitted the job to the cluster, the state in which the job is, the number of nodes that are allocated to the job, and the list of compute nodes in which the job is running and/or the reason for which the job is held in the submission queue. Concerning the job state, this can be one of: PD (pending), R (running), CA (cancelled), CF (configuring), CG (completing), CD (completed), F (failed), TO (timeout), NF (node failure) and SE (special exit state).

Another useful command for monitoring jobs is the *sacct* command the output of which is depicted in Figure 15. By default, the acct command displays information for jobs submitted by the user that issues the command. For showing information for all jobs you can use de the "-a" option and for viewing information of a particular job you use de "–j *jobID*" option, in which *jobID* should be replaced by the ID of the job.

```
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
37860        test.task+  shortterm     laced        480    RUNNING      0:0
37860.0      hydra_pmi+                laced         20    RUNNING      0:0
37861        test.task+  shortterm     laced        480    PENDING      0:0
37862        test.task+  shortterm     laced        480    PENDING      0:0
```

Figure 15: Default output of the *sacct* command.

For cancelling jobs, you use the *scancel* command. For cancelling a single job with a given ID, use the command "scancel –j *jobID*", where *jobID* should be replaced by the ID of the job. For cancelling all of your jobs, use the command "scancel –u *username*", where *username* should be replaced by your username in the cluster. Notice that users can only cancel the jobs that they have submitted.

Job Scheduling

Jobs submitted to the cluster are ordered in the waiting queue according to a priority of execution such that jobs with higher priority values are placed in higher positions of the queue and will start executing sooner. In the Minerva cluster, the priority of a job is constantly computed by the SLURM multifactor plugin (see the documentation for details)as a weighted sum of the following parameters, each of which has a priority factor comprised between zero and 1:

- Partition: The priority factor, with a weight of 1,000 in the final priority of the job, is equal to the partition priority (see TABLE 2 on page 34) divided by the higher priority of all the partitions. So, the partition priority of a given job is constant throughout the time the job is waiting in the queue.
- Fair-Share: The fair-share factor of a user/account, with a weight of 10,000 in the final priority of the job, is based on the difference between the resources granted to that user/account and the resources consumed at any given time by that user/account. So, the fair-share priority of a given job that is waiting in the queue may vary over time if the user/account that submitted the job has other jobs running in the cluster.
- Job Size: The job size priority factor, with a weight of 1,000 in the final priority of the job, correlates to the number of nodes or CPU cores requested for the job, such that a job requesting all the cores of the cluster (480) will have a job size factor of 1. So, the job size priority of a given job is constant throughout the time the job is waiting in the queue.
- Job Age: The job age priority factor, with a weight of 1,000 in the final priority of the job, is computed as the ratio between the time that the job has been waiting in the queue and a time limiting factor, which is equal to seven days. So, job age priority of a given job that is waiting in the queue increases over time, favouring then jobs that are waiting in the queue for longer times.

From the above, the final priority of a given job in the Minerva cluster is a 32-bit integer computed by:

$$\begin{aligned} Job\ Priority = {} & Partition\_Weight \times Partition\_Factor + \\ & FairShare\_Weight \times FairShare\_Factor + \\ & JobSize\_Weight \times JobSize\_Factor + \\ & JobAge\_Weight \times JbAge\_Factor \end{aligned}$$

Since SLURM constantly computes the priority of all jobs that are waiting in the queue, these can be requeued at any time.

Besides the above mentioned mechanism to compute job priority, the Minerva cluster also uses the SLURM backfill plugin. With this plugin, jobs having a lower priority can be sent to execution if by doing so the expected time for sending higher priority jobs for execution is not delayed. The expected time for sending a job to execution is computed taking into account the time requested for computation of jobs in pending state, the remaining time allowed for execution of jobs that are already running in the cluster, and the resources that jobs that are running will set free at the end of the time requested for computation. So, to take advantage of the backfill mechanism, users should specify with some accuracy the time requested for their jobs to run. However, do not request time to accurately, because if your job will be killed without notice if it has not finished by the end of the requested time.

To see the priority of a given job, you can use the command “sprio –j *jobID*”, where *jobID* is the ID of the job. This will work only for jobs that are in the pending state and will show the job priority along with the priorities of the abovementioned individual factors used for computing the job priority.

To see your fair-share at any given time use the command “sshare –u *username*”, where *username* should be replaced by your username in the cluster. This will also show you the fair-share of the accounts to which you are associated.