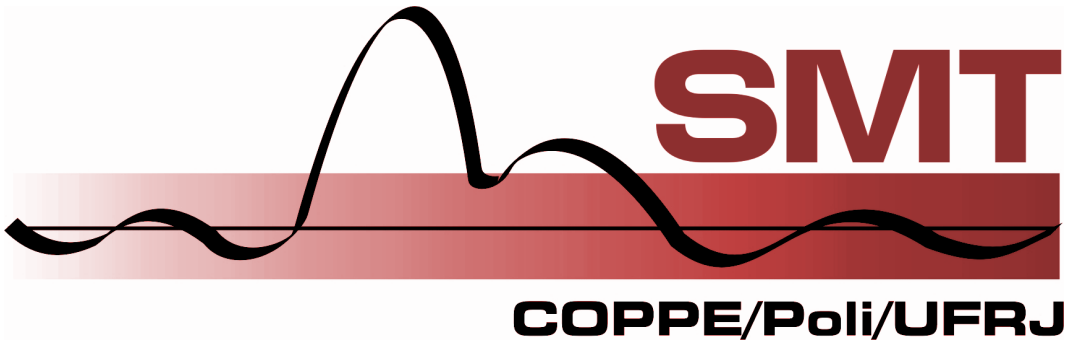

Laboratório de Sinais, Multimídia e Telecomunicações

Grupo de Processamento de Sinais

Estações de Trabalho e Cluster do Laboratório SMT

Documentação ETCL-SMT-R01

Abril/2016

Documento Versão 1

CLuster SMT

AUTHOR(S)

José F. L. de Oliveira

jose.oliveira@smt.ufrj.br

jleitex@gmail.com

Version	Anotation	Date	Author(s)
1	Primeira versão do documento em Português	29 Mar, 2016	José
2			
3			
4			

Capítulo 2

Recursos do Laboratório

2.1 Estações de Trabalho

Há um total de 13 estações de trabalho em operação no Laboratório **SMT**, com as configurações mostradas na Tabela 2.1. São máquinas de uso geral e empregadas para executar programas com interfaces gráficas.

Tabela 2.1: Estações de trabalho do Laboratório **SMT**.

IP	Nome	Nome Completo	Processador	Clock	RAM
10.221.70.3	leiria	leiria.smt.ufrj.br	Intel® Core™ i3-2100	3.10 GHz	8 GB
10.221.70.4	cordoba	cordoba.smt.ufrj.br	Intel® Core™2 Duo E7500	2.93 GHz	4 GB
10.221.70.5	roma	roma.smt.ufrj.br	Intel® Pentium® Dual E2160	1.80 GHz	4 GB
10.221.70.6	oslo	oslo.smt.ufrj.br	Intel® Core™2 Duo E7500	2.93 GHz	4 GB
10.221.70.7	taiwan	taiwan.smt.ufrj.br	Intel® Core™ i7-2700K	3.50 GHz	8 GB
10.221.70.8	moscou	moscou.smt.ufrj.br	Intel® Core™ i7-4790K	4.00 GHz	32 GB
10.221.70.9	tampere	tampere.smt.ufrj.br	não instalada	0.00 GHz	0 GB
10.221.70.10	trodheim	trodheim.smt.ufrj.br	não instalada	0.00 GHz	0 GB
10.221.70.11	rio	rio.smt.ufrj.br	Intel® Core™ i7-4790K	4.00 GHz	32 GB
10.221.70.12	ouropreto	ouropreto.smt.ufrj.br	Intel® Core™2 6320	1.86 GHz	4 GB
10.221.70.13	zermatt	zermatt.smt.ufrj.br	Intel® Core™ i7-4790K	4.00 GHz	32 GB
10.221.70.14	saopaulo	saopaulo.smt.ufrj.br	Intel® Core™ i7-4790K	4.00 GHz	32 GB
10.221.70.15	helsinque	helsinque.smt.ufrj.br	Intel® Core™2 Quad Q6600	2.40 GHz	4 GB
10.221.70.16	munique	munique.smt.ufrj.br	Intel® Core™ i7 X 990	3.47 GHz	32 GB
10.221.70.17	geneve	geneve.smt.ufrj.br	Intel® Core™ i7-4790K	4.00 GHz	32 GB

2.2 Estrutura do Cluster

O **Cluster-SMT** é composto por seis máquinas como mostrado na Tabela 2.2. A máquina **head** faz a distribuição de processos quando se usa o comando qsub, abordado na Seção 3.2.2, *Submissão Avançada de Processos*. As máquinas **node** são usadas para rodar os processos. O **Cluster-SMT** foi configurado para operar em dois modos:

- **Single-node**
  cada nó do cluster pode ser usado como qualquer outra máquina do **SMT**. Basta fazer o *login* em um dos nós e rodar os processos que desejar em *background*;
- **Multi-node**
  neste modo é necessário que o usuários façam o login no nó **head** e usem o comando **qsub** para rodar seus processos. O comando **qsub** escolhe os nós onde os processos dos usuários serão executados[1] e, no caso de o *cluster* estar totalmente ocupado, põe os processos numa fila de espera (ver Seção 2.2.2). Além disto, em caso de uma eventutal reiniciação dos nós os processos serão automaticamente reiniciados

O nó **head** não deve ser usado para rodar processos no modo **Single-node** para evitar sua superutilização, visto que é o nó que gerencia o cluster e atua como servidor do serviço de monitoramento de processos, o ganglia (ver Seção 2.2.1).

Tabela 2.2: Máquinas que compõem o **Cluster-SMT**.

IP	Nome	Nome Completo	Processador	Clock	RAM
10.221.90.2	head-01-01	head-01-01.smt.ufrj.br	Intel® Xeon® E5-2403	1.80 GHz	96 GB
10.221.90.3	node-01-01	node-01-01.smt.ufrj.br	Intel® Xeon® E5-2403	1.80 GHz	96 GB
10.221.90.4	node-01-02	node-01-02.smt.ufrj.br	Intel® Xeon® E5-2403	1.80 GHz	96 GB
10.221.90.5	node-01-03	node-01-03.smt.ufrj.br	Intel® Xeon® E5-2403	1.80 GHz	96 GB
10.221.90.6	node-01-04	node-01-04.smt.ufrj.br	Intel® Xeon® E5-2403	1.80 GHz	96 GB
10.221.90.7	node-01-05	node-01-05.smt.ufrj.br	Intel® Xeon® E5-2403	1.80 GHz	96 GB

2.2.1 Página de Monitoramento do Cluster

A página de monitoramento do cluser pode ser acessada através do endereço

`http://head-01-01.smt.ufrj.br/ganglia`

[1] Esta escolha pode ser configurada por opções fornecidas pelo usuário.

a qual mostra várias métricas de desempenho do cluster tais como:

- uso dos processadores;
- uso de memória;
- nós ativos e inativos.

Ver Figura 2.1 na página 4. As métricas estão disponíveis para o cluster como um todo ou por nó.

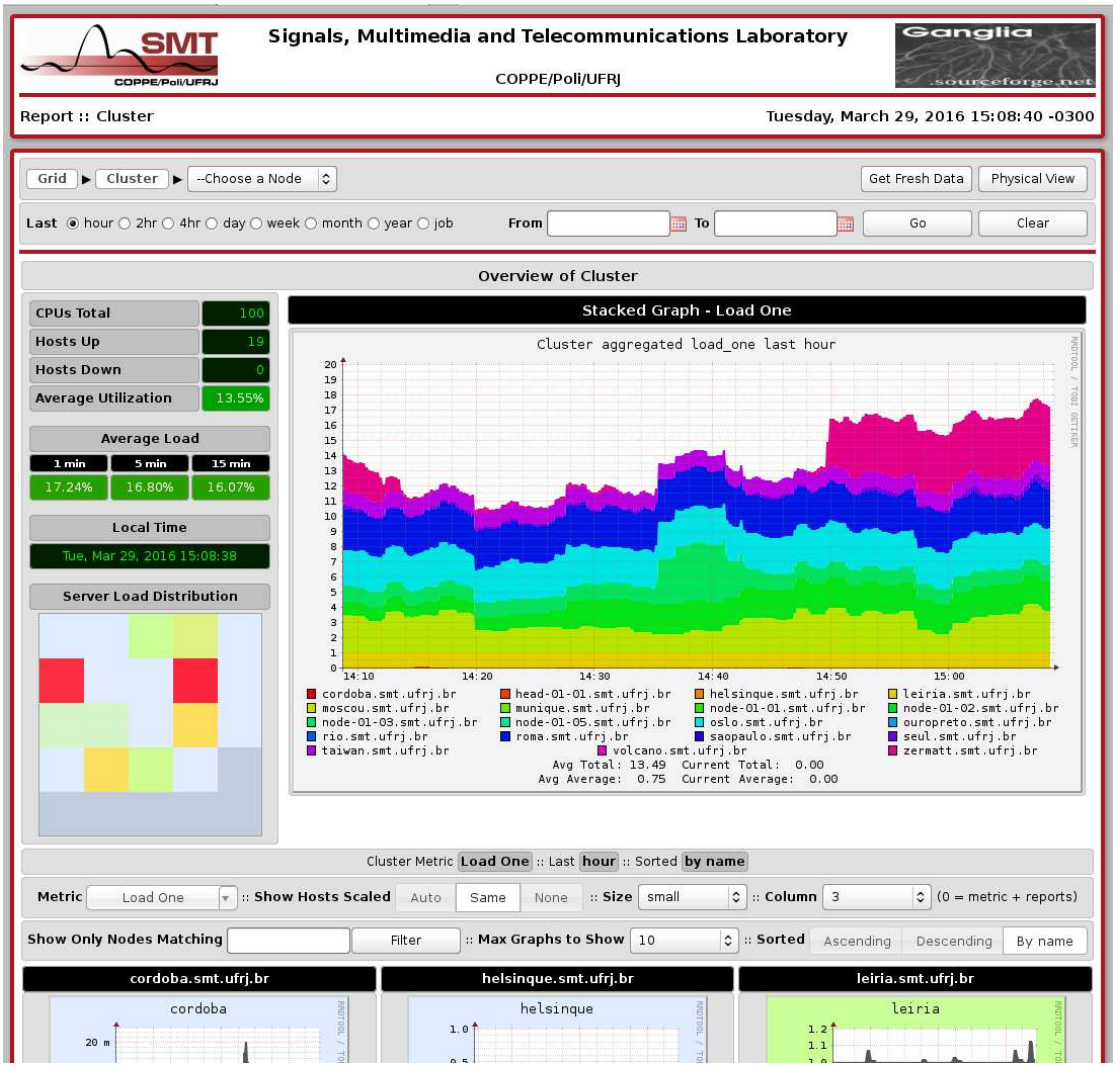

Figura 2.1: Ganglia. Página inicial de monitoramento do **Cluster-SMT**.

2.2.2 Gerenciador de Fila

Para processos executados através do **qsub**, o **Cluster-SMT** emprega o gerenciador de filas do **PBS-Torque**. Este gerenciador seleciona um nó de computação livre para ser

usado para cada processo submetido (*job*) por um usuário. Se todos os processadores de todos os nós estiverem sendo totalmente utilizados, um novo processo submetido através do **qsub** será posto na fila e aguardará até que um ou mais processadores do cluster sejam liberados.

Capítulo 3

Comandos do Usuário

Visto que o sistema operational instalado no **Cluster-SMT** é a distribuição Linux **Fedora 20**, será útil a todos os usuários do cluster se familiarizar com os comandos deste sistema operacional.

3.1 Fazendo Login no Cluster

A forma de fazer *login* no **Cluster-SMT** depende de como o usuário peferir utilizá-lo. Se for para utilizá-lo no modo **singe-node**, o usuário poderá fazer *login* nos nós como faria em qualquer máquina do laboratório SMT, evitando executar processos nó **head** pelos motivos apresentados na Seção 2.2, e submeter seus processos normalmente como explicado na Seção 3.2.1. Se for para utilizá-lo no modo **multi-node** o usuário deverá fazer *login* no nó **head** e submeter seus processos através do comando **qsub** como explicado na Seção 3.2.2.

3.1.1 Fazendo Login pelo Linux

Para ter acesso ao **Cluster-SMT** usando um sistema operacional baseado no Linux, os usuários podem simplismente utilizar o comando **`ssh`** em uma das seguintes formas:

```
$ ssh <user>@head-01-01.smt.ufrj.br
$ ssh <user>@head-01-01
$ ssh <user>@10.221.90.2
```

Apesar de ser possível abrir interfaces gráficas remotamente[1] adicionando o parâmetro **`-X`** ao comando **`ssh`**, é recomendado não fazer uso indiscriminado desta carac-

[1] Tal como o MATLAB® Desktop.

terística. Isto poderá consumir banda considerável da rede e reduzir o desempenho do próprio cluster. Utilize somente para fazer testes e/ou quando for realmente necessário. Se os usuários tiverem realmente que fazer uso contínuo destas interfaces gráficas, devem se logar diretamente em uma das estações de trabalho do Laboratório **SMT** II, listadas na Tabela 2.1, as quais estão aptas a rodar este tipo de aplicações muito mais rápido por disporem em geral de placas vídeo de alto desempenho.

3.1.2 Fazendo Login pelo Windows

Para ter acesso ao **Cluster-SMT** usando um sistema operacional Windows®, os usuários devem instalar um programa tal como **winscp** que pode ser encontrado no sítio **http://winscp.net**. Este programa permite copiar arquivos para o cluster e máquinas do **SMT**. Também é possível copiar arquivos das máquinas Linux para máquinas Windows®.

3.1.3 Copiando Arquivos no Linux

Para copiar arquivos entre máquinas Linux, pode-se utilizar o comando **scp** como

```
$ scp <files> <user>@head-01-01.smt.ufrj.br:targetfolder
```

Deve ser observado que, usando esta forma do comando, o diretório **targetfolder** deve estar localizado no diretório *home* (**/home/<user>**) do usuário, ou seja, o comando acima é equivalente a

```
$ scp <files> <user>@head-01-01.smt.ufrj.br:/home/<user>/targetfolder
```

Se o usuário preferir ou se sentir mais confortável usando um progrma com uma interface gráfica, há uma grande variedade delas disponíveis para sistemas Linux. Procurar por **Linux client sftp graphical** no Google listará várias delas. Particularmente, para usuários do **KDE** ou para aqueles que têm o **konqueror** ou o **dolphin** instalados, é possível acessar arquivos entre máquinas Linux utilizado o protocolo **fish**. Ou seja, entrando o texto

```
fish://<user>@head-01-01.smt.ufrj.br/home/<user>
```

na barra de localização, por exemplo, mostrará o conteúdo do diretório *home* do usuário na máquina especificada. Copiar arquivos pode então ser feito através de um simples *drag and drop*.

3.2 Submissão de Processos

Como já apresentado na Seção 2.2, o usuário do **Cluster-SMT** poder executar processos de duas formas, uma utilizando o modo **single-node** e a outra utilizado a forma **multi-node**. A primeira forma é o que se vai chamar de submissão simples de processos e a segunda forma será denominada submissão avançada de processos. Ambas serão abordadas nas subseções a seguir.

3.2.1 Submissão Simples de Processos

A submissão simples de processos pode ser utilizada em qualquer máquina do Laboratório **SMT**, incluindo o cluster, desde que a máquina atenda aos requisitos do processo (ou processos) do usuário tais como memória disponível. O usuário pode rodar seus processos como normalmente faz, lembrando que os processos submetidos não utilizam interface gráfica, mas apenas processam dados de entrada e produzem dados de saída como, por exemplo,

```
$ nohup nice -n 19 ./my_program input.dat output.dat > /dev/null &
```

O nome do programa do usuário deve ser precedido pelos comandos

1. **nohup**
   para evitar que ao fazer *logout* ou fechar um terminal o processo do usuário seja encerrado;

2. **nice -n 19**
   para evitar que os processos sendo executados em *background* em uma estação de trabalho do **SMT** inviabilize o uso da interface gráfica por outro usuário que esteja fisicamente utilizado a máquina. No caso do cluster, permite que maior prioridade seja dada aos processos submetidos com o comando **qsub** (Ver Seção 3.2.2).

É preciso também que o **&** seja adicionado ao final da linha de comando para que o precesso seja executado em *background*. O redirecionamento feito por **>** para **`/dev/null`** tem a finalidade de descartar as mensagens de *standard output* e *standard error*, ao invés de salvá-las no arquivo **`nohup.out`**. Isto evita que um arquivo **`nohup.out`** gigantesco de texto possa ser criado em algumas situações. Se o usuário desejar salvar esta saída basta remover o redirecionamento **>** para **`/dev/null`**. O usuário também pode especificar outro nome de arquivo para salvar esta saída tal como **`my_program.log`** no lugar de **`/dev/null`**.

3.2.2 Submissão de Avançada Processos

A submissão avançada de processos só pode ser feita através do nó **head** (head-01-01) do cluster, ao contrário da submissão simples abordada anteriormente. Como apresentado brevemente na Seção 2.2.2, o **Cluster-SMT** foi configurado para utilizar o gerenciador de filas de processos PBS-Torque. Isto permite que o comando **qsub** possa, a partir do nó **head**, distribuir processos pelos nós de computação (node-01-01 ao node-01-05), checar a disponibilidade dos nós e reiniciar processos interrompidos por uma eventual paralização dos nós. O comando **qsub** submete um *script* executável ao servidor de batelada (*batch server*). Portando, a forma mais simples de rodar um programa já existente através do **qsub** é criar um *script* **my_program_script** como o da Listagem 3.1 e, a partir do nó **head**, utilizar a seguinte linha de comando

```
$ qsub my_program_script
```

Note que não foram, e nem devem ser, utilizados os comandos **nohup** e **nice** ou o terminador **&** para por o processo em *background*. O **qsub** e o gerenciador de filas de processos cuidam disto para o usuário.

Listagem 3.1: *Script* simples para submissão de processos através do **qsub**.

```
1  #!/bin/bash
2
3  #*********************************************************************************
4  #
5  # my_program_script
6  #
7  #*********************************************************************************
8  #
9  # Author.....: Jose F. L. de Oliveira
10 #
11 # Started....: Mon Apr 11 2016
12 # Modified...: Mon Apr 11 2016
13 #
14 # Emails.....: jose.oliveira@smt.ufrj.br
15 #
16 # Address....: Universidade Federal do Rio de Janeiro
17 #              Caixa Postal 68.504, CEP: 21.945-970
18 #              Rio de Janeiro, RJ - Brasil.
19 #
20 #*********************************************************************************
21 #
22 # Simple qsub submission script
23 #
24 #*********************************************************************************
25
26 ./my_program input.dat output.dat
```

Entretando, é necessário que o programa **my_program** e os dados que ele processa estejam disponíveis nos nós de computação que serão selecionados pelo **qsub** no nó **head**. Em geral isto ocorre, visto que as contas dos usuários e seus respectivos diretórios *home* estão montados automaticamente em todas as máquinas do SMT.

Porém, o **Cluster-SMT** dispõe de discos de armazenamento local para processos que precisem ler e/ou gravar grandes quantidades de dados. Estes discos estão montados em **`/workspace`**. O propósito deste armazenamento local é evitar que o dados processados e gerados carreguem desnecessariamente o tráfego de rede do **SMT**. Como o armazenamento é local, é presiso que

1. os dados a serem processados sejam copiados para diretório de trabalho dos nós de processamento selecionados pelo **qsub**, nós estes que o usuário não sabe quais serão apriori;
2. o programa seja executado;
3. os dados resultantes do processamento sejam copiados de volta para o nó **head** dentro do diretório de trabalho;
4. remover os dados já processados dos nós de computação;
5. copiar os resultados para a conta do usuário.

O *script* desenvolvido para executar as etapas descritas acima está localizado na pasta **`/usr/share/cluster`** no nó **head**. Pode ser copiado para qualquer lugar usando comandos como

```
$ ssh <user>@head-01-01
$ mkdir -p /workspace/<user>
$ cd /usr/share/cluster
$ cp qsub-submission-script-example /workspace/<user>
$ cd /workspace/<user>
```

Dentro da pasta **`/workspace/<user>`**, devem estar o programa a ser executado[2] e todos os outros arquivos necessários à sua correta execução, tais como arquivos de dados a serem processados.

Isto posto, o arquivo **`qsub-submission-script-example`** deve ser editado pelo usuário para acomodar as tarefas que precisam ser executadas. As funções a serem editadas pelo usuário são[3]:

[2] No caso de programas desenvolvidos pelo usuário. Programas que estão no caminho do sistema, como o MATLAB®, não precisam obedecer a este requerimento.

[3] Pode ser necessário editar ou criar outras funções para uma dada tarefa.

- **stageIn**

  copia arquivos do nó **head** para o nó de computação selecionado. Ver Listagem 3.2;

- **runApp**

  executa os processos do usuário. Esta função precisa ser editada para executar os processos do usuário. Ver Listagem 3.3;

- **stageOut**

  Copia arquivos do nó de computação selecioando para o nó **head**. Ver Listagem 3.4.

A função **main**, ver Listagem 3.5 chama todas as funções acima de modo a executar os processos do usuário corretamente. Os usuários devem ler as instruções em **qsub-submission-script-example**, cuja versão completa é mostrada na Listagem 3.6. Após este *script* ser editado adequadamente, o comando

```
$ qsub my-qsub-submission-script
```

deve ser executado dentro da pasta **/workspace/<user>**.

3.2.3 Checando o Estado da Fila de Processos

If a user wants to check the cluster load, it can employ the command **pbstop** on the server. There is also the command **qstat**.

In order to remove a process (job) from the queue, users must use the command **qdel**:

```
qdel JOBID
```

on the server, where `JOBID` is the PID (*Process ID*) of the user job.

Listagem 3.2: QSub submission script example. Function **stageIn**.

```
1  #*******************************************************************************
2  # FUNCTION stageIn
3  #*******************************************************************************
4  #
5  # Transfer files from server to local disk.
6  #
7  #*******************************************************************************
8
9  function stageIn
10 {
11   echo "-------------------------------------------------------------------"
12   echo "Transferring files from server to compute node"
13   echo "Writing files in node directory"
14   echo "-------------------------------------------------------------------"
15   echo
```

```
16
17    echo "------------------------------------------------------------------------"
18    echo "Copying Files"
19    echo
20    echo "From: $HEAD:$HEAD_WORK_DIR"
21    echo "To  : $NODE_WORK_DIR"
22    echo "------------------------------------------------------------------------"
23    echo
24
25    mkdir $NODE_WORK_DIR
26
27    cd $NODE_WORK_DIR
28
29    rsync -auv $HEAD:$HEAD_WORK_DIR/ ./
30
31    echo "------------------------------------------------------------------------"
32    echo "Files in node working directory are as follows"
33    echo "------------------------------------------------------------------------"
34    echo
35
36    ls -l
37
38    echo "------------------------------------------------------------------------"
39    echo
40  } # End of stageIn.
```

Listagem 3.3: QSub submission script example. Function **runApp**.

```
1  #*******************************************************************************
2  # FUNCTION runApp
3  #*******************************************************************************
4  #
5  # Execute the application.
6  #
7  # Do not run in the background, that is, do not use & at the end of the
8  # command line.
9  #
10 #*******************************************************************************
11
12 function runApp
13 {
14   cd $NODE_WORK_DIR
15
16   # Example of command line.  No '&' at the end of the command line.
17   ./my_program < my_input_file.dat > my_output_file.dat
18 } # End of runApp.
```

Listagem 3.4: QSub submission script example. Function **stageOut**.

```
1  #*******************************************************************************
2  # FUNCTION stageOut
3  #*******************************************************************************
4  #
5  # Copy the necessary files back to permanent directory.
6  #
7  #*******************************************************************************
8
9  function stageOut
10 {
11   echo "------------------------------------------------------------------------"
12   echo "Transferring files from compute nodes to server"
13   echo "Writing files in permanent directory"
14   echo "------------------------------------------------------------------------"
15   echo
16
17   echo "------------------------------------------------------------------------"
18   echo "Copying Files"
19   echo
20   echo "From: $NODE_WORK_DIR"
21   echo "To  : $HEAD:$HEAD_WORK_DIR"
```

```
22    echo "------------------------------------------------------------------------"
23    echo
24
25    cd $NODE_WORK_DIR
26
27    rsync -auv ./ $HEAD:$HEAD_WORK_DIR/
28
29    echo "------------------------------------------------------------------------"
30    echo "Final files in permanent data directory"
31    echo "------------------------------------------------------------------------"
32    echo
33
34    $SSH $HEAD "cd $HEAD_WORK_DIR; ls -l"
35
36    echo "------------------------------------------------------------------------"
37    echo
38
39    # ---------------------------------------------------------------------------
40    # Remove node working directory so as to not fill the node home directory.
41    # This may not be a good idea since intermidiate results can be lost.
42    # Comment the following commands if you do not want this step.
43    # ---------------------------------------------------------------------------
44
45    echo "------------------------------------------------------------------------"
46    echo "Removing compute node working directory"
47    echo "------------------------------------------------------------------------"
48    echo
49
50    cd
51    rm -rf $NODE_WORK_DIR
52
53    echo "------------------------------------------------------------------------"
54    echo
55  } # End of stageOut.
```

Listagem 3.5: QSub submission script example. Function **main**.

```
1   #*******************************************************************************
2   # FUNCTION main
3   #*******************************************************************************
4   #
5   # Staging in, running the job, and staging out were specified above as
6   # functions. Now call these functions to perform the actual file transfers and
7   # program execution.
8   #
9   #*******************************************************************************
10
11  function main
12  {
13    local SIG_INT=2
14    local SIG_KILL=9
15    local SIG_TERM=15
16
17
18    trap 'early; stageOut' $SIG_INT $SIG_KILL $SIG_TERM
19
20    prepare
21
22    stageIn
23    runApp
24    stageOut
25
26    exit
27  } # End of main.
```

Listagem 3.6: The full qsub submission script example.

```
1   #!/bin/bash -f
2
3   #*******************************************************************************
```

```
#
# qsub-submission-script-example
#
#*******************************************************************************
#
# Author.....: qcd.phys.cmu.edu
#              Trace.IT Consultoria
#              José F. L. de Oliveira
#
# Started....: Wed Jul 16 2014
# Modified...: Tue Aug 05 2014
#
# Emails.....: jose.oliveira@smt.ufrj.br
#              jleitex@gmail.com
#
# Address....: Universidade Federal do Rio de Janeiro
#              Caixa Postal 68.504, CEP: 21.945-970
#              Rio de Janeiro, RJ - Brasil.
#
#*******************************************************************************
#
# Script to send processes to PBS-Torque. Adapted from:
#
#   http://qcd.phys.cmu.edu/QCDcluster/pbs/run_serial.html
#
# This script copies the folder containing programs and data files on the
# server to the compute node, selected by qsub, to execute the job. At the end
# of the job execution the script copies all the files back to the original
# server folder.
#
#*******************************************************************************
#
# Usage:
#
# 1. Prepare a directory in your home containing the program to be executed and
#    all input data files needed to its execution. For example:
#
#      $HOME/mysimulation-1
#
#    which contains
#
#      my_program
#      my_input_file.dat
#
# 2. Copy this script to $HOME/mysimulation-1
#
# 3. Edit the function runApp ins this script, inserting the command line that
#    executes your program. The output data (your results) can be written in
#    the folder $HOME/mysimulation-1 or in another folder inside this folder.
#    For example (inside runApp):
#
#      mkdir -p results/temporal-series
#      mkdir -p results/fourier-transform
#
#    Since your progam will save the results inside these folders that are
#    created inside mysimulation-1 in the compute node, when the job finishes
#    they will be copied back to $HOME/mysimulation-1 in the server node.
#
# 4. Enter the directory $HOME/mysimulation-1 and run your edited script. For
#    example:
#
#      qsub my-qsub-submission-script
#
#*******************************************************************************


#*******************************************************************************
# GLOBAL VARIABLES
#*******************************************************************************

SCP="/usr/bin/scp"
SSH="/usr/bin/ssh"

```

```
77  HEAD=""
78
79  NODE_WORK_DIR=""
80  HEAD_WORK_DIR=""
81
82
83  #*******************************************************************************
84  # FUNCTION prepare
85  #*******************************************************************************
86
87  function prepare
88  {
89    local JOB_NODE=""
90
91
92    # -----------------------------------------------------------------------
93    # The PBS directives. Directives must be placed before program execution.
94    # -----------------------------------------------------------------------
95
96    # -----------------------------------------------------------------------
97    # Specify a job name. Maximum of 15 characters, no spaces, and starting
98    # with alphanumeric. It is used to identify the job in the process queue.
99    # -----------------------------------------------------------------------
100   #PBS -N Job
101
102   # -----------------------------------------------------------------------
103   # By default, the standard output and error streams are sent to files in the
104   # current working directory with names:
105   #
106   #   job_name.osequence_number  <-  output stream
107   #   job_name.esequence_number  <-  error stream
108   #
109   # where job_name is the name of the job and sequence_number is the job number
110   # assigned when the job is submitted. Use the directives below to change the
111   # files to which the standard output and error streams are sent.
112   # -----------------------------------------------------------------------
113   #PBS -o stdout_file
114   #PBS -e stderr_file
115
116   # -----------------------------------------------------------------------
117   # Output some useful job information.
118   # -----------------------------------------------------------------------
119
120   JOB_NODE=$(cat $PBS_NODEFILE)
121
122   echo "------------------------------------------------------------------------"
123   echo "Job is running on node: $JOB_NODE"
124   echo "------------------------------------------------------------------------"
125   echo
126
127   echo "------------------------------------------------------------------------"
128   echo
129   echo "PBS: qsub is running on.......: $PBS_O_HOST"
130   echo "PBS: originating queue is.....: $PBS_O_QUEUE"
131   echo "PBS: executing queue is.......: $PBS_QUEUE"
132   echo "PBS: working directory is.....: $PBS_O_WORKDIR"
133   echo "PBS: execution mode is........: $PBS_ENVIRONMENT"
134   echo "PBS: job identifier is........: $PBS_JOBID"
135   echo "PBS: job name is..............: $PBS_JOBNAME"
136   echo "PBS: node file is.............: $PBS_NODEFILE"
137   echo "PBS: current home directory is: $PBS_O_HOME"
138   echo "PBS: path.....................: $PBS_O_PATH"
139   echo
140   echo "------------------------------------------------------------------------"
141   echo
142
143   # -----------------------------------------------------------------------
144   # The server host.
145   # -----------------------------------------------------------------------
146
147   HEAD="$PBS_O_HOST"
148
149   # -----------------------------------------------------------------------
```

```
150   # The prologue script automatically makes a directory on the local disks
151   # for you. The name of this directory depends on the job id, but you need
152   # only refer to it using $NODE_WORK_DIR.
153   # ------------------------------------------------------------------------
154
155   NODE_WORK_DIR="$HOME/PBS_$PBS_JOBID"
156
157   # ------------------------------------------------------------------------
158   # Specify the permanent directory(ies) on the server host. Note that when
159   # the job begins execution, the current working directory at the time the
160   # qsub command was issued becomes the current working directory of the job.
161   # ------------------------------------------------------------------------
162
163   HEAD_WORK_DIR="$PBS_O_WORKDIR"
164
165   echo "------------------------------------------------------------------------"
166   echo
167   echo "Head.........................: $HEAD"
168   echo
169   echo "Working Directory on Node.....: $NODE_WORK_DIR"
170   echo "Working Directory on Head.....: $HEAD_WORK_DIR"
171   echo
172   echo "------------------------------------------------------------------------"
173   echo
174
175   JOB_NODE=$(cat $PBS_NODEFILE)
176
177   echo "------------------------------------------------------------------------"
178   echo "Job is running on node: $JOB_NODE"
179   echo "------------------------------------------------------------------------"
180   echo
181 } # End of prepare.
182
183
184 #*******************************************************************************
185 # FUNCTION stageIn
186 #*******************************************************************************
187 #
188 # Transfer files from server to local disk.
189 #
190 #*******************************************************************************
191
192 function stageIn
193 {
194   echo "------------------------------------------------------------------------"
195   echo "Transferring files from server to compute node"
196   echo "Writing files in node directory"
197   echo "------------------------------------------------------------------------"
198   echo
199
200   echo "------------------------------------------------------------------------"
201   echo "Copying Files"
202   echo
203   echo "From: $HEAD:$HEAD_WORK_DIR"
204   echo "To  : $NODE_WORK_DIR"
205   echo "------------------------------------------------------------------------"
206   echo
207
208   mkdir $NODE_WORK_DIR
209
210   cd $NODE_WORK_DIR
211
212   rsync -auv $HEAD:$HEAD_WORK_DIR/ ./
213
214   echo "------------------------------------------------------------------------"
215   echo "Files in node working directory are as follows"
216   echo "------------------------------------------------------------------------"
217   echo
218
219   ls -l
220
221   echo "------------------------------------------------------------------------"
222   echo
```

```
223 } # End of stageIn.
224
225
226 #******************************************************************************
227 # FUNCTION runApp
228 #******************************************************************************
229 #
230 # Execute the application.
231 #
232 # Do not run in the background, that is, do not use & at the end of the
233 # command line.
234 #
235 #******************************************************************************
236
237 function runApp
238 {
239   cd $NODE_WORK_DIR
240
241   # Example of command line.  No '&' at the end of the command line.
242   ./my_program < my_input_file.dat > my_output_file.dat
243 } # End of runApp.
244
245
246 #******************************************************************************
247 # FUNCTION stageOut
248 #******************************************************************************
249 #
250 # Copy the necessary files back to permanent directory.
251 #
252 #******************************************************************************
253
254 function stageOut
255 {
256   echo "------------------------------------------------------------------------"
257   echo "Transferring files from compute nodes to server"
258   echo "Writing files in permanent directory"
259   echo "------------------------------------------------------------------------"
260   echo
261
262   echo "------------------------------------------------------------------------"
263   echo "Copying Files"
264   echo
265   echo "From: $NODE_WORK_DIR"
266   echo "To  : $HEAD:$HEAD_WORK_DIR"
267   echo "------------------------------------------------------------------------"
268   echo
269
270   cd $NODE_WORK_DIR
271
272   rsync -auv ./ $HEAD:$HEAD_WORK_DIR/
273
274   echo "------------------------------------------------------------------------"
275   echo "Final files in permanent data directory"
276   echo "------------------------------------------------------------------------"
277   echo
278
279   $SSH $HEAD "cd $HEAD_WORK_DIR; ls -l"
280
281   echo "------------------------------------------------------------------------"
282   echo
283
284   # ------------------------------------------------------------------------
285   # Remove node working directory so as to not fill the node home directory.
286   # This may not be a good idea since intermidiate results can be lost.
287   # Comment the following commands if you do not want this step.
288   # ------------------------------------------------------------------------
289
290   echo "------------------------------------------------------------------------"
291   echo "Removing compute node working directory"
292   echo "------------------------------------------------------------------------"
293   echo
294
295   cd
```

```
296   rm -rf $NODE_WORK_DIR
297
298   echo "--------------------------------------------------------------------------"
299   echo
300 } # End of stageOut.
301
302
303 #*******************************************************************************
304 # FUNCTION early
305 #*******************************************************************************
306
307 function early
308 {
309   echo "=========================================================================="
310   echo "WARNING: EARLY TERMINATION"
311   echo "=========================================================================="
312   echo
313 } # End of early
314
315
316 #*******************************************************************************
317 # FUNCTION main
318 #*******************************************************************************
319 #
320 # Staging in, running the job, and staging out were specified above as
321 # functions. Now call these functions to perform the actual file transfers and
322 # program execution.
323 #
324 #*******************************************************************************
325
326 function main
327 {
328   local SIG_INT=2
329   local SIG_KILL=9
330   local SIG_TERM=15
331
332
333   trap 'early; stageOut' $SIG_INT $SIG_KILL $SIG_TERM
334
335   prepare
336
337   stageIn
338   runApp
339   stageOut
340
341   exit
342 } # End of main.
343
344
345 #*******************************************************************************
346 # CALL main
347 #*******************************************************************************
348
349 main
```