# SLURM-BULL-veredas Documentation

*Versão 0.0.1*

**Domingos Rodrigues**

04/05/2011

# Conteúdo

**Nota:** Prezados usuários:

Tal como já foi anunciado, a licença do escalonador de filas PBSPro expirou no dia 20 de abril de 2011 e por esta razão foi realizada a migração para um outro gerenciador com licença livre, o SLURM. Este guia prático encontra-se em construção e por isso é bem possível que contenha algumas incorreções. Contamos então com algum retorno dos usuários no sentido de melhorar a apresentação e o conteúdo deste documento de modo a facilitar a compreensão e utilização do SLURM.

- *Guia básico para o SLURM do cluster BULL-UFMG*
- *Submissão de tarefas computacionais*
- *Comparação entre o SLURM e o PBS Pro*

CAPÍTULO 1

# Guia básico para o SLURM do cluster BULL-UFMG

O SLURM é num escalonador de recursos que possui básicamente três funções, a saber:

- alocar nós computacionais para acesso exclusivo e/ou não-exclusivo (compartilhado) aos usuários por um determinado período de tempo necessário para executar as tarefas computacionais submetidas (*jobs*).
- oferece um ambiente integrado que permite executar e monitorar em tempo real as tarefas lançadas nos nós computacionais alocados.
- gerencia a fila de submissão, arbitrando conflitos entre os pedidos de recursos computacionais.

A lista de ferramentas essenciais que permite a interação dos usuários com o SLURM consiste no seguinte:

- **SRUN**: submete na linha de comando um *job* para execução.
- **SBATCH**: submete *scripts* shell (eg. bash) para a fila de espera do SLURM.
- **SALLOC**: reserva recursos do cluster (tempo de cpu, memória, quantidade de nós, etc) para a execução de uma tarefa computacional.
- **SBCAST**: para transmissão de um arquivo para todos os nós que foram alocados para um determinado *job*.
- **SCANCEL**: para cancelar um *job* que esteja sendo executado ou que ainda esteja na fila de espera.
- **SQUEUE**: para monitorar o estado dos *jobs* nas diversas filas de espera do SLURM.
- **SINFO**: para monitorar o estado global das partições configuradas (filas).
- **SVIEW**: providencia a integração de toda a informação e disponibiliza-a através de uma interface gráfica.

Uma mnemônica muito útil é lembrar que todos os comandos do SLURM começam pela letra `s`. Todos os comandos possuem manuais que podem ser acessados *online* via linha de comando, tal como por exemplo:

```
[test3@veredas0 ~]$ man sbatch
```

Em geral a grande maioria dos usuários precisa familiarizar-se com três comandos básicos do SLURM, nomeadamente o **sbatch**, **squeue** e **scancel**. Tal como no PBS Pro, a submissão de uma tarefa envolve a elaboração de um script shell (em geral em `bash`) que possui no seu preâmbulo diretrizes específicas do SLURM. Essas diretrizes começam com a palavra-chave `#SBATCH` (o análogo da palavra-chave `#PBS` para o PBS Pro). No corpo do script é então evocado o executável do programa com os seus respectivos parâmetros.

# Submissão de tarefas computacionais

## 2.1 Filas de submissão

No UNIVERSO do SLURM o conceito de `filas` envolve a noção de partição de nós computacionais em grupos. Esses grupos não são necessariamente disjuntos (nós podem participar de mais de uma partição) e podem ser solicitados de acordo com a especificação de recursos do sistema (tempo de *cpu*, memória, etc). Para respeitar a herança do PBS Pro iremos nos referir indistintamente aos termos `filas` e `partições`.

O cluster BULL-UFMG possui no momento três filas de submissão definidas apenas pelo recurso `walltime`:

- partição **full**: nós **veredas[12-107]** sem restrição de tempo.
- partição **short**: nós **veredas[2-11]** máximo 1 hora.

O usuário poderá listar a qualquer momento os recursos oferecidos por cada partição com o seguinte comando:

```
[test3@veredas0 ~]$ scontrol show partitions
PartitionName=full TotalNodes=96 TotalCPUs=768 RootOnly=NO
   Default=YES Shared=NO Priority=1 State=UP MaxTime=UNLIMITED Hidden=NO
   MinNodes=1 MaxNodes=UNLIMITED DisableRootJobs=NO AllowGroups=ALL
   AllocNodes=ALL
   Nodes=veredas[12-107] NodeIndices=10-105 DefaultTime=NONE

PartitionName=short TotalNodes=10 TotalCPUs=80 RootOnly=NO
   Default=NO Shared=NO Priority=1 State=UP MaxTime=01:00:00 Hidden=NO
   MinNodes=1 MaxNodes=UNLIMITED DisableRootJobs=NO AllowGroups=ALL
   AllocNodes=ALL
   Nodes=veredas[2-11] NodeIndices=0-9 DefaultTime=NONE
```

ou então com,

```
[test3@veredas0 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
full*        up   infinite     96   idle veredas[12-107]
short        up    1:00:00      9   idle veredas[2,4-11]
short        up    1:00:00      1   down veredas3.
```

Para racionalizar a utilização de recursos é conveniente que o usuário respeite a real demanda de sua tarefa e submeta seu script à partição mais apropriada, ou seja, *jobs* de curta duração (< 1 hora) na partição **short**. Caso contrário, pode correr o risco de a sua solicitação permanecer por muito tempo na fila de espera.

O estado dos *jobs* submetidos ou em execução poderá também ser examinado no SLURM, tal como era feito no PBS Pro com o comando *qstat*:

```
[test3@veredas0 ~]$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    179     short HelloWor    test3   R       0:02      2 veredas[4-5]
```

O cancelamento do *job* anterior pode ser feito facilmente do seguinte modo:

```
[test3@veredas0 ~]$ scancel -v 179
scancel: auth plugin for Munge (http://home.gna.org/munge/) loaded
scancel: Terminating job 179
```

Vejamos então como elaborar alguns scripts de submissão de acordo com o tipo de tarefa.

## 2.2 Tarefas seqüenciais

**Aviso:** Os exemplos de *scripts* de submissão que se seguem são para propósitos ilustrativos e representam *templates* que os usuários poderão adaptar conforme as suas necessidades específicas.

Executar um *job* seqüencial é bastante simples. Basta requisitar a duração necessária para executar a tarefa. Qualquer tarefa puramente seqüencial, que não utiliza MPI ou threads, irá usar necessariamente `um core apenas` do nó computacional.

```
#!/bin/bash
#SBATCH --job-name="TESTJOB"
#SBATCH --ntasks=1
#SBATCH --time=01:00:00

./a.out > OUTFILE
```

Por defeito o *output* do script acima será redirecionado para um arquivo do tipo *slurm*-XXXX.out, onde `XXXX` é o número do *job* que foi atribuído pelo SLURM. Este arquivo de *output* estará localizado no diretório de onde foi feita a submissão. O interessante a notar, e a que nos referiremos mais adiante (*ver*), é de que este arquivo é criado no instante da execução do *script* e o seu conteúdo aumenta à medida que o programa vai sendo executado. Remover este arquivo antes da finalização do *job* acarreta portanto perda total do *output* da tarefa. Diferentemente no PBS Pro os arquivos de *output* só eram criados na área do usuário depois de o *job* finalizar. É possível especificar um nome para o arquivo de *output* fazendo uso da diretriz `#SBATCH --output=meu_job.out`. O mesmo vale para o arquivo de erro, `#SBATCH --error=meu_job.err`.

## 2.3 Tarefas paralelas (MPI)

O exemplo seguinte representa a submissão de um *job* paralelo MPI que foi compilado com o compilador proprietário da BULL (MPI BULL): `mpicc -DALIGNED -c hello.c -o hello.o; mpicc hello.o -o hello.x`. Nesta situação permitimos que o SLURM se encarregasse não só da alocação dos recursos, mas também da execução do programa e da inicialização das comunicações na *Infiniband* através do comando `srun`.

```
#!/bin/bash
#SBATCH -J HelloWorld
#SBATCH --ntasks 16
#SBATCH --time 00:30:00
#SBATCH --partition short
```

```
source /opt/mpi/mpibull2-1.3.9-18.s/share/mpibull2.sh
mpibull2_devices -d=ibmr_gen2
echo '-----------------------------------------'
ldd ./hello.x
echo '-----------------------------------------'
sleep 15

srun hostname | sort

srun  ./hello.x
```

Segue o resultado da execução deste script:

```
[test3@veredas0 SLURM]$ cat slurm-188.out
-----------------------------------------
     libmpi.so => /opt/mpi/mpibull2-1.3.9-18.s/lib/libmpi.so (0x00002b1b48343000)
     librt.so.1 => /lib64/librt.so.1 (0x00000038d1a00000)
     libdl.so.2 => /lib64/libdl.so.2 (0x00000038d0e00000)
     libpmi.so => /usr/lib64/libpmi.so (0x00002b1b486a0000)
     libpthread.so.0 => /lib64/libpthread.so.0 (0x00000038d1200000)
     libuuid.so.1 => /lib64/libuuid.so.1 (0x00000038e1800000)
     libm.so.6 => /lib64/libm.so.6 (0x00000038d0a00000)
     libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00000038dba00000)
     libc.so.6 => /lib64/libc.so.6 (0x00000038d0600000)
     libmpidev.so => /opt/mpi/mpibull2-1.3.9-18.s/lib/drivers/ibmr_gen2/libmpidev.so (0x00002b1b488a
     /lib64/ld-linux-x86-64.so.2 (0x00000038d0200000)
     libslurm.so.20 => /usr/lib64/libslurm.so.20 (0x00002b1b48ba6000)
     libibverbs.so.1 => /usr/lib64/libibverbs.so.1 (0x00002b1b48e66000)
     libsvml.so => /opt/intel/Compiler/11.1/069/lib/intel64/libsvml.so (0x00002b1b49074000)
-----------------------------------------
veredas4
veredas4
veredas4
veredas4
veredas4
veredas4
veredas4
veredas4
veredas5
veredas5
veredas5
veredas5
veredas5
veredas5
veredas5
veredas5
Hello MPI: processor 5 of 16 on veredas4
Hello MPI: processor 15 of 16 on veredas5
MPIBull2 1.3.9-s (Astlik) 20091113-1606 MPI_THREAD_FUNNELED (job 188) (device gen2)
Hello MPI: processor 0 of 16 on veredas4
Hello MPI: processor 1 of 16 on veredas4
Hello MPI: processor 2 of 16 on veredas4
Hello MPI: processor 3 of 16 on veredas4
Hello MPI: processor 4 of 16 on veredas4
Hello MPI: processor 7 of 16 on veredas4
Hello MPI: processor 8 of 16 on veredas5
Hello MPI: processor 6 of 16 on veredas4
Hello MPI: processor 9 of 16 on veredas5
Hello MPI: processor 10 of 16 on veredas5
```

```
Hello MPI: processor 11 of 16 on veredas5
Hello MPI: processor 12 of 16 on veredas5
Hello MPI: processor 13 of 16 on veredas5
Hello MPI: processor 14 of 16 on veredas5
```

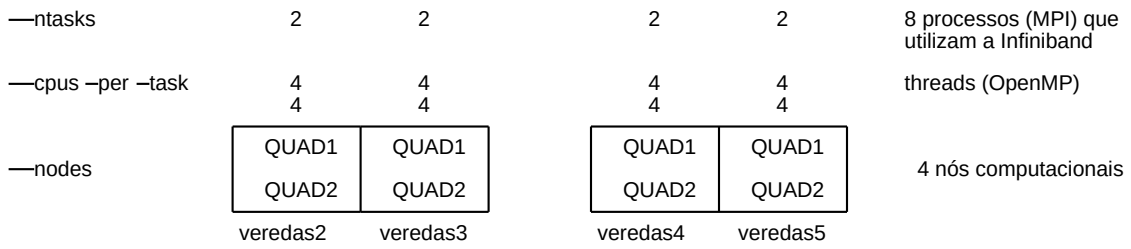
## 2.4 Tarefas paralelas com threads (MPI+OpenMP)

Cada nó computacional do cluster BULL da UFMG possui `2 quad-cores` totalizando `8 cores`. Mostramos a seguir um exemplo de submissão de uma tarefa paralela (MPI+OpenMP) que utiliza a *Infiniband*, cujo executável foi compilado com o `Intel MPI` e que requisita `8 processos`, sendo que cada processo é um conjunto de `4 threads`.

Repare que propositalmente, e contrariamente ao caso anterior, a ferramenta `mpiexec` foi utilizada em vez do `srun` para lançar os processos MPI. O SLURM foi então utilizado apenas para reservar os recursos necessários.

```
#!/bin/bash
#SBATCH -J HelloWorld
#SBATCH --partition short
#SBATCH --nodes 4
#SBATCH --ntasks 8
#SBATCH --cpus-per-task 4
#SBATCH --time 00:30:00

source /opt/intel/impi/4.0.0/bin64/mpivars.sh

export OMP_NUM_THREADS=4
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
export I_MPI_FABRICS=dapl

sleep 20

echo '-------------------------------------'
ldd ./hello
echo '-------------------------------------'
srun hostname -s | sort -u >slurm.hosts


mpdboot -n $SLURM_NNODES -f slurm.hosts -r ssh
mpiexec -perhost 2 -np 8 ./hello
mpdallexit
```

O diagrama seguinte procura ilustrar a forma como os processos MPI e seus respetivos *threads* foram distribuídos nos `32 cores` dos quatro nós requisitados. A descrição detalhada de cada diretriz `#SBATCH` poderá ser consultada *online* através do comando `man sbatch`. A descrição de cada *flag* do mpiexec também pode ser obtida através de `/opt/intel/impi/4.0.0/bin64/mpiexec -help`.



# Comparação entre o SLURM e o PBS Pro

Segue alguns comandos essenciais do SLURM e seus equivalentes no PBS Pro.

| Comando | Descrição | PBS Pro |
|---|---|---|
| `sbatch` | submete um *script* shell | `qsub` |
| `srun` | submete um comando via SLURM no modo interativo | `qsub -I` |
| `squeue` | lista todos os jobs (em execução ou no modo de espera) nas filas | `qstat` |
| `scontrol` | modifica o estado de um *job* (maioria só é permitida ao *root*) | `qalter` |
| `scancel` | cancela um *job*, quer em execução ou na fila de espera | `qdel` |

## 3.1 Variáveis de ambiente

Existem também algumas diferenças no tocante às variáveis de ambiente como ilustra a seguinte tabela:

| SLURM | PBS Pro |
|---|---|
| $SLURM_JOB_ID | $PBS_JOBID |
| $SLURM_JOB_NAME | $PBS_JOBNAME |

O SLURM não possui o equivalente à variável `$PBS_NODEFILE` que aponta para o arquivo que contém os nomes dos nós computacionais reservados para a execução do *job*. Contudo, quando necessário, é possível criar manualmente esse tipo de arquivo com o comando *srun*:

```
#!/bin/bash -x

HOSTFILE=/tmp/hosts.$SLURM_JOB_ID

srun hostname -s > $HOSTFILE

if [ -z "$SLURM_NPROCS" ] ; then
  if [ -z "$SLURM_NTASKS_PER_NODE" ] ; then
    SLURM_NTASKS_PER_NODE=1
  fi
  SLURM_NPROCS=$(( $SLURM_JOB_NUM_NODES * $SLURM_NTASKS_PER_NODE ))
fi

/path/to/mpirun -machinefile $HOSTFILE -np $SLURM_NPROCS programa.x
```

```
rm /tmp/hosts.$SLURM_JOB_ID
```

## 3.2 Diretório de execução

Uma outra diferença importante que distingue o SLURM do PBS Pro é que enquanto no primeiro escalonador a tarefa *batch* é executada automáticamente no diretório de onde foi submetido o seu *script*, o segundo precisa explicitamente que seja introduzida a seguinte linha:

```
#!/bin/bash -x
...
cd $PBS_O_WORKDIR
...
```

As variáveis de ambiente que são definidas durante uma sessão shell do SLURM são automáticamente exportadas para o *job batch* em todos os nós computacionais no instante em que o script entra em execução. Já no *PBS Pro* é necessário garantir isso com a introdução da diretriz:

```
#!/bin/bash -x
...
#PBS -V
...
```

## 3.3 Output

O SLURM apresenta ainda uma vantagem considerável e muito útil para aqueles usuários que precisam inspecionar numéricamente e em tempo real a evolução do seu programa, como por exemplo, a convergência de uma determinada iteração: os arquivos de saída `stdout` e de erro, `stderr` (*ver as flags do SLURM*) , são criados automáticamnte no diretório de onde foi submetido o *script* e vão crescendo gradualmente de tamanho à medida que a simulação evolui no tempo. O **PBS Pro** mostra-se insuficiente neste quesito: somente no final da execução da tarefa é que os arquivos de saída são escritos no diretório destinado.

## 3.4 Directrizes de submissão

Finalmente com o intuito de auxiliar os usuários na migração dos seus scripts de submissão do **PBS Pro** para o **SLURM** apresentamos a seguinte tabela comparativa das diretrizes de submissão:

| PBS Pro | SLURM |
|---|---|
| #PBS -N *job_name* | #SBATCH –job-name="*job_name*" ou #SBATCH -J *job_name* |
| #PBS -l nodes=*n* | #SBATCH –nodes=*n* |
| #PBS -l walltime=*HH:MM:SS* | #SBATCH –time=*HH:MM:SS* |
| #PBS -l min_walltime=*HH:MM:SS* | #SBATCH –time-min=*HH:MM:SS* |
| #PBS -q <*queue*> | #SBATCH –partition=<*queue*> |
| #PBS -l mppwidth=*n* | #SBATCH –ntasks=*n* |
| #PBS -l mppnppn=*N* | #SBATCH –ntasks-per-node=*N* |
| #PBS -l mppdepth=*d* | #SBATCH –cpus-per-task=*d* |
| #PBS -l mppmem=*m*M | #SBATCH –mem=*m* |
| #PBS -l mppnodes=<*nid-list*> | #SBATCH –nodelist=<*nid-list*> |
| #PBS -W group_list=*a_group* | #SBATCH –account=*a_group* |
| #PBS -o */path/to/stdout* | #SBATCH –output=*/path/to/stdout* (pode ser usado %j para *jobid*) |
| #PBS -e */path/to/stderr* | #SBATCH –error=*/path/to/stderr* |
| #PBS -V | *não é necessário* |

Devemos também ressaltar que as diretrizes do SLURM podem ser especificadas na linha de comando em vez de colocadas no *script*:

```
[test3@veredas0 ~]$ sbatch --job-name="meu job" --ntasks=N ...
```

**Aviso:** Para uma informação mais detalhada sobre o escalonador **SLURM** o usuário poderá *baixar* e consultar o arquivo `Bullx cluster suite: User's Guide`. Para quem estiver interessado o Swiss National Supercomputing Centre disponibiza um video no qual são feitas várias demonstrações de como o usuário pode interagir com o **SLURM**