# Submissão Tradicional

**Atualmente recomendamos a submissão via GO para todos os usuários, inclusive os que compilam a própria apliação: [[SubmissaoSimplificada]]**

## Compilar

Você pode compilar seu códigos nessa máquina. Estão disponíveis:

- GCC (C, C++ e Fortran)
- Intel (C, C++ e Fortran)

Também está disponível o Intel MPI para compilações de programas paralelos utilizando MPI. Para isso utilize mpicc e suas variantes (mpigcc, mpiicc, mpif90...)

**ATENÇÃO: NÃO EXECUTE AS SIMULAÇÕES NESSA MÁQUINA. NÃO RODE PROGRAMAS MPI NESSA MÁQUINA. APENAS COMPILE.**

Sempre que possível use compilação estática ou ainda use a ferramenta condor_compile.

## Submeter para o grid

Para submeter jobs para o grid será necessário criar um arquivo de submissão informando qual programa executar, quais arquivos copiar na entrada e saída...

Abaixo estão dois exemplos, um para programas simples e outro para programas MPI.

Em qualquer caso é necessário autenticar o seu certificado antes de iniciar a submissão:

```
myproxy-logon
```

a chave valid permite especificar a duração da proxy, é importante definir esta variável
por um tempo **superior a duas vezes a duração dos processos submetidos**.

O padrão desse comando (sem a chave -t) é 12 horas. Para requisitar 72 horas:

```
myproxy-logon -t 72
```

Para submeter:

```
condor_submit arquivo.submit
```

### Exemplo de arquivo de submissão para programas simples:

```
universe=grid
grid_resource=gt2 ce.grid.unesp.br/jobmanager-pbs

executable = executavel
arguments = argumentos

transfer_input_files = arquivo_de_entrada
transfer_output_files = arquivo_de_saida
WhenToTransferOutput = ON_EXIT

log       = test.log
output    = test.out
error     = test.error
```

```
queue
```

### Exemplo de arquivo de submissão para programas MPI:

```
universe=grid
grid_resource=gt2 ce.grid.unesp.br/jobmanager-pbs

# 16 é o número de processos MPI
globusrsl = (host_xcount=16)(jobType=mpi)

executable = programa_mpi
arguments = argumentos

log       = test.log
output    = test.out
error     = test.error

transfer_input_files = arquivo_de_entrada
transfer_output_files = arquivo_de_saida

WhenToTransferOutput = ON_EXIT

queue
```

## Verificar o status

Para verificar a fila de processos:

```
condor_q
```

Ou para listar apenas os seus processo:

```
condor_q username
```

## Cancelar job

Para remover um job, é necessário saber o seu número (verifique no item anterior):

```
condor_rm numero_do_processo
```

Para remover todos os seus processos:

```
condor_rm username
```

# Parâmetros para a submissão

Aqui estão descritos os parâmetros que devem ser especificados no arquivo de submissão dos jobs.

## Como utilizar

Adicionar o parâmetro "globusrsl" no arquivo de submissão.

Exemplo:

```
globusrsl = (queue=long)
```

Exemplo utilizando MPI com multiplas opções:

```
globusrsl = (host_xcount=20)(jobType=mpi)(queue=long)
```

# Opções

## Tempo de execução

A opção **queue** limita o tempo máximo de execução de um processo.

Caso não seja especificado, será adotado o tempo padrão (24 horas).

Escolhendo a categoria de menor duração garante um início mais rápido dos jobs.

- **short**: execução limitada a 24 horas
- **long**: execução limitada a 30 dias
- **medium**: execução limitada a 7 dias

```
globusrsl = (queue=long)
```

## Programas multi-thread (SMP)

Essa opção requisita mais de um core em um mesmo servidor.

Ideal para programas paralelizados com memória compartilhada (SMP).

```
globusrsl = (xcount=4)(jobType=single)
```

## MPI

Permite a execução de programas paralelos que sejam MPI.

```
globusrsl = (host_xcount=20)(jobType=mpi)
```

## Paralelo, não MPI

Essa opção só deve ser utilizada por usuários avançados. Ela permite a execução de programas paralelos que não sejam MPI.

Cabe ao usuário fornecer um script para configurar o ambiente paralelo.

```
globusrsl = (host_xcount=30)(jobType=single)
```

# Ajustes de performance

## Escrita de arquivos no /tmp

Por padrão, os jobs enviados são executados de um diretório compartilhado por NFS. Esse diretório tem uma performance menor que o disco local dos nós (/tmp).
Configure o seu programa para escrever os arquivos temporários (scratch) no /tmp. Essa medida simples pode melhorar a execução do seu programa em algumas vezes, dependendo do padrão de escrita.

Observe o exemplo abaixo, em que o mesmo job foi disparado duas vezes e medidos os respectivos tempos de execução. Na primeira execução o job foi configurado para escrever no diretório /tmp, enquanto que na segunda o job não foi modificado, sendo disparado através das configurações padrões.

```
carlosm@treinamento01:~/jobs/produtor_consumidor/pthread$ cat error.tmp
```

```
real    0m38.973s
user    0m22.610s
sys    2m1.604s


carlosm@treinamento01:~/jobs/produtor_consumidor/pthread$ cat error.sem_tmp

real    1m8.418s
user    0m22.579s
sys    2m21.039s
```

Repare que o ganho, em tempo de execução, do job configurado para escrever no diretório /tmp, sobre o job com as configurações padrão, foi de aproximadamente 50%.

Caso o seu programa não aceite como parâmetro um diretório de execução temporário, pode-se utilizar um script que mova os dados para o local correto e traga de volta.
Segue um exemplo:

script **wrapper.sh**

```
#!/bin/bash

OLD_PWD=`pwd`
TMP_DIR="/tmp/$USER-$$"
mkdir $TMP_DIR
cp * $TMP_DIR
cd $TMP_DIR

chmod +x $1

$*

rsync -r . $OLD_PWD/
cd $OLD_PWD
rm -rf $TMP_DIR
```

Trecho do arquivo de submissão

```
(..)

executable = wrapper.sh
arguments = ./programa arg1 arg2

transfer_input_files = programa, outros_arquivos_de_entrada

(...)
```

Lembre de apagar qualquer arquivo temporário no fim da execução.

## Submissão de múltiplos jobs

Quando submeter diversos jobs ou jobs com várias tarefas, inclua sempre no arquivo de submissão:

```
stream_output = False
stream_error = False
```

Isso evita sobrecarregar o cluster.

# Depuração

Antes de começar é importante ressaltar que o ambiente de grid adiciona uma camada de complexidade ao seu problema. Apenas submeta programas que tenham sido testados em um ambiente padrão de desenvolvimento.

A ferramenta Condor:http://www.cs.wisc.edu/condor/ é uma ferramenta poderosa que usa de forma oportunística recursos computacionais, da mesma forma que a ave sobrevoa os altiplanos andinos a procura de algum animal morto para se alimentar. Ela é versátil e pode ser utilizada em ambientes, heterogêneos, e sujeitos a falhas. Mas uma vez disparado um job, o usuário não tem controle sobre ele com exceção da possibilidade de interromper a execução.

Para iniciar o processo de depuração descubra o número do seu job:

```
condor_q username


-- Submitter: access.grid.unesp.br : <200.145.46.37:59072> : access.grid.unesp.br
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
14762.0   lemke           4/9  13:51   0+00:25:11 R  0   2.4  mdrun_mpi -v -s fu
14764.3   lemke           4/9  13:57   0+00:00:00 I  0   0.0  geraforcas -l 30 -

2 jobs; 1 idle, 1 running, 0 held
```

O número do job é o ID de seu job. Nesse caso o usuário lemke possui dois jobs 14762.0 e 14764.3.
Observe o status (ST) de seu job:

- R significa que o seu job está em execução (RUNNING)
- I significa que o seu job está Idle (em espera)
- H significa que seu job não foi executado até o final (HALTED)

Outra possibilidade é usar a chave -globus

```
condor_q -globus lemke


-- Submitter: access.grid.unesp.br : <200.145.46.37:59072> : access.grid.unesp.br
 ID      OWNER          STATUS  MANAGER  HOST              EXECUTABLE
12131.0   ipgtt        STAGE_OUT condor   ce.grid.unesp.br    /opt/condor-7.2.4/
12145.0   ipgtt        STAGE_OUT condor   ce.grid.unesp.br    /opt/condor-7.2.4/
12717.0   morgon         UNKNOWN condor   ce.grid.unesp.br    /home/morgon/bin/R
12718.0   morgon         UNKNOWN condor   ce.grid.unesp.br    /home/morgon/bin/R
12720.0   morgon         UNKNOWN condor   ce.grid.unesp.br    /home/morgon/bin/R
13948.0   jdreis       STAGE_OUT condor   ce.grid.unesp.br    /home/jdreis/sempi
14486.0   trtomei      STAGE_OUT condor   ce.grid.unesp.br    /home/trtomei/CMSS
14772.0   fabiofima    STAGE_OUT condor   ce.grid.unesp.br    /opt/gromacs/bin/m
14773.0   grilo         ACTIVE condor   ce.grid.unesp.br    /home/grilo/teste/
14774.0   lemke        UNSUBMITTED condor   ce.grid.unesp.br    /home/lemke/surf/.
14774.1   lemke          PENDING condor   ce.grid.unesp.br    /home/lemke/surf/.
14774.2   lemke          PENDING condor   ce.grid.unesp.br    /home/lemke/surf/.
14774.3   lemke          PENDING condor   ce.grid.unesp.br    /home/lemke/surf/.
14774.4   lemke         STAGE_IN condor   ce.grid.unesp.br    /home/lemke/surf/.
```

O comando mais importante para depurar um programa é

```
condor_q -l numjob
```

Este comando indica uma série de variáveis que podem ser úteis para checar se o job submetido é adequado. Em particular verifique:

- os arquivos de entrada estão corretos?
- os arquivos de saída estão corretos?
- o executável existe?
- os parâmetros estão corretos?

Para simplificar o processo é conveniente utilizar um script para submeter um job. Isto facilita o processo de depuração. No script de submissão você pode realizar testes preliminares, compilar o programa se este for o caso, ou ainda realizar varias etapas do seu processo computacional em um único job.

```
#!/bin/bash
./configura
echo configurou
condor_compile g++ -o prog prog.cc -lm
echo compilou
./prog
echo rodou
```

Neste exemplo executamos um script de configuração, compilamos nosso programa e executamos.

O condor_compile é uma boa alternativa para compilar programas pois garante que o seu job esteja adequado ao ambiente Grid. A sintaxe é simples: anteceda o comando de compilação com o condor_compile. Além disso compilar o programa na máquina em que o programa irá rodar aumenta sua portabilidade e facilita a depuração de erros como falta de bibliotecas.

Você pode escrever mensagens usando o comando echo, o que pode facilitar a depuração em muitos casos.

Para ler as messagens de erro  e de saída do seu job, acesse os arquivos de erro, log e saída indicados em seu arquivo de submissão. Nos casos em que for solicitar ajuda para outros usuários ou para a equipe de suporte do NCC, inclua estes arquivos em seu e-mail.

O ambiente de grid está sujeito  a problemas que ocorrem em estações de trabalho, como quedas de energia, máquinas com problema de hardware, etc. Em alguns casos esta pode ser a explicação para o problema observado com seu job.

Além disso a complexidade e os recursos utilizados por um job ou pelo usuário, podem também ser responsáveis por problemas de execução. Uso excessivo de disco e memória podem causar problemas. Algumas considerações sobre stes limites:

- um único usuário só pode rodar simultaneamente 100 jobs, mas pode submeter um número muito maior de jobs,
- jobs com mais de 2 Gb de memória podem ocasionar problemas de execução e
- jobs mpi que usem muitos slots podem demorar para iniciar sua execução.

Uma importante diferença entre o Condor:http://www.cs.wisc.edu/condor/ e o Condor-G:http://www.cs.wisc.edu/condor/condorg/ utilizado no NCC, é que no segundo ambiente os arquivos de saída devem ser explicitados no arquivo de submissão.

A principal saída de erro é:

```
HoldReason = "Globus error 155: the job manager could not stage out a file"
```

Ela ocorre sempre que o arquivo de saída não for gerado. Isto pode ocorrer por diversos motivos:

- seu executável não conseguiu iniciar (neste caso a execução durou muito pouco tempo)
- seu executável iniciou mas não concluiu (a execução custou algum) e o arquivo de saída não foi gerado
- o nome do arquivo de saída está errado

Um exemplo de script para testar estas possbilidades

```
#!/bin/bash
ls -l
# testa se os arquivos foram transferidos
./configura
date
echo configurou
condor_compile g++ -o prog prog.cc -lm
date
echo compilou
./prog
date
ls -l
echo rodou
```