Introdução ao SLURM

(Versão em português)

António Pina
(pina@di.uminho.pt)

Este documento cobre o uso básico da infraestrutura SLURM (ver original) .

❖ Inspecionar o estado do cluster

Existem dois comandos principais que podem ser usados para exibir o estado do cluster. São *sinfo*, para mostrar informações do nó, e *squeue*, para mostrar informações do trabalho.

Informação do nó

sinfo exibe o estado dos nós, estruturados por partição (fila).

```
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch-short    up 2-00:00:00      3  alloc hpc[004-005,007]
batch-short    up 2-00:00:00      7   idle hpc[006,008-013]
batch-long     up   infinite      1  down* hpc170
```

O mais relevante é provavelmente a coluna STATE, que agrupa grupos de nós de computação, por partição, com o mesmo estado.

- Os nós em estado *idle* não estão em uso no momento e, portanto, um trabalho usando esses nós pode ser agendado imediatamente para execução.
- Os estados *alloc* ou *mix* estão atualmente em uso.
- *alloc* significa que todo o nó está em uso
- *mix* (não exibido acima) indica que um utilizador está usando apenas parte dos recursos deste nó
- *down* significa que o nó não está disponível.

A estrela * após um estado é irrelevante para o usuário, apenas indica que SLURM não é capaz de contactar este nó.

❖ Informação de trabalho (job)

squeue exibe uma linha por trabalho, mostrando o estado do trabalho (normalmente executando "R" ou "PD" pendente). A maioria das colunas são auto descritivas, conforme mostrado no exemplo abaixo.

```
$ squeue
  JOBID PARTITION  NAME  USER  ST   TIME     NODES   NODELIST(REASON)
  15002 batch-sho  gs    ken    R    8:51:31     1    hpc004
  14907 be-long    en   theresa R    1-17:40:40  2    hpc-be[029,034]
```

Outras opções úteis incluem exibir apenas os seus próprios trabalhos, usando *squeue -u $ USER*, ou trabalhos de uma partição específica, como *squeue -p batch-long*

❖ Trabalhos em execução

Existem três maneiras de executar trabalhos:

- *srun* para execuções interativas. Usado para testes rápidos, mas bloqueia o terminal até que o trabalho seja concluído.
- *sbatch* para submissão de lotes. Este é o caso principal, pois permite criar um *script* de submissão onde se especificam todos os argumentos, comandos e comentários para lançar o trabalho específico. Também é útil para registar ou partilhar a forma como o trabalho é executado.
- *salloc* simplesmente aloca recursos (normalmente um conjunto de nós), desemparelhando a alocação da execução. Usado quando é preciso usar os comandos mpirun / mpiexec. Com **salloc**, aloca-se um conjunto de nós, e mpirun / mpiexec normalmente usa os parâmetros de alocação do ambiente, sem necessidade de usar *hostfiles*.

❖ Submissão em lote (*batch*)

A submissão em lote consiste num ficheiro, que é essencialmente apenas um *script* que informa o SLURM da quantidade de recursos necessários (por exemplo, partição, número de tarefas / nós) como esses recursos serão usados (por exemplo, tarefas por nó), e um ou diferentes etapas do trabalho (ou seja, execuções do programa). Este arquivo é então processado usando o comando *sbatch*.
Considere, por exemplo, o seguinte ficheiro:

```
#!/usr/bin/bash
#SBATCH -p batch-short
#SBATCH -t 1:00:00
#SBATCH -n 64

srun ./mpi_program parameters
```

A primeira linha indica o tipo de *shell* que será executado, normalmente o *bash*. Em seguida, os parâmetros de submissão SLURM são definidos por linhas que começam com #SBATCH, seguidos pelos parâmetros de submissão documentados na página de manual do *sbatch* (man sbatch), também acessível em sbatch documentation online. Esses parâmetros são quase um espelho **1: 1** das opções disponíveis para *srun*.
No caso do *script* acima, o pedido é para a partição `batch-short` (argumento *-p* ou *--partition*), definindo um limite de tempo máximo de 1 hora (argumento -t ou --time) e solicitando 64 tarefas (-n ou *--ntasks*).
Para fazer a submissão, pode usar-se o comando abaixo em que *file* contém a *script*

```
$ sbatch file
Submitted batch job 15276
```

Por omissão, tanto a saída de dados como os erros padrão são direcionados para um ficheiro "`slurm-% j.out`", onde "`% j`" é substituído pelo número de alocação do *job*. Para além do próprio *script*, o Slurm não manipula os ficheiros do utilizador.
Para obter mais detalhes sobre os diferentes padrões e parâmetros de submissão, consultar Submission patterns.

❖ Consulta aos trabalhos (*jobs*) concluídos

Para o efeito pode ser usado o comando *sacct*, que produzirá o nome do job, a quantidade de nós alocados, cpus, memória, estado do trabalho e assim por diante. É possível obter o código de saída e o estado final da tarefa, mas não a saída ou o registo completo de saída.

```
$sacct -j 15374

JobID        JobName     Partition    Account  AllocCPUS    State       ExitCode
---------- -------    ---------- ---------- -------   ------ --------
15374        helloworld batch-sho+ hpc-batch+    1        COMPLETED      0:0
15374.batch  batch       hpc-batch+              1        COMPLETED      0:0
15374.0      echo        hpc-batch+              1        COMPLETED      0:0
```

Neste caso, podem ver-se três linhas diferentes. Esses detalhes são irrelevantes, mas é desta forma que o SLURM reporta as informações do *job*.
A primeira linha contém as informações (globais) do trabalho. A segunda linha apenas descreve o fato de que este é um trabalho em *batch*. Se tivéssemos usado *srun*, veríamos apenas a primeira linha. Finalmente, vemos uma entrada de linha para cada etapa do trabalho, uma para cada *srun* que foi iniciado de dentro do script de submissão. Este *job* apenas executou *srun echo* uma vez e, portanto, criou uma única etapa de *job* (15374.0) cujo JobName será, por omissão, o nome do programa, neste caso "*echo*".
Também podem ser listados os nós que estiveram envolvidos, da seguinte maneira:

```
sacct-j15370 –format JobID, JobName, Partition, Account, AllocCPUS, State, ExitCode, NodeList

JobID         JobName   Partition    Account  AllocCPUS  State   ExitCode   NodeList
------------ ----- --------- ---------- ------ ----    -----   --------
15370        triMagnet+  be-long   te-vsc-scc   280     FAILED    127:0     hpc-be[010,079+
15370.batch       batch            te-vsc-scc    40     FAILED    127:0     hpc-be010
15370.0      hydra_pmi+            te-vsc-scc    8     FAILED      7:0      hpc-be[010,079+
```

Por definição, só podem ser consultar os próprios *jobs.*

[amp@search7 Outros comandos

```
$sacctmgr list qos format=name,maxwall,MaxJobsPU,priority
$scontrol show -d job 2376
```

Comandos de submissão Slurm

Esta seção descreve os passos mínimos para começar a submeter trabalhos em SLURM Linux HPC. Para permissão de acesso ao Linux HPC, consulte Section Access.

Primeiro, faça login por ssh ao hpc-batch:

```
$ ssh hpc-batch.cern.ch
```

Antes de iniciar um trabalho, deve selecionar-se um ambiente MPI. A disponibilidade do módulo em execução listará as distribuições MPI disponíveis. Para carregar MVAPICH2-2.2, que é uma distribuição estável que funciona bem neste cluster, use o seguinte.

```
$module load mpi/mvapich2/2.2
```

Submeta o trabalho para uma das partições disponíveis. Para trabalhos de curta duração (<48h), pode enviar para a partição curta, caso contrário, envie para a partição longa. Pode ver-se o estado do cluster, incluindo partições e a quantidade de nós estão disponíveis em cada partição usando `sinfo`. O uso do `squeue` exibirá os trabalhos atualmente em execução (ou enfileirados) para cada partição.

Ao enviar um trabalho, no mínimo deve-se especificar: * A partição (argumento -p) * O tempo de execução máximo (tempo de relógio) para o trabalho. (argumento -t) * O número de tarefas (argumento -n)

Por exemplo, para enviar um trabalho de 64 tarefas para a partição de lote curto com um limite de tempo de 1h, pode usar-se o seguinte:

```
$ srun -p batch-short -t 1:00:00 -n 64 ./mpi_program parameters
```

Para obter mais informações sobre os parâmetros srun, consulte a documentação SLURM documentation.

A principal limitação do srun é que ele bloqueará o terminal até que a execução do trabalho esteja concluída. Para o envio mais **tradicional de lotes** , pode usar-se sbatch em vez de srun e colocaria os parâmetros num ficheiro de submissão de lote. Essa seria a maneira recomendada de trabalhar, embora srun possa ser útil para tentar execuções rápidas. A maneira equivalente de lançar o comando acima usando sbatch seria a seguinte. Imagine que temos o seguinte ficheiro de submnissão em lote chamado myjob:

```
#!/usr/bin/bash
#SBATCH -p batch-short
#SBATCH -t 1:00:00
#SBATCH -n 64

srun ./mpi_program parameters
```

E a submissão deste trabalho teria a seguinte forma:

```
$ sbatch myjob
```

Neste momento, SLURM irá imediatamente enfileirar o trabalho num lote curto, fornecer-nos o ID do trabalho recém-criado e regressar à *shell*. Neste ponto, é possível verificar o estado de nossos trabalhos enviados usando:

```
$ squeue -u $USER
```

Ou um trabalho específico, digamos JobID 100, usando:

```
$ squeue -j 100
```

Também pode cancelar-se um trabalho a qualquer momento usando o scancel. Para cancelar um trabalho, basta anexar o JobID (s) (lista separada por vírgulas) para verificar:

```
$ scancel 100
```

Padrões e parâmetros de envio comuns

Os parâmetros de envio dependerão muito das características da aplicação, incluindo a escalabilidade.

Na sua forma mais básica, apenas se selecionaria o número de tarefas (ou seja, o número de processos) que serão executados. Cada tarefa será executada num núcleo de CPU, e se forem solicitados mais núcleos do que os disponíveis num único nó (o que deveria estar a ser feito), as tarefas do trabalho serão distribuídas por vários nós e normalmente comunicariam por MPI.

Posicionamento de tarefa

O posicionamento da tarefa pode afetar muito o desempenho e, para isso, a topologia da CPU e a densidade do posicionamento da tarefa / nó terão um grande papel. Novamente, o resultado dependerá muito de cada aplicação, portanto, algumas tentativas e erros ou, de preferência, um conhecimento profundo da parte interna da aplicação, até mesmo a criação de perfis, ajudará a determinar a melhor maneira de executar uma aplicação MPI.

Geralmente, desejará experimentar usando as seguintes combinações de opções:

`--ntasks.` Usar esta opção sozinha é suficiente em muitos casos e funciona bem ao usar um múltiplo do número de núcleos disponíveis para um nó.
--**`nodes`** e **`--ntasks-per-node`**. Também pode ser definido o número de nós que deseja usar e quantas tarefas por nó o trabalho precisa. Se não se definir **`--ntasks`**, o SLURM o definirá como **`nodes * ntasks-per-node`**. Se for definido, será verificado se a equação anterior é válida.

Outras opções

Memória por CPU. Por padrão, o SLURM oferece uma partilha de memória proporcional à quantidade de núcleos de CPU solicitados. Portanto, se forem solicitados todos os núcleos da CPU, será obtida toda a memória. Podendo essa opção substituída por:
`--mem`, para definir a memória necessária por nó.
`--mem-per-cpu`, para substituir a memória padrão por cpu.

Configurar o nome do trabalho, usando **`--job-name`**. Este é o nome que será exibido no `squeue`.

Altere o *e-mail* padrão em caso de falha, usando **`--mail-user.`**

Consulte a documentação da página man do `sbatch` para obter as opções e explicações completas.

Aplicações híbridas OpenMP + MPI

Não é preciso fazer nada especial para executar programas compilados com OpenMP na infraestrutura Linux HPC. Apenas leve em consideração que Slurm não sabe qual o número de *threads* do OpenMP que se pretendem mas pode ser dado uma dica relevante usando **`--cpus-per-task.`**
No entanto, cada compilador e estrutura MPI tratará a afinidade de uma maneira diferente, e as configurações padrão podem resultar num mau desempenho.
Um exemplo de uso de OpenMP (com, por exemplo, OpenMPI) a seguir:

```
#!/bin/bash

#SBATCH --job-name openmp-test
#SBATCH --partition inf-short
#SBATCH --time 00:01:00
#SBATCH --ntasks 12
#SBATCH --cpus-per-task 2

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun --ntasks $SLURM_NTASKS ./a.out
```

Hyperthreading

Os processadores modernos fornecem uma forma de paralelismo chamada hyperthreading, em que um único núcleo físico aparecerá como dois núcleos para o sistema operacional. Enquanto algumas unidades de execução funcional serão "espelhadas" para fornecer execução paralela, nem todos os recursos de hardware fornecerão esse tipo de paralelismo e, portanto, a aplicação normalmente não será capaz de atingir um aumento de velocidade 2x. Na verdade, algumas aplicações podem ter um desempenho degradado como resultado de hyperthreading, por exemplo, devido ao aumento do número de tarefas que se tornam um gargalo de memória.

Os recursos do Linux HPC têm hyperthreading habilitado, mas se a sua aplicação beneficiará ou não disso vai depender muito da própria aplicação. Isso é algo que cada utilizador deve tentar por conta própria ou seguir as recomendações do fornecedor da aplicação.

Uso de Hyperthreading

Os utilizadores não precisam fazer nada para usar o hyperthreading. Como o hyperthreading já está habilitado no processador, apenas usando todos os núcleos disponíveis, as aplicações estarão usando efetivamente o hyperthreading.

Desativação Hyperthreading

Embora não seja possível desabilitar o hyperthreading, é possível iniciar jobs de forma que as tarefas usem apenas um núcleo físico por vez, em vez de partilhar o núcleo físico entre mais de uma (geralmente 2) tarefas. Como resultado, metade dos núcleos do processador aparecerão como não usados da perspectiva do sistema operacional, o que seria a intenção de não usar hyperthreading.

Para fazer isso, basta usar a seguinte opção no comando **`srun: --hint = nomultithread.`** Observe que o nome é altamente enganoso, pois esta é essencialmente uma opção *nohyperthreading*, isso não significa que o multithreading está desabilitado no nível do processador. (Benchmarks mostraram que o efeito desta configuração em aplicações HPC para, por exemplo, CFD como Fluent é o mesmo, ou ligeiramente melhor do que executar no mesmo hardware com hyperthreading desativado.)

A seguir está um exemplo de um aplicação que deseja executar 40 tarefas sem hyperthreading:

```
#!/bin/bash

#SBATCH --job-name nohyperthreading-test
#SBATCH --partition inf-short
#SBATCH --time 01:00:00
#SBATCH --ntasks 40
#SBATCH --ntasks-per-node 20

srun -n $SLURM_NTASKS --hint=nomultithread ./a.out
```

Neste exemplo, Slurm calcula automaticamente quantos nós serão necessários com base no número de tarefas solicitadas, o que pode ser mais fácil de raciocinar ao iniciar os trabalhos.
Observe que, quando se trata de alocação de recursos, Slurm considera apenas as linhas #SBATCH e não olha para nenhuma outra linha, incluindo as opções de srun, portanto, precisamos de fornecer informações suficientes para indicar quantos núcleos de CPU e número de nós que iremos necessitar para o seguinte srun.
Isso é obtido usando `--ntasks` e `--ntasks-per-node`, mas também pode usar-se `–ntasks` junto com `--nodes` para ser explícito.

Se deixássemos a opção `--ntasks-per-node` 20 de fora, Slurm não saberia sobre a opção `--hint = nomultithread` durante a alocação de recursos (`--hint = nomultithread` às vezes faz coisas fragmentadas quando usado numa linha

#SBATCH), e pensaria que um único nó de 40 núcleos seria capaz de atender a essas 40 tarefas.
Portanto, ele alocaria apenas um nó. Isso seria muito mau, pois no seguinte srun, a dica nomultithread seria respeitada, mas seria tarde demais para usar 2 nós, pois a alocação de recursos já aconteceu, resultando em 40 tarefas sendo executadas em apenas 20 núcleos, o que é obviamente mau para desempenho e certamente não o que pretendíamos.
Portanto, a opção `--ntasks-per-node` 20 é necessária como uma dica nas linhas #SBATCH para que Slurm aloque os 2 nós de que precisamos.
Além disso, imagine-se que queremos executar um aplicação com 12 tarefas, cada tarefa executando 2 threads OpenMP, e queremos beneficiar da desativação do hyperthreading. O exemplo a seguir mostra um exemplo de como fazer isso (para infpartições).

```
#!/bin/bash

#SBATCH --job-name nohyperthreading-openmp-test
#SBATCH --partition inf-short
#SBATCH --time 01:00:00
#SBATCH --exclusive
#SBATCH --ntasks 12
#SBATCH --ntasks-per-node 10
#SBATCH --cpus-per-task 2

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun -n $SLURM_NTASKS --hint=nomultithread ./a.out
```

Ter em consideração que temos que usar `--exclusive` para que a alocação ocupe todo o nó. As opções restantes em `--exclusive` serão "herdadas" por srun.

A opção `--ntasks-per-node = 10` existe para garantir que obtenhamos alocações de nós suficientes para poder executar todos os 12 * 2 threads OpenMP, cada um em seu próprio núcleo físico (sem uso de hyperthreading). No entanto, para desativar o hyperthreading, apenas 20 núcleos podem ser usados no total por nó e, além disso, queremos que cada tarefa ocupe 2 núcleos físicos. Portanto, o valor de 10: ((number_of_hyperthreaded_cores_per_node / 2) / cpus_per_task), que seria avaliado como ((40/2) / 2) = 10.
Pode-se também simplesmente fazer as contas e fornecer o valor apropriado para `--nodes` em vez de usar `--ntasks-per-node`.

Para os nós das partições batch *, o equivalente seria válido, exceto os nós com 32 hyperthreaded