Tutorial Slurm

1. Introdução

Slurm é o software responsável pelo escalonamento de *jobs* no cluster. Ele provê diversos comandos úteis para execução, manutenção e monitoramento destes *jobs*, objetivando tornar a execução de programas de forma paralela em um cluster quase tão simples quanto em um computador pessoal. Este tutorial tem como objetivo listar e explicar a utilização dos comandos mais convenientes aos usuários do cluster.

2. Termos técnicos

- **Cluster:** Um cluster(do inglês: 'aglomerado') é um grupo de computadores conectados, geralmente por uma rede local, trabalhando de forma conjunta visando agir como um único supercomputador.
- **Job:** Job(do inglês: 'trabalho') no Slurm representa um conjunto de um ou mais programas solicitado para execução no cluster.
- **Partition:** Partition(do inglês: 'partição') no Slurm representa um conjunto contendo um ou mais *nodes.*
- **Node:** Node(do inglês: 'nó') no Slurm representa uma máquina física com suas especificações técnicas.

3. Submetendo um job

O exemplo a seguir especifica uma partição, um tempo limite e o número de núcleos desejados. O script a seguir executa uma tarefa bem simples - Gera um arquivo contendo números aleatórios e em seguida os ordena.

4. Paralelizando com *job arrays*

Job arrays são uma ferramenta muito útil do Slurm para **executar e gerir múltiplos *jobs* similares** de forma simples e rápida. É especialmente útil quando se deseja executar o **mesmo experimento múltiplas vezes.**

O exemplo abaixo demonstra a utilização dos *job arrays.* O primeiro código é um script em shell para **configurar e executar** os *job arrays*. O segundo é um programa em python para representar um único experimento a ser executado várias vezes. Neste exemplo, o experimento recebe como parâmetro o seu índice de execução dentro do array, e calcula uma lista de oito números aleatórios. Ao final, imprime na tela a lista. A explicação dos códigos linha-a-linha vem logo a seguir.

- jobArray.sh

```
#!/bin/bash

#SBATCH --array=1-10
#SBATCH --output=jobArray_%A_%a.out
#SBATCH --error=jobArray_%A_%a.err
#SBATCH --ntasks=1
#SBATCH --partition=large

##########################
# Tarefas dos seus jobs #
##########################

# Linha de execução do seu experimento
python pythonNumbers.py $SLURM_ARRAY_TASK_ID
```

- randomNumbers.py

```
import sys
import random

if len(sys.argv) < 2:
    print 'Falta um parametro'
    quit()

print 'Simulation[', str(sys.argv[1]), '] = ', [random.random() for
r in range(8)]

```

- Linha-a-linha: jobArray.sh

```
#!/bin/bash
```

Define o caminho do interpretador de comandos desejados. Majoritariamente utilizada da forma acima.

```
#SBATCH --array=1-10
```

Define que o experimento será executado **10** vezes, com seus identificadores de *array* variando de **1** até **10**.

```
#SBATCH --output=jobArray_%A_%a.out
#SBATCH --error=jobArray_%A_%a.err
```

Define os nomes desejados para os arquivos onde se deseja salvar as mensagens de saída e as mensagens de erro, respectivamente. Perceba no exemplo a utilização dos parâmetros **%A** e **%a**. Parâmetros são demarcações que serão substituídas durante a execução pelos valores correspondentes a eles, por exemplo, o parâmetro **%A** representa o ID que o *slurm* vai definir para o seu *job*, e o **%a** define o índice atual de execução dentro do *job array*. Como resultado, ao salvar os arquivos de uma execução com o ID de exemplo de **675432**, os seus nomes irão variar de jobArray**_675432_1**.out e jobArray**_675432_1**.err até jobArray**_675432_10**.out e jobArray**_675432_10**.err

```
#SBATCH --partition=large
```

Define em qual partição do slurm o experimento será executado. Para a lista de partições presentes no cluster, veja o comando **sinfo**. Para informações sobre especificações técnicas da partição, veja o Manual de Usuário.

```
python pythonNumbers.py $SLURM_ARRAY_TASK_ID
```

Esta é a linha que chama o seu experimento. Neste exemplo, utilizamos um código em python. Nesta linha temos também o uso de um parâmetro, o `$SLURM_ARRAY_TASK_ID`, que assume a mesma função que o parâmetro **%a** citado acima, passando para o script em Python o índice de execução.

- Linha-a-linha: randomNumbers.py

```
import sys
import random
```

Importa o pacote **sys** para fazer a verificação dos parâmetros recebidos e o pacote **random** para gerar sequências de números aleatórios.

```
if len(sys.argv) < 2:
    print 'Falta um parametro'
    quit()
```

Verifica se o programa recebeu o parâmetro. Neste caso, o Python sempre envia ao programa o seu nome de arquivo como primeiro parâmetro, então o `$SLURM_ARRAY_TASK_ID` que enviamos seria o segundo na lista de argumentos.

```
print 'Simulation[', str(sys.argv[1]), '] = ', [random.random() for r in
range(8)]
```

Gera a lista de oito números aleatórios e a imprime após imprimir o índice do experimento atual.

- Para executar:

```
sbatch jobArray.sh
```

- Saída do script:
  Para o exemplo acima, supondo um id de *job* “685895” serão gerados os seguintes arquivos:

```
jobArray_685895_10.err  jobArray_685895_10.out
jobArray_685895_1.err   jobArray_685895_1.out
jobArray_685895_2.err   jobArray_685895_2.out
jobArray_685895_3.err   jobArray_685895_3.out
jobArray_685895_4.err   jobArray_685895_4.out
jobArray_685895_5.err   jobArray_685895_5.out
jobArray_685895_6.err   jobArray_685895_6.out
jobArray_685895_7.err   jobArray_685895_7.out
jobArray_685895_8.err   jobArray_685895_8.out
jobArray_685895_9.err   jobArray_685895_9.out
```

- Conteúdo dos arquivos:

O comando abaixo vai listar o conteúdo de todos os arquivos que tenham o nome começando com **jobArray_** e terminando com **.out**, independentemente do que houver entre isso. Neste caso, lista o conteúdo dos 10 arquivos de saída que foram gerados.

```
$ cat jobArray_*.out
Simulation[ 10 ] =  [0.086531238989275172, 0.64260512158890559, 0.76837905388370531,
0.88341687006218472, 0.085681481000309367, 0.73050784375957289, 0.55651209210432129,
0.97299859562874702]
Simulation[ 1 ] =  [0.96762282752288786, 0.43204663952767108, 0.33786144163551302,
0.70611171780899418, 0.87538939827420825, 0.9645575855706916, 0.90578117491858856,
0.23382005956966889]
Simulation[ 2 ] =  [0.80478470791158818, 0.8657466406117903, 0.7994896026824736,
0.49100520516023038, 0.0016665135920682639, 0.68014145261600856, 0.77814329331659571,
0.2513894628655593]
Simulation[ 3 ] =  [0.50262154841006201, 0.61313187206453668, 0.5831380338171529,
0.054309512055682019, 0.24362312622233284, 0.80784667240079311, 0.85863371512764286,
0.25858828200669337]
Simulation[ 4 ] =  [0.90104725041866474, 0.77316594849911158, 0.47853147675421903,
0.62421250940078865, 0.51509894189016592, 0.46841646021680605, 0.45086601219776212,
0.23117641419818624]
Simulation[ 5 ] =  [0.63328875071170099, 0.029357301283833515, 0.76414340126293234,
0.44807419136916604, 0.17850695059190136, 0.84229529714406437, 0.14494952841473407,
0.30131890684787421]
Simulation[ 6 ] =  [0.92960950668875075, 0.92727654802407888, 0.95165762529430542,
0.42697458299911517, 0.93325128588188577, 0.69418119113736898, 0.047325390817721358,
0.95960506186069194]
Simulation[ 7 ] =  [0.14062096070728525, 0.075954162085946297, 0.69284382076254625,
0.28381097500498453, 0.47213116473877292, 0.38612229167157996, 0.7575166832956296,
0.64047410838133234]
Simulation[ 8 ] =  [0.42708962528611671, 0.78390853971860763, 0.2488085210091403,
0.49483147600098243, 0.75406493871335478, 0.12711645953592443, 0.31971328748093997,
0.54628705987254977]
Simulation[ 9 ] =  [0.51179710699890779, 0.9620481165555117, 0.23983936105423054,
0.8837894283419937, 0.20167120589218102, 0.99039374683462944, 0.21564528497103796,
0.29467248362859633]
```

Analogamente, o comando `$ cat jobArray_*.err` lista o conteúdo dos arquivos de erro gerados. Como não houve erro em nenhum dos scripts, não listará nada.

5. Lista de comandos

5.1 Submissão de Jobs

a. **salloc:** Obtém uma alocação de *job* (um grupo de *nodes*).
b. **sbatch:** Submete um script para ser executado**.**
c. **srun:** Obtém uma alocação de *job* e os executa de forma paralela.

- Argumentos:

Opção	Função
--array=<indices> (ex. “--array=1-10”)	Especifica um vetor de *jobs.* (Veja a Seção 4. Somente comando **sbatch**)
--begin=<tempo> (ex. “--begin=18:00:00”)	Especifica o momento desejado para início do job.
--cpu-per-task=<quantidade> (ex. “--cpu-per-task=10”)	Define número mínimo de CPUs solicitadas por tarefa.
--dependency=<estado:jobid>	Aguarda as dependências serem satisfeitas antes de iniciar o *job*.
--error=<arquivo>	Define o arquivo para salvar as mensagens de erro.
--exclude=<nodes>	Especifica *nodes* para remover da lista de alocação.
--export<nome[=valor]>	Exporta variáveis de ambientes especificadas para serem usadas pelos jobs.
--gres	Utilizado para especificar recursos genéricos que serão usados pelo *job*, por exemplo, GPUs.
--input<arquivo>	Define o arquivo de entrada do script.
--job-name=<name>	Define o nome do *job*.
-n<count>	Número de tarefas a serem executadas.
--nodelist	Especifica uma lista de *nodes* para incluir na alocação do *job*.
--output<arquivo>	Define o arquivo para salvar a saída do script.
--partition=<nomes>	Partição/Fila onde executar o *job*.
--qos=<nome>	Define a Qualidade do Serviço. Mais em Manual do Usuário: QoS
--time=<time>	Define um tempo limite de execução do *job*. Formatos aceitos incluem "minutos", "minutos:segundos", entre outros.

5.2 Gestão de jobs

a. scancel:
   Cancela um job em execução.
   Uso: scancel jobid

b. squeue2
   Lista informações de jobs executados pelo usuário atual.

6. Referências

SchedMD -Slurm Support and Development

https://slurm.schedmd.com/pdfs/summary.pdf

Job array:
https://rcc.uchicago.edu/docs/running-jobs/array/index.html