Manual `sci-com`

Vídeo de explicação pelo Rodrigo Amorim que configurou o scicom.

A Máquina

Inicialmente, o sci-com conta com um head node (AMD Threadripper 5965WX 24c 128GB) e 14 compute nodes. Há um storage HDD de 64TB (logo 96TB) e um scratch local SSD de 1TB em cada nó. O sistema operacional é Rocky Linux e o sistema de gerenciamento de fila é SLURM. A conexão de rede é 1Gbps (com previsão de upgrade a 10Gbps) e há um nobreak de 11KVA e outro de 6KVA.

Agradecimento

Lembrem-se de agradecer em artigos, pôsteres etc o uso do sci-com. Segue uma sugestão:

The authors would like to acknowledge the use of the computational resources provided by the Sci-Com Lab of the Department of Physics at UFES, which was funded by FAPES (Brazil, TO 1081/2022).

Contas

No começo, só terá conta de professores que compartilharão o acesso com seus alunos. A ideia é que cada aluno use uma pasta dentro da pasta do professor.

Acesso

Da UFES:

```
ssh conta_scicom@172.20.76.10
```

botar senha ou configurar chave ssh para entrar sem senha de um PC de confiança

De fora da UFES:

```
ssh -J conta_ufes@sshgate.ufes.br:22222 conta_scicom@172.20.76.10 -p 22
```

ou usando o VPN da UFES.

Para usar o sshgate.ufes.br precisa botar a chave ssh na sua conta ufes em https://git.ufes.br/

Transferência de dados

Para transferir dados do seu computador para o scicom é conveniente usar rsync:

```
rsync -avh --progress --partial --inplace $VSOUR $VDEST >$VOUT 2>$VERR
```

onde:

```
VOUT=test.out
VERR=test.err
```

```
VSOUR=test
VDEST=conta_scicom@172.20.76.131:/home/conta_scicom/fulano/
```

Vejam mais opções no final deste documento.

Bibliotecas

Há bibliotecas já prontas para as arquitetura Zen, Zen2 e Zen3. Fazer `module avail` para ver as bibliotecas e ‘module load’ para carregá-las.

SLURM

Dentro da pasta home de cada prof há uma pasta `script` com arquivos de submissão SLURM de exemplo. Para submeter uma conta, basta copiar e modificar esses scripts.

Há quatro grupos de filas.

Zen3, 7 nodes, **threadripper 5975WX 32c 128GB**:

- thread-long: 3d
- thread-week: 14d (só para dois nós)

Zen3, 3 nodes, **Ryzen 9 5950X 16c 64GB**:

- ryzen9-long: 3d

Zen1, 3 nodes, **threadripper 2990WX 32c 128GB**:

- zen1-long (temporaneamente thread-long-z1): 3d

Zen2, 1 nodes (ainda não disponível), **threadripper 3970X 32c 128GB + GPU Nvidia Rtx A6000 48GB**:

- zen1gpu-long: 3d

Exemplo:

```
#!/bin/bash
#SBATCH --job-name=valerio-test                 #Nome job com o nome do aluno/prof
#SBATCH --nodes=1                               #Numero de Nós
#SBATCH --ntasks=1                              #Numero total de tarefas MPI/OPENMP
#SBATCH --partition thread-short                #Fila (partition) a ser utilizada
#SBATCH --output %x-slurm_job-%j.out
#SBATCH --error  %x-slurm_job-%j.err

#opcional
#SBATCH --time 00:05:00                         #hh:mm:ss
##SBATCH --exclusive                            #para usar um nó exclusivamente

echo $SLURM_SUBMIT_DIR
cd $SLURM_SUBMIT_DIR

# ...standard slurm stuff...
```

```
# transferir tudo pro scratch do nó e depois voltar para pasta local
scratch_dir='/local-scratch/'${SLURM_JOB_NAME}

mkdir -p $scratch_dir
cp -r * $scratch_dir
cd $scratch_dir
rm *slurm_job*

#mpirun -np $SLURM_NTASKS “exe”
touch result.txt
srun sleep 60

home_dir=${SLURM_SUBMIT_DIR}/${SLURM_JOB_NAME}
mkdir -p $home_dir
mv ${scratch_dir}/* $home_dir

echo 'done!'
```

Pode ser útil redefinir o comando squeue:

```
alias squeue='squeue --iterate=3 --format="%.8i %20P %20j %10u %8T %.10M %9D %20R"'
```

```
Thu Jul 06 16:59:24 2023
   JOBID PARTITION             NAME                USER       STATE          TIME NODES     NODELIST(REASON)
     250 thread-long           7deg_cc.sh          fernando   RUNNING  2-08:12:18 1         node4
     291 thread-long           cc-trip             fernando   RUNNING  1-09:12:30 1         node1
     343 thread-long           hex-p-t             wanderla   RUNNING     1:53:52 1         node5
     344 thread-long           gcar-FLAVIA         wanderla   RUNNING     1:53:52 1         node4
     350 thread-long           7deg_cc.sh          fernando   RUNNING     1:31:52 1         node4
     326 thread-short          frankNO22-relax     wendel     RUNNING    18:52:45 1         node2
     328 thread-short          frankNO3-relax      wendel     RUNNING    17:38:02 1         node6
     348 thread-short          frankNO-relax       wendel     RUNNING     3:26:29 1         node6
     368 thread-short          valerio-test        valerio    RUNNING        0:07 1         node7
```

Lembrar que o scratch será esvaziado a cada 15 dias.

Introdução ao SLURM

SLURM é uma ferramenta de código aberto para gerenciar e agendar trabalhos em clusters de computadores. Ele permite que você use e compartilhe recursos computacionais de maneira eficiente. O SLURM é altamente flexível e pode ser configurado para gerenciar de algumas a muitas milhares de máquinas.

Comandos básicos

- `sinfo`: fornece informações sobre os nós do cluster SLURM. Mostra quais nós estão disponíveis, quais estão sendo usados e qual é o seu estado.
- `squeue`: mostra todos os trabalhos atualmente na fila.

- `sbatch`: usado para submeter um trabalho para o cluster. Você precisa escrever um script de shell que inclui comandos SLURM e o comando que deseja executar.
- `scancel`: permite que você cancele um trabalho que submeteu.
- `srun`: permite executar comandos interativamente no nó. Você pode usá-lo para desenvolvimento e depuração.
- `scontrol show jobid #NUMERO-DO-JOB -dd`: exibe informações detalhadas sobre um job.

Submetendo um trabalho

Um trabalho é submetido ao SLURM usando o comando `sbatch`. A submissão de um trabalho requer um script de shell que contém os comandos que você deseja que o SLURM execute. O script também inclui diretivas para o SLURM, que são comentários especiais que começam com `#SBATCH`.

Aqui está um exemplo de um script de trabalho:

```bash
#!/bin/bash
#SBATCH --job-name=test_job
#SBATCH --output=result.txt
#SBATCH --ntasks=1
#SBATCH --time=10:00
srun hostname
srun sleep 60
```

Neste script:

- `--job-name` é o nome que você dá ao seu trabalho.
- `--output` é o arquivo onde a saída do trabalho será escrita.
- `--ntasks` é o número de tarefas que seu trabalho irá executar.
- `--time` é a quantidade de tempo que você espera que seu trabalho leve.
- `srun` é usado para executar comandos no nó.

Depois que seu script estiver pronto, você pode submetê-lo com `sbatch nome_do_seu_script.sh`.

Cancelando um trabalho

Para cancelar um trabalho, use o comando `scancel` seguido pelo ID do trabalho. Você pode encontrar o ID do trabalho com o comando `squeue`.

Manual para o comando sbatch: https://slurm.schedmd.com/sbatch.html

Introdução a rsync options

`rsync -PRavzhHS`

- `-a` (or `-archive`): Preserves file attributes, permissions, timestamps, and symbolic links.
- `-v` (or `-verbose`): Provides verbose output.
- `-z` (or `-compress`): Enables compression during the transfer, reducing network bandwidth usage.

Those 3 above seems like the main flags to use.

- `-h` (ou `-human-readable`): Displays the output in a human-readable format, using units like kilobytes (KB), megabytes (MB), etc.
- `-P` combination of `-progress` and `-partial`
    - `-progress`: Displays the progress of the transfer, keeping you informed about the current status.
    - `-partial`: Allows partial file transfers, enabling resumption of interrupted transfers.
- `-R` ensures that the entire directory hierarchy is replicated on the destination
    - It's worth noting that if you only need to sync the contents of a directory without preserving the entire path structure, using `-r` is usually sufficient instead of `-R`
- `-S`: Using this flag is particularly beneficial when syncing files that contain **sparse** sections, such as virtual machine disk images or log files with large blank spaces. It ensures that only the actual data is transferred, improving the efficiency of the synchronization operation.
- `-H`: If the source has files with hard links, to preserve those hard links on the destination or/and to save disk space on the destination by using hard links instead of duplicating data. Don’t use it if the source data do not contain hard links or the destination system do not support hard links.
- `–inplace`: When `rsync` transfers a file to the destination, it creates a temporary file with a different name and then renames it to the original filename once the transfer is complete. However, with `--inplace`, `rsync` updates the destination file in-place without creating a temporary file. This can save disk space, so if disk space is a concern use it. The same thing can also increase the speed, so if speed is a concern, use. Because with this flag `rsync` updates the destination directory without creating temporary file, if the transfer process gets interrupted, the destination file may be left in an inconsistent state. In such cases, you may end up with a partially updated file on the destination.
- `--exclude=".*”`: to not transfer the hidden files.