Checkpoint SLURM jobs

Introduction

As you probably noticed, execution time for jobs in DCSR clusters is limited to 3 days. For those jobs that take more than 3 days and cannot be optimized or divided up into smaller jobs, DCSR's clusters provide a Checkpoint mechanism. This mechanism will save the state of application in disk, resubmit the same job, and restore the state of the application from the point at which it was stopped. The checkpoint mechanism is based on CRIU which uses low level operating system mechanisms, so in theory it should work for most of the applications.

How to use it

First, you need to do the following modifications to your job script:

1. You need to source the script */dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh*
2. Use *launch_app* to call your application
3. (optional) add --error and --output to slurm parameters. This will create two separate files for standard output and standard error. If you need to process the output of your application, you are encouraged to add these parameters, otherwise you will see some errors or warnings from the checkpoint mechanism. If your application generates custom output files, you do not need these options.
4. make sure to change requested time by 12h

The script below summarizes those changes:

```
#!/bin/sh
#SBATCH --job-name job1
#SBATCH --cpus-per-task 4
#SBATCH --partition cpu
#SBATCH --time 12:00:00
#SBATCH --mem=16G
#SBATCH --error job1-%j.error
#SBATCH --output job1-%j.out

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh
```

```
launch_app $APP
```

> the --time parameter does not limit the duration of the job but It will be used to create the checkpoint. For example for a --time 12:00:00 , after 12 hours the job will be checkpointed and it will be rescheduled some minutes later. The checkpoint uses low level Operating System mechanism so it should work for most of applications, however, there coud be some error with some exotic applications. That is why it is good to check the job after the first checkpointing (12 hours), so as to know if the application is compatible with checkpointing.

Launching the job

```
Sbatch job.sh
```

Make sure to use `Sbatch` and not `sbatch`. Additionally to the out and error files produced by SLURM, the execution of the job will generate:

1. checkpoint-JOB_ID.log: checkpoint log
2. checkpoint-JOB_ID: application checkpoint files. Please do not delete this directory until your job has finished otherwise the job will fail.

> Make sure not to use the option `#SBATCH --export NONE` in your jobs

Job examples:

```
#!/bin/sh
#SBATCH --job-name job1
#SBATCH --cpus-per-task 1
#SBATCH --partition cpu
#SBATCH --time 12:00:00
#SBATCH --mem=16G

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

launch_app ../pi_css5 400000000
```

Tensorflow:

```
#!/bin/sh
#SBATCH --job-name job1
#SBATCH --cpus-per-task 4
#SBATCH --partition cpu
#SBATCH --time 12:00:00
#SBATCH --mem=16G

export OMP_NUM_THREADS=4
source ../tensorflow_env/bin/activate

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

launch_app python run_tensorflow.py
```

Samtools:

```
#!/bin/sh
#SBATCH --job-name job1
#SBATCH --cpus-per-task 1
#SBATCH --partition cpu
#SBATCH --time 12:00:00
#SBATCH --mem=16G

module load gcc samtools

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

launch_app samtools sort
/users/user1/samtools/HG00154.mapped.ILLUMINA.bwa.GBR.low_coverage.20101123.bam -o
sorted_file.bam
```

Complex job scripts

If your job script look like this:

```
#!/bin/sh
#SBATCH --job-name job1
#SBATCH --cpus-per-task 1
```

```
#SBATCH --partition cpu
#SBATCH --time 12:00:00
#SBATCH --mem=16G

module load gcc samtools

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

command_1
command_2
command_3
command_4
launch_app command_n
```

Only the *command_n* will be checkpointed. The rest of the commands will be executed each time the job is restored. This can be a problem in the following cases:

1. command_1, command_2 ... take a considerable amount time to execute
2. command_1, command_2 generate input for command_n. This will make the checkpoint fail if the input file differs in size

For those cases, we suggest to wrap all those commands inside a shell script and checkpoint the given shell script.

```
command_1
command_2
command_3
command_4
command_n
```

and make the script executable:

```
chmod +x ./script.sh
```

job example:

```
#!/bin/sh
#SBATCH --job-name job1
#SBATCH --cpus-per-task 1
#SBATCH --partition cpu
```

```
#SBATCH --time 12:00:00
#SBATCH --mem=16G

module load gcc samtools

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

launch_app ./script.sh
```

> Make sur to not redirect standard output on your commands, for example, `command > file`. If you want to do this, you have to put the command in a different script

Custom location for log and checkpoint files

If you want checkpoints logs and files to be located in a different directory, you can use the following variable:

```
export CKPT_DIR='ckpt-files'
```

Be sure to define it either in your shell before submitting the job or in the job script before loading `ckpt_methods.sh` script. Here below, an example:

```
#!/bin/sh
#SBATCH --job-name ckpt-test
#SBATCH --cpus-per-task 1
#SBATCH --time 00:05:00


module load python
export CKPT_DIR='ckpt-files'
source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh
launch_app python app.py
```

Email notifications

If you use the options `--mail-user` and `--mail-type` on your job you could receive a lot of notifications. The job will be go thoroughout the normal job cycle start and end several times. So, you will end up with a lot of notification which depends on the walltime of your job.

You can reduce this notifications with:

```
  --mail-type END,FAIL
```

Resume a failed checkpoint

If for whatever reason the jobs stoped, you can try to reuse the checkpoint created by adding the following variable at your job script:

```
CKPT_FILES=checkpoint-55214091
```

You have to change the previous value by the name of the checkpoint directory created by the failed job. Example of the whole job script:

```
#!/bin/sh
#SBATCH --job-name ckpt-test
#SBATCH --cpus-per-task 1
#SBATCH --time 00:05:00

export CKPT_FILES=checkpoint-55214091
source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh
module load python
launch_app python app.py
```

> The slurm output file will be the one that belongs to the previous failed job. The new job will have an empty slurm output.

Applications based on r-light module

`r-light` module provides R and Rscript commands in different versions using a container. If your job depends on this module you should replace `Rscript` by the whole singularity command. Let's suppose you have the following script:

```
#!/bin/bash -l
#SBATCH --job-name r-job
#SBATCH --cpus-per-task 1
#SBATCH --time 00:05:00

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

module load r-light
launch_app Rscript test.R
```

In order to know the whole singularity command, you need to type `which Rscript`, which will produce the following output:

```
Rscript ()
{
    singularity exec /dcsrsoft/singularity/containers/r-light.sif /opt/R-4.4.1/bin/Rscript
"$@"
}
```

You copy paste that into your job like this:

```
#!/bin/bash
#SBATCH --job-name ckpt-test
#SBATCH --cpus-per-task 1
#SBATCH --time 00:05:00

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

module load r-light
launch_app singularity exec /dcsrsoft/singularity/containers/r-light.sif /opt/R-
4.4.1/bin/Rscript test.R
```

Java applications

In order to checkpoint java applications, we have to use two parameters for launching the application:

```
-XX:-UsePerfData
```

This will deactivate the creation of the directory */tmp/hsperfdata_$USER,* otherwise it will make the checkpoint restoration fail

```
-XX:+UseSerialGC
```

This will enable the Serial Garbage collector which deactivates the parallel garbage collector. The parallel garbage collector generates a GC thread per thread of computation. Thus, making the restoration of checkpoint more difficult due to the large number of threads.

Snakemake

version => 8

You need to install a new SLURM plugin that we develop to support checkpoint:

```
pip install git+https://git.dcsr.unil.ch/Scientific-Computing/snakemake-executor-plugin-slurm
```

Then you need to activate it and choose a checkpoint frequency in secs, something like:

```
snakemake --jobs 2 --executor slurm --slurm-checkpoint
```

The checkpoint will be done 30 minutes before the job ends and the job will be requeued for execution.

(Deprecated) versions < 8

In order to use the checkpoint mechanism with snakemake, you need to adapt the SLURM profile used to submit jobs into the cluster. Normally the SLURM profile defines the following options:

- cluster: slurm-submit.py (This script is used to send jobs to SLURM)
- cluster-status: "slurm-status.py" (This script is used to parse jobs status from slurm)
- jobscript: "slum-jobscript.sh" (Template used for submitting snakemake commands as job scripts)

We need to modify how jobs are launched to slurm, the idea is to wrap snakemake jobscript into another job. This will enable us to checkpoint all processes launched by snakemake.

The procedure consists of the following steps (the following steps are based on the slurm plugin provided here: *https://github.com/Snakemake-Profiles/slurm*)

Create checkpoint script

Please create the following script and call it *job-checkpoint.sh*:

```
#!/bin/bash

source /dcsrsoft/spack/external/ckptslurmjob/scripts/ckpt_methods.sh

launch_app $1
```

make it executable: chmod +x *job-checkpoint.sh.* This script should be placed at the same directory as the other slurm scripts used.

Modify slurm-scripts

We need to modify the sbatch command used. Normally a jobscript is passed as a parameter, we need to pass our aforementioned script first and pass the snakemake jobscript as a parameter, as shown below (lines 6 and 9):

```
def submit_job(jobscript, **sbatch_options):
    """Submit jobscript and return jobid."""
    options = format_sbatch_options(**sbatch_options)
    try:
        # path of our checkpoint script
        jobscript_ckpt = os.path.join(os.path.dirname(__file__),'job-checkpoint.sh')
        # we tell sbatch to execute the chekcpoint script first and we pass
        # jobscript as a parameter
        cmd = ["sbatch"] + ["--parsable"] + options + [jobscript_ckpt] + [jobscript]
        res = sp.check_output(cmd)
    except sp.CalledProcessError as e:
        raise e
    # Get jobid
    res = res.decode()
    try:
        jobid = re.search(r"(\d+)", res).group(1)
    except Exception as e:
        raise e
    return jobid
```

Ideally, we need to pass extra options to sbatch in order to control output and error files:

```
sbatch_options = { "output" : "{rule}_%j.out", "error" : "{rule}_%j.error"}
```

This is necessary to isolate errors and warnings raised by the checkpoint mechanism into an error file (as explained at the beginning of this page). This is only valid for the official slurm profile as it will treat snakemake wildcards defined in Snakefile (e.g rule).

Export necessary variables

You still need to export some variables before launching snakemake:

```
export SBATCH_OPEN_MODE="append"
export SBATCH_SIGNAL=B:USR1@1800
snakemake --profile slurm-chk/ --verbose
```

With this configuration, the checkpoint will start 30 min before the end of the job.

Limitations

- It does not work for MPI and GPU applications
- The application launched should be composed of only one command with its arguments. If you need complex workflows wrapt the code inside a script.

Créé 2022-07-11 09:45:13 UTC par Cristian Ruiz
Mis à jour 2026-10-28 07:43:05 UTC par Margot Sirdey