META: running a large number of jobs conveniently

Sergey Mashchenko
SHARCNET / Compute Canada

October 6, 2021

Overview

- Usage scenario
- Algorithm
- Main features
- Commands
- Installation
- Live demo

Scenario

- You need to run a large (many thousands) number of jobs, e.g.
  - Soil evolution model for different patches across the globe
  - Design and optimization of a multi-element optical lens, starting from a random setup (Monte-Carlo)

Solution #1

- Simplest: submit all these jobs using a bash loop:

```
for ((i=0; i<5000; i++))
   do
   sbatch my_job_script.sh $i
   done
```

- Issues: slow, heavy load for the scheduler, ...

Solution #2

- Use Job Array feature of our scheduler, SLURM

```
$ sbatch --array=0-4999 my_job_script2.sh
$ cat my_job_script2.sh
…
i = $SLURM_ARRAY_TASK_ID
…
```

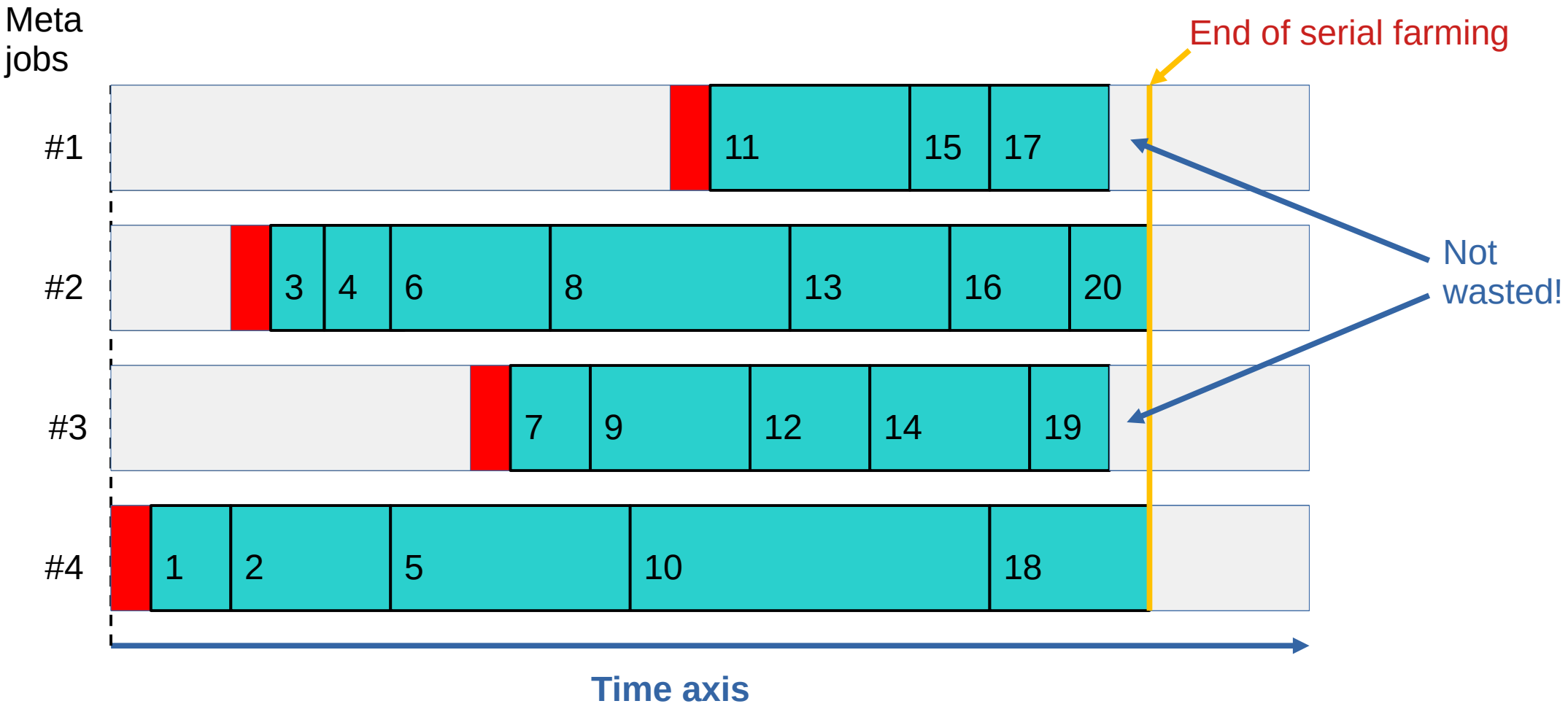
- Heavy load for the scheduler, ...

Solution #3

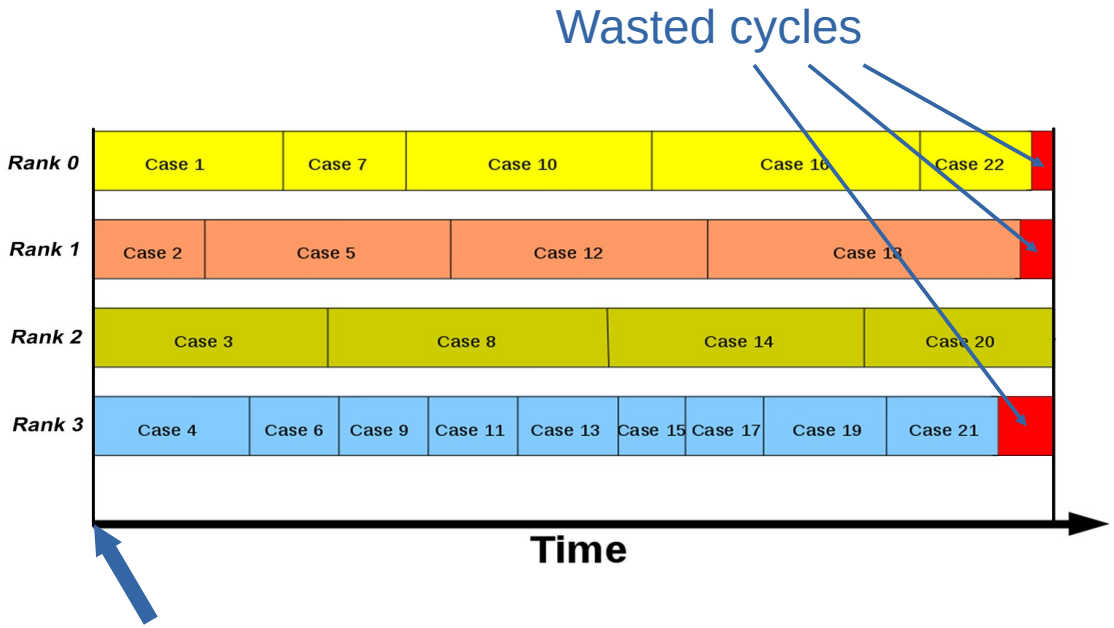
- One can also try to bundle up a bunch of serial jobs into one large parallel (MPI) job, where each rank processes one or more code executions – GNU parallel, GLOST
- Issues: queue wait time can become significantly longer, ...

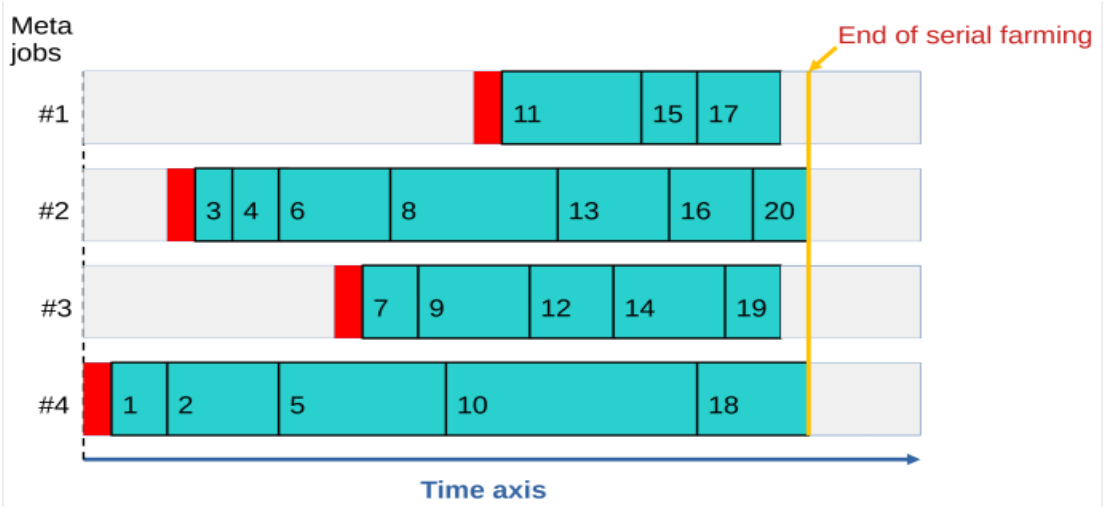
META: the optimal solution

- In META approach, you submit a bunch of “meta-jobs”, each of which would process multiple independent computations.
- Similarly to GLOST, each allocated cpu core runs multiple code instances.
- Unlike GLOST, the cpu cores are not bundled up in a large parallel job.
- So META has both
  - low overhead/scheduler load, and
  - short queue wait time.

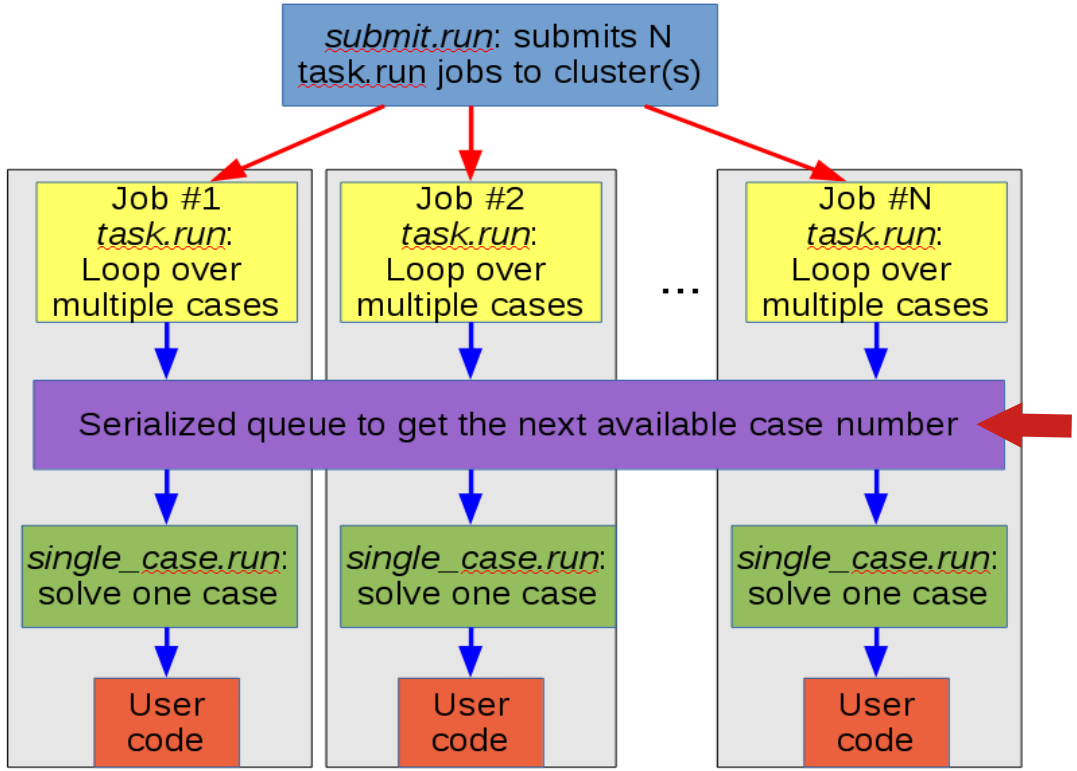

Meta jobs
End of serial farming
#1
11
15
17
#2
3
4
6
8
13
16
20
Not wasted!
#3
7
9
12
14
19
#4
1
2
5
10
18
Time axis

GLOST vs META





META algorithm



Serialized access to the file containing the current computation #, using lockfile command.

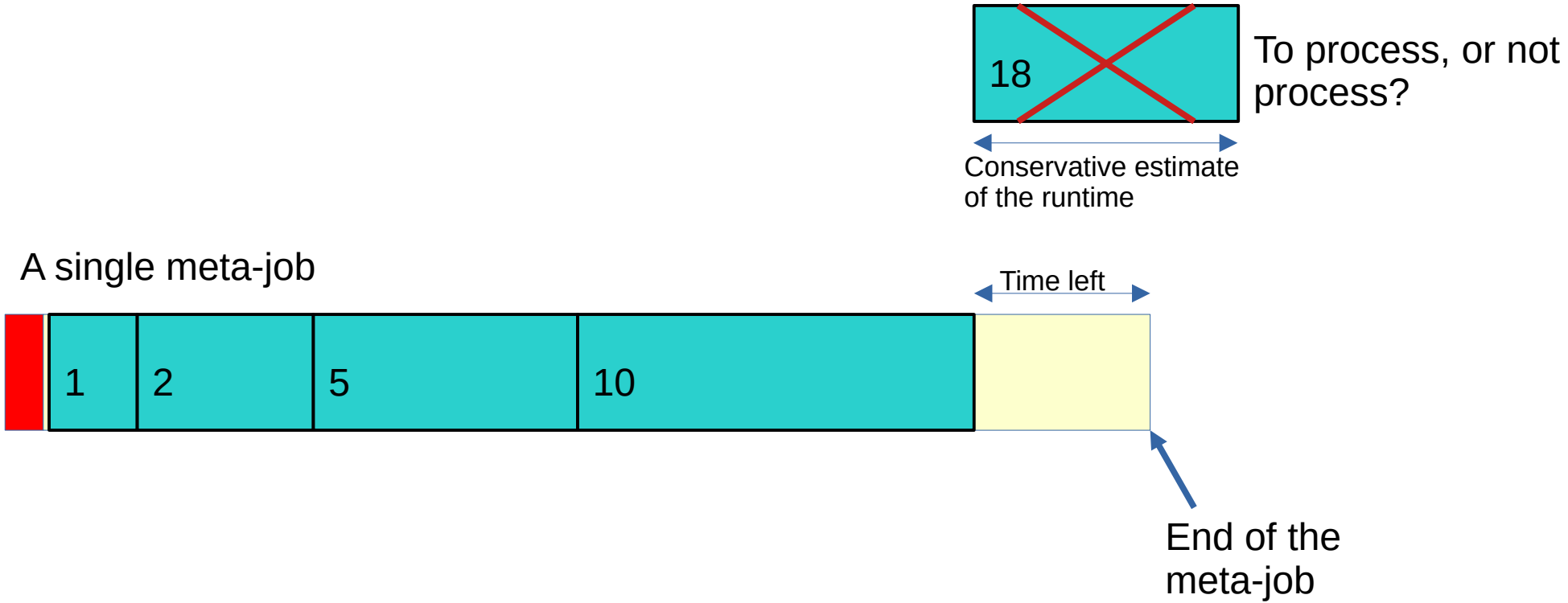
Additional META advantages

- It has a very convenient resubmit feature
  - With a single command, all computations which failed or never ran will be resubmitted as a new serial farm
- Meta-jobs constantly analyze runtimes for individual code executions, and use these data to improve the efficiency.

Improving efficiency with runtimes

- All running meta-jobs store runtimes for individual computations in a single file
- Once the list is long enough (>=8 runtimes), metajobs use the list to make a conservative (top 12.5% quantile) estimate of how long the runtime typically is.
- Metajobs use this information to determine whether they should start the next code execution, or die early.

To process or not?



Main features of META

- Two modes of operation
  - META mode
  - Simpler mode of one job for each code execution
- Convenient resubmit feature
  - For all computations which failed or never ran
- Ability to independently operate multiple serial farms on the same cluster
- The research code can be of any kind – serial, multi-threaded, MPI, GPU
- The exit status and timing are captured for each code execution
- For convenience, additional commands: list, query, kill, status, prune, clean

META commands

```
submit.run  N {optional sbatch arguments}
```

Submits the farm to the scheduler.

Here `N` is either the number of meta-jobs to use, or “`-1`” (minus one) if you want to use the simpler “one job for each computation” mode. In the latter case, the number of jobs will be equal to the number of computations to perform.

Most of sbatch arguments can be provided either in the job script file, or as an optional argument to submit.run.

META commands

```
resubmit.run  N {optional sbatch arguments}
```

Resubmits all computations which failed or never ran as a new farm. Can only be executed after the previous run is completed. You can use resubmit as many times as needed.

The arguments are the same as for submit.run. They do not have to match the arguments from the prior submit.run or resubmit.run execution.

META commands

`list.run`

Will list all the jobs with their current state for the farm.

`query.run`

Will provide a one line summary (number of queued / running / done jobs) in the farm, which is more convenient than using “list.run” when the number of jobs is large. It will also “prune” queued jobs if warranted.

META commands

`kill.run`

Will kill all the running/queued jobs in the farm.

`prune.run`

Will only kill (remove) queued jobs.

META commands

`Status.run {-f}`

(Capital “S”!) will list statuses of all processed cases. With the optional "-f" switch, the non-zero status lines (if any) will be listed at the end.

`clean.run`

Will delete all the files in the current directory (including subdirectories if any present), except for *.run scripts, job_script.sh, table.dat, and bin subdirectory.

Important notes

- META commands have to be executed inside a farm directory.
- Multiple farms can be operated at the same time, one just has to switch to the corresponding directory.
- By default, a new subdirectory is created for each independent computation. This can be changed inside single_case.sh script.

Installing META

- Login to the cluster.
- Use "git" to clone our META repository:

```
$ git clone https://git.sharcnet.ca/syam/META.git
```

- Create directory ~/bin if you don't have one:

```
$ mkdir ~/bin
```

- Copy all the files inside META/bin subdirectory to ~/bin:

```
$ cp -p META/bin/* ~/bin
```

- Add ~/bin to your $PATH variable (you can add the line below at the end of your ~/.bashrc file):

```
$ export PATH=/home/$USER/bin:$PATH
```

Installing META

- Copy your code and initial conditions files to the META directory if needed.
- Create table.dat inside the META directory
  - Text file, each line describes one independent computation
  - Multiple commands can be used, separated by “;”
  - Redirects (<, >) and pipes (|) can be used
- Modify "job_script.sh" file to suit your needs. In particular, use a correct account name, and set an appropriate job runtime.
- Modify “single_case.sh” if needed.
- To run another farm concurrently with the first one, create another directory - say, META1 - and copy there and customize the files single_case.sh and job_script.sh, and create a new table.dat file there.

Using META

- cd to the corresponding farm directory:
  `$ cd ~/META`
- Initiate the farm, say using 8 meta-jobs:
  `$ submit.run 8`
- Or you can use the simpler “one job per computation” mode:
  `$ submit.run -1`
- List the state of all the (meta-)jobs:
  `$ list.run`
- One line summary of the farm’s state:
  `$ query.run`

job_script.sh file

```
#!/bin/bash
# Here you should provide the sbatch arguments to be used in all jobs in this serial farm
# It has to contain the runtime switch (either -t or --time):
#SBATCH -t 0-00:10
#SBATCH --mem=4G
#SBATCH -A Your_account_name


# Don't change this line:
task.run
```

single_case.sh file

```
# +++++++++++++++++++++  This part can be customized:  ++++++++++++++++++++++++
#  Here:
#  $ID contains the case id from the original table (can be used to provide a unique seed to the code etc)
#  $COMM is the line corresponding to the case $ID in the original table, without the ID field
#  $SLURM_JOB_ID is the jobid for the current meta-job (convenient for creating per-job files)

mkdir -p RUN$ID
cd RUN$ID

echo "Case $ID:"

# Executing the command (a line from table.dat)
# It's allowed to use more than one shell command (separated by semi-columns) on a single line
eval "$COMM"

# Exit status of the code:
STATUS=$?

cd ..
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

table.dat file

```
~/bin/code1  1.0  10  2.1
~/bin/code1  1.5  12  2.9
cp -f /input_dir/input1 input; /code_dir/code
cp -f /input_dir/input2 input; /code_dir/code
code.exe < ../IC.1
code.exe < ../IC.2
sleep 10m
...
```

Live demo

Help resources

- Full documentation is maintained on our wiki (feel free to make corrections if needed):

  https://docs.computecanada.ca/wiki/META_package_for_serial_farming

- Submit a ticket to support@computecanada.ca (mention my name – Sergey Mashchenko).

Thank you!