Snakemake

Snakemake-node

We have a special compute node to start Snakemake pipelines in order to reduce the load on the login. Please do

```
ssh snakemake-node
```

from the login node to go there.

Snakemake profile

With Snakemake v6.4.0, we have released an optimized configuration file that ensures best compatibility of Snakemake with our cluster. The configuration profile:

- ensures, that Snakemake rules are executed as **individual jobs** on our cluster instead of inside the Snakemake main process, which allows you to
  - use combined computing power of all cluster nodes instead of just a single one, and
  - create smaller jobs with fine-grained resource requirements that will start sooner instead of one big job
- writes **error and console output** of individual rules to files inside your Snakemake directory
- handles dependencies on **software modules** on the cluster automatically for individual rules
- check the **status of jobs** across the cluster, which will recognize finished and failed rules earlier
- provides an automatic **jobscript template** so you don't have to provide one
- integrates **Singularity**, the containerization framework, into Snakemake, so you can archive optimal performance from rules executing in containers

Execution

This makes running Snakemake workflows on the cluster much easier, as you don't need to compose Snakemake's lengthy command line arguments yourself. After logging in on our cluster using SSH (and VPN if outside the campus):

1. Log in on our dedicated Snakemake node: `ssh snakemake-node`
2. `cd` to the directory containing your Snakemake workflow
3. Load the Snakemake module: `module load Snakemake/6.4.0`
4. Execute Snakemake: `snakemake --profile /software/Snakemake/hhu-profile`

That's it! No more composing lengthy Snakemake command lines. Of course, you can take a look at the profile and the defaults chosen in our universities GitLab or in `/software/Snakemake/hhu-profile` on the cluster. Any defaults can still be overridden on the command line using the appropriate command line argument. See the Snakemake reference for the full specification, including guidance on how to write your own workflows. Some examples are listed down below.

Resource configuration

To specify the resources allocated to each rule, you need to create a `cluster.yaml` file inside your Snakemake directory. As for regular jobs on the cluster, be sure to calculate your requirements as tight as possible. Otherwise resources will be wasted and unusable to other users, and jobs will take longer to start.

An example might look like this:

cluster.json

```
__default__:
project: BenchMarking
walltime: 00:00:30
mem: 1G
cpus: 1
gpus: 0
test:
walltime: 00:01:59
mem: 8G
cpus: 8
modules:
- Snakemake
```

Troubleshooting

Error: "/bin/bash: unbound variable"

When using `shell`-Blocks, Snakemake adds `set -u` (amongst others) during execution. This instruction causes bash to report uninitialized (unbound) variables, which might be an indication for a bug in your shell script. However, using uninitialized variables is often desired in bash, for example when changing `PATH` variables. To allow this, add explicitly `set +u` at the beginning in your shell block.

I am here listing information about running Snakemake on the HPC for my pipeline `spike` https://github.com/sjanssen2/spike, which performs multiple steps to analyse NGS whole exome data.

Requirements

`Snakemake` is available as a module on our cluster. You can list all available modules with

list cluster modules

```
module avail
```

(look for `Snakemake` in that list) and load the corresponding module via `module load`, e.g.:

load Snakemake module

```
module load Snakemake/5.10.0
```

If you configured additional python packages in your `Snakemake` workflow using a `conda` environment, you might need to load `miniconda` as well, e.g.:

```
module load Miniconda/3_snakemake
```

`Snakemake` greatly supports execution of a pipeline on a grid compute system with very little overhead. However, it will help reading `Snakemake`'s own documentation to get familiar with with basic concepts and commands.

Compose the `Snakemake` master command

Here, I am giving some background information about the multiple flags you should set when executing a spike pipeline via Snakemake.

Arguments

--cluster-config cluster.json

Resources on shared computer clusters are not infinite and the job of the admins is to ensure that multiple users get their fair share. This is only possible if each user behaves nicely, i.e. spends some efforts to find small upper boundaries for their applications in terms of memory, CPUs/cores and execution time. By nature, you are greedy, but the requirements for smaller jobs are easier to meet, so the smaller the job is, the earlier it will be executed. This parameter allows you to define your resource requirements in a dedicated file instead of the main rule file. You can also use the parameters in the `resources`-attribute of each rule. More details are in the `Snakemake` documentation.

For example, `spike` has ~70 different rules, each rule running a different command with different resource demands, depending on typical input sizes. Thus, finding working upper bounds required some experience, see the corresponding cluster.json-file as an example. First entry is the default, which will be applied to every rule if no further matching definition can be found. The other entries correspond to the names of rules defined in your Snakemake workflow. The parameters specified in each entry will override the corresponding parameters from the `__default__` entry.

- In the default rule, we define the `account` that is authorized to used this service. This is your project identifier which you specified when applying for your HPC account.
- The HPC has several `queues` into which your jobs can be submitted. For `spike`, we always use the "default" queue.
- `nodes` is the number of machines / computers you request to execute your job. In most cases, you'll want to run each job on just one node, instead of spreading a single job across multiple nodes.
- `time` is the maximum execution time of your job. In the above example, the default it is set to 59 minutes and 59 seconds.
- `mem` is the main memory you request for your job. The scheduler will kill your program if you exceed `time` or `memory` for a significant amount. But keep in mind, too greedy resource definitions will delay execution of your job.
- `ppn` is the number of cores / CPUs for your job. Make sure your program can actually use multiple cores if you request them. Otherwise resouces will be wasted and can't be used by other users!

--cluster "qsub -A {cluster.account} -q {cluster.queue} -l select={cluster.nodes}:ncpus{cluster.ppn}:mem={cluster.mem} -l walltime={cluster.time}"

You make use of all those above settings by invoking `snakemake` with the flag `--cluster-config cluster.json`, but you also have to define which grid command shall be used to actually submit a job to the grid. For the HPC and the cluster.json file of spike, it looks like `--cluster "qsub -l select={cluster.nodes}:ncpus{cluster.ppn}:mem={cluster.mem} -l walltime={cluster.time}"`. I might recognize the variable names from cluster.json appear here in curly brackets, i.e. those strings will be replaced by the values defined in the cluster.json file.

--cluster-status scripts/barnacle_status.py

From Snakemakes documentation `Status command for cluster execution. This is only considered in combination with the –cluster flag. If provided, Snakemake will use the status command to determine if a job has finished successfully or failed. For this it is necessary that the submit command provided to –cluster returns the cluster job id. Then, the status command will be invoked with the job id. Snakemake expects it to return ‘success’ if the job was successfull, ‘failed’ if the job failed and ‘running’ if the job still runs.` We are using the script https://github.com/sjanssen2/spike/blob/master/scripts/barnacle_status.py for this purpose. (Side note: "barnacle" is the cluster system I was using in San Diego and I was lazy and copy & pasted this script from https://github.com/biocore/oecophylla Thanks Jon for working that one out!)

--max-status-checks-per-second 1

`Snakemake` need to "ping" the scheduler frequently to ask for the status of its jobs. In order to avoid too much asking overhead, I am limiting the number of questions to just one per seconds. This works fine, since `spike` jobs usually run for hours and thus this is no real delay for executing the whole pipeline.

--latency-wait 900

Jobs are executed on specific machines and results will be written in some file. The grid file system than needs to make sure that this file is synchronized with all other machines before execution of a dependent job. This process sometimes takes some time. In my experience, it doesn't hurt to be patient and wait for 900 seconds. If the file does not appear during this long period of time, `snakemake` will treat the job as failed, even though the correct result might pop up later.

--use-conda

HPC admins encourage you to **not** use conda installed packages and instead use their optimized software versions which can be loaded via `module` http://modules.sourceforge.net/
However, since reproducibility is of paramount importance for `spike`, I decided against HPCs suggestion and used many different conda environments to enforce exact program versions that can also easily installed by collaboration partners on their machines. Via the parameter `--use-conda` you enable `snakemake`s fantastic mechanism that automatically downloads and creates conda environments for different rules.

-j 100

This parameter specifies how many of your maybe ten thousands of jobs are submitted at the same time to the scheduler. Don't use much higher numbers as there is the risk that it crashes or slows down the scheduler; not only for you but for *all* users of the HPC!

--keep-going

It might happen that single rules / programs of your pipeline execution fails. Since with `spike` you typically process a multitude of independent samples / trios you don't want to immediately stop execution of all jobs if one fails for one sample. Once you identified and fixed the issue with the one failing job, just re-execute the snakemake command and it will continue from there. Very convenient.

-p and -r

The flag `-p` will print the (shell) commands to be executed. Useful for debugging. The flag `-r` reports why a specific rule for a specific input need to be (re) executed. Using both aids debugging and understanding the flow of your pipeline.

Executing

1. You should start a new screen https://linuxize.com/post/how-to-use-linux-screen/ session: `screen -S spike`.
2. a. Since the login node has very limited resources, you should start an new interactive gird job, with medium memory, one node and one core, but relatively long runtime:

```
qsub -I -A ngsukdkohi -l mem=10GB -l walltime="40:59:50,nodes=1:ppn=1"
```

   b. Alternatively you can run your master job on a dedicated node (hilbert211 at the moment, ask in the Rocket.Chat in case this information is outdated). Make sure that you are not actually causing workload there!
3. navigate to your `spike` clone and make sure you configure `spike` correctly (docs to be come), specifically `snupy` credentials and selection of samples to be processed.
4. Trigger a dry run of the pipeline by using the `-n` flag of snakemake and check that everything looks good:

```
snakemake --cluster-config cluster.json --cluster "qsub -A {cluster.account} -q {cluster.queue} -l
select={cluster.nodes}:ncpus{cluster.ppn}:mem={cluster.mem} -l walltime={cluster.time}" -j 100 --latency-
wait 900 --use-conda --cluster-status scripts/barnacle_status.py --max-status-checks-per-second 1 --keep-
going -p -r -n
```

5. (During development of `spike` it happened to me that `snakemake` wanted to re-execute long running programs because of changes in the Snakefiles, but it would produce identical results. To avoid the waste of compute, I am sometimes "touching" output files to update the time stamp such that snakemake will not reinvoke execution. This harbours the risk of computing with outdated or even incomplete intermediate results! Be careful: replace `-n` with `--touch`.
6. Trigger actual snakemake run, by removing `-n` (and `--touch`) from the above command.

Working Example

Here is an example for a SM workflow that can be called by simply executing the clusterExecution.sh script. It also makes use of the HPC modules for Snakemake and Conda instead of relying on a separate installation: