Runnimg the PSB Pipeline from a Singularity Container, 2021-09-05

Overview

The Personal Structural Biology Pipeline reads a table of mutations from a missense.csv file for a given patient (also known as project, or case) and launches a variety of analysis algorithms (pathprox, ddG, sequence) on each mutation.

A typical case run involves ~50 mutations, and runs of ~500 indepent programs. Currently, the Pipeline runs on "slurm" or "LSF" compute clusters, though the code has been architected to be as cluster-independent as possible.

The final output is a summary .html report, which can be drilled-down to per-mutation structural analysis reports.

Quick Rampup

.config files connect the container to downloaded external filesets.

1) Obtain the singularity image file from Chris Moth.
2) Download structure, genomic, and variant files as documented in DownloadData.md
3) Create a ddg Repository directory on your file system. DDG calculation results will be archived under this directory, to avoid re-calculations. Place ddg_repo.config in the top level of that directory, and edit one line in file, following the instructions in the template.
4) Create an empty master (parent) parent work directory on your file system.
   - Each variant set, or UDN 'case' will occupy a child directory under this parent. It is convenient to point an environment variable to this directory in your .bashrc file. We tend to 'export UDN=/our/case_caseparent' in our .bashrc files to accomplish this
   - It may be convenient, though unnecessary, to place the .simg Singularity container file in this parent directory.
   - Verify that you can access this directory from inside your container
     - i.e. $ singularity shell /your/location/image_phase9.simg
     - ls /wherever/parent
5) Create a config/ directory under the parent. Copy the provided global.config file into the parent/config/, and customize it to the filesystem locations where you downloaded data.

  - The provided global.config sample file is highly commented, and is not re-documented here.
  - The directories in the .config files must be visible from *inside* the singularity container.

6) In the parent work directory (I will move to parent/config/ in future version), create a your_username.config file. In a typical multi-user pipeline installation, this file allows each user to override global settings. Minimally, each user will place their email address, and slurm (or LSF) cluster job notification preferences in this file. These are copied into the headers of the pipeline-generated .slurm files.

  - See mothcw.config as an example

Running case a case (analyzing a set of variants) in brief:

1) Under the parent work directory, create a new case directory, in which a set of variants will be analyzed together. All pipeline outputs are written into this directory and subdirectories created by the pipeline.

  - mkdir $UDN/mycase
  - cd $UDN/mycase

2) Create a file specifically named mycase_missense.csv following the instructions below ("Preparation of Input")

  - A sample mycase_missense.csv is shown under **Preparation of Input**

The section below shows what should happen when the container launches, and you run psb_plan.py. Return to the instructions here, with step 6.

The four automation steps, in brief: Prep, Parse, Plan, Launch, Monitor, Report

0) Launch the container, so that $UDN/mycase is the current directory.

In our experience, this is no more complex than

```
- cd $UDN/mycase
- singularity shell /your/location/image_phase9.simg
```

Except for a sub-step to launch all the slurm jobs, the commands below are executed *inside* the container.

1) Plan the work (jobs to be run for each mutation, based on available structures)

```
psb_plan.py
```

2) Launch the jobs

```
psb_plan.py --nolaunch
```

The –nolaunch option is required because .slurm files cannot be launched (sbatch) from inside the container.
Follow the on-screen instructions to run the created launch.py program. Then, return to the container to monitor jobs.

3) Monitor progress at intervals.

You can "watch" the progress of your jobs with cluster commands like 'squeue' HOWEVER, all pipeline jobs update a status file system that is cluster-indepednent.

From the pipeline's perspective the status of jobs is given by this command:

```
psb_monitor.py
```

8) When Pathprox jobs have completed, run the final reports. Ddg results will be included as they are available

```
psb_rep.py
```

Important: Always re-run psb_monitor.py before psb_rep.py if you believe more jobs may have finished.

You may move the final .zip or .tar.gz files to extract to a laptop, or public website.

Preparation of input

The format of the yourcase_missense.csv file is precise, but simple. It is typically "easy" to create or edit these files by hand if a source .xlsx is not available, or is badly mis-formatted.

Each row contains an index value (ignored), a gene name, a refseq identifier, a transcript-referenced amino acid mutation, and a uniprot ID for the refseq ID (likely optional - need to check).

```
/capra_lab/projects/psb_collab/UDN/Test501234% cat Test501234.csv
,gene,refseq,mutation,unp
0,KCNA2,NM_001204269,R300C,P16389-2
1,FRAS1,NM_025074,H3285Y,Q86XX4-2
2,FRAS1,NM_025074,P214L,Q86XX4-2
3,AP3B2,NM_004644.4,E465K,Q13367-1
4,ATP6V0A1,NM_001130020.1,R741Q,Q93050-3
5,RAPGEF6,NM_016340,N1075S,Q8TEU7-1
6,ALG11,NM_001004127,Q213P,Q2TAA5
```

- There are several supplemental programs available to automatically create the mycase_missense.csv file from sources such as genomic coordinates, gene names, UDN medical case spreadsheets. Please ask if any of those would be helpful. For testing, please create a file with a couple of variants.

The four automation steps, in Detail: Prep, Parse, Plan, Launch, Monitor, Report

Configuration files: In 99% of cases, configuration files are not re-specified on command lines, but are located from default locations described under the config notes above. In addition to global.config, and username.config, a third .config override file may be placed in the case directory. Name it mycase.config and it will be picked up automatically by all pipeline tools. A typical use for a mycase.config is to override PathProx input variant sets, or for testing of new codes or options without perturbing the global configuration.

Pipeline step 1: Create a work plan from the csv file

(Don't stress over remembering command line arguments. Use the standard '-h' option for a helpful reminder of the default inputs)

Be sure to specify the global and local config files, as well as your job name. Your command sequence will look like:

$ cd $UDN/mycase $ singularity shell /wherever/image_phase9.simg

Then at the blue container prompt. . . .

$ psb_plan.py

Here is what I see on my system with the above command:

```
140 ~/psbwork % psb_plan.py Test123456
psb_plan.py: Pipeline execution plan generator.  -h for detailed help
Collaboration-wide  psb_plan log file is /dors/capra_lab/projects/psb_collab/UDN/Test123456/
Loading swiss model JSON metadata from /dors/capra_lab/data/swissmodel/SWISS-MODEL_Reposito
Loading idmapping file from /dors/capra_lab/mothcw/mydata/idmapping/HUMAN_9606_idmapping_sp
Retrieving project mutations from /dors/capra_lab/projects/psb_collab/UDN/Test123456/Test12
Work for 2 mutations will be planned
Planning  11,SPRY3,NM_001304990,R242C,O43610
    2 structures retained     0 dropped.    6 jobs will run.  See: $UDN/Test123456/SPRY3_NM_
Planning  16,TPRN,NM_001128228,P39L,Q4KMQ1-1
    3 structures retained     2 dropped.    8 jobs will run.  See: $UDN/Test123456/TPRN_NM_
```

Logging

psb_plan.py creates a master psb_plan.log file for the entire run, as well as an additional log file for each mutation that echoes the running log information. Usually, these logged details are too dense for screen display. Occasionally, they can be very helpful to sort through problems.

To add log entries to your screen display, additionally include "–verbose" when running any of the psb pipeline applications. There is an additional –debug option that could include more messages. By default, only the less common WARNING, ERROR, FATAL, EXCEPTION, and CRITICAL log messages are displayed to the screen.

Typically, you will not want to inspect fine details of the workplan.csv files that are generated for each mutation. The locations of these files are well-documented in the .log files that are listed in the above output. You can review all of the log file, or just "grep" for what interests you:

```
git hub markdown force line break % grep "workplan.csv" /dors/capra_lab/projects/psb_collab/
14:34:22 INFO [         psb_plan.py:730] Workplan written to
/dors/capra_lab/projects/psb_collab/UDN/Test123456/SPRY3_NM_001304990_R242C/SPRY3_NM_00130499
14:34:22 INFO [         psb_plan.py:730] Workplan written to
/dors/capra_lab/projects/psb_collab/UDN/Test123456/TPRN_NM_001128228_P39L/TPRN_NM_001128228_
```

The tab-delimited workplan.csv files are easily reviewed in libreoffice or excel. The first row is a header with column names. The second and following list specific details about each job. Depending on the available structures, a workplan.csv file could have 1 job or 100. The average seems to be just under 10. For purpose of explanation, I have transposed the first two rows of a workplan.csv file (row 1 is shown as the left column below):

```
uniquekey   SPRY3_NM_001304990_R242C_ENSP00000302978_1_A_PathProxCOSMIC
chain       A
command     pathprox2.py
config      /dors/capra_lab/users/psbadmin/config/global.config
cwd     /dors/capra_lab/projects/psb_collab/psb_pipeline/bin
flavor      PathProxCOSMIC
gene        SPRY3
method      MTALL
mutation    R242C
options     -c /dors/capra_lab/users/psbadmin/config/global.config -u mothcw.config ENSP0000
outdir      /dors/capra_lab/projects/psb_collab/UDN/Test123456/SPRY3_NM_001304990_R242C/ENSP
pdbid       ENSP00000302978_1
project     Test123456
refseq      NM_001304990
unp     O43610
userconfig  mothcw.config
```

Structure Reports

Another important output from psb_plan.py is the . . . structure_report.csv and . . . dropped_models.csv.
(**Need to change names of these files**) These files, in combination with the details in the log file itself, document the consideration of all candidate 3D structures that are incorporated into (or dropped from) the workplan. Typical reasons for dropping a structure include the availability of higher resoltion .pdb files, higher sequence identity models, or duplication of models between swiss, modbase16, and modbase13, etc.

Pipeline step 2: Launch jobs

Login to your slurm cluster head node, and launch all the jobs with

```
$ psb_launch.py --nolaunch
```

The program generates a custom launch.py program, which in turn must be run outside the container.

For each of the jobs in the workplan.csv files, a slurm file will be created, and submitted with the "sbatch" command

Here is a sample run (which is not right at all -because things look different when you are using hte container, and the external launch program - sorry).

```
~/psbwork$ psb_launch.py TestJED -u mothcw.config --relaunch
psb_launch.py: Pipeline launcher.  Run after psb_plan.py.   -h for detailed help
Retrieving project mutations from /dors/capra_lab/projects/psb_collab/UDN/TestJED/TestJED.cs
Launching all jobs for 2 mutations
Launching SPRY3      NM_001304990 R242C
       24753955:SPRY3_NM_001304990_R242C_ENSP00000302978_1_A_PathProxCOSMIC
       24753956:SPRY3_NM_001304990_R242C_ENSP00000302978_1_A_PathProxClinvar
       24753957:SPRY3_NM_001304990_R242C_ENSP00000302978_2_A_PathProxCOSMIC
       24753958:SPRY3_NM_001304990_R242C_ENSP00000302978_2_A_PathProxClinvar
       24753959:SPRY3_NM_001304990_R242C_ENSP00000302978_2_A_ddG
       24753960:SPRY3_NM_001304990_R242C_SequenceAnnotation
Recording all jobids to /dors/capra_lab/projects/psb_collab/UDN/TestJED/SPRY3_NM_001304990_F
Launching TPRN       NM_001128228 P39L
       24753961:TPRN_NM_001128228_P39L_ENSP00000387100.4_1_A_PathProxCOSMIC
       24753962:TPRN_NM_001128228_P39L_ENSP00000387100.4_1_A_PathProxClinvar
       24753963:TPRN_NM_001128228_P39L_ENSP00000387100_1_A_PathProxCOSMIC
       24753964:TPRN_NM_001128228_P39L_ENSP00000387100_1_A_PathProxClinvar
       24753965:TPRN_NM_001128228_P39L_ENSP00000387100_1_A_ddG
       24753966:TPRN_NM_001128228_P39L_ENSP00000387100_2_A_PathProxCOSMIC
       24753967:TPRN_NM_001128228_P39L_ENSP00000387100_2_A_PathProxClinvar
       24753968:TPRN_NM_001128228_P39L_SequenceAnnotation
```

```
Recording all jobids to /dors/capra_lab/projects/psb_collab/UDN/TestJED/TPRN_NM_001128228_P3
```

In the above output, the slurm job ids are shown to the left of the job names (which are simply the 'uniquekey' components of the original work plan)

Pipeline step 3: Monitor jobs until completion

Every job has its own "outdir" where its results are stored. Under that directory is a 'status' directory which is cleared by psb_launch.py. This directory contains up to 3 files that are updated as each job runs:

- status/complete The presence of this empty file informs the monitor that the job has exited with status code 0, indicating that it completed satisfactorily, and that its output is ready for processing
  OR
- status/FAILED The presence of this file tells the monitor that the job has exited with "code 1" which indicates
  ALSO
- status/info The contents of the file tell the monitor how the job is doing. It is set to "Submitted" on initial submit by psb_launch.py
- status/progress Each time the info file is updated, the progress file is updated with the line number being execited in the .py file

As soon as the monitor sees the "complete" or "FAILED" marker files, the workstatus.csv row for the job is updated with status/info, and no further checks are performed.

(In previous version, psb_monitor.py would additionally inspect the output of "scontrol show job jobid" to help fill in the workstatus.csv file with the most complete information possible. However, this had dramatically slow runtime performance, and that section of code was commented out)

IMPORTANT: Final recording of Exit Code is done from the definitive presence of the status/complete or status/FAILED flags. It is not necessary for slurm to "remember" a job in order to have its exit status recorded

Here is an example of a psb_monitor.py run on my system

```
$ cd ~/psbwork
~/psbwork $ psb_monitor.py TestJED
./psb_monitor.py: Pipeline monitor for launched jobs.  -h for detailed help.
Retrieving project mutations from /dors/capra_lab/projects/psb_collab/UDN/TestJED/TestJED.cs
Monitoring all jobs for 2 mutations
Monitoring SPRY3      NM_001304990 R242C
Recording all updates to /dors/capra_lab/projects/psb_collab/UDN/TestJED/SPRY3_NM_001304990_
4 of 6 jobs still incomplete:
          Jobid:Flavor                Info
       24753955:SPRY3_NM_001304990_R242C_ENSP00000302978_1_A_PathProxCOSMIC    Submitted
       24753956:SPRY3_NM_001304990_R242C_ENSP00000302978_1_A_PathProxClinvar    Submitted
       24753957:SPRY3_NM_001304990_R242C_ENSP00000302978_2_A_PathProxCOSMIC    Submitted
       24753958:SPRY3_NM_001304990_R242C_ENSP00000302978_2_A_PathProxClinvar    Submitted
Monitoring TPRN       NM_001128228 P39L
Recording all updates to /dors/capra_lab/projects/psb_collab/UDN/TestJED/TPRN_NM_001128228_
6 of 8 jobs still incomplete:
          Jobid:Flavor                Info
       24753961:TPRN_NM_001128228_P39L_ENSP00000387100.4_1_A_PathProxCOSMIC    Submitted
       24753962:TPRN_NM_001128228_P39L_ENSP00000387100.4_1_A_PathProxClinvar    Submitted
       24753963:TPRN_NM_001128228_P39L_ENSP00000387100_1_A_PathProxCOSMIC    Submitted
       24753964:TPRN_NM_001128228_P39L_ENSP00000387100_1_A_PathProxClinvar    Submitted
       24753966:TPRN_NM_001128228_P39L_ENSP00000387100_2_A_PathProxCOSMIC    Submitted
       24753967:TPRN_NM_001128228_P39L_ENSP00000387100_2_A_PathProxClinvar    Submitted
```

Reminder to document J'ns method for repeatedly calling something until it is done. We need this

You are welcome to independently inspect your slurm job queue with slurm commands. However, psb_monitor.py brings the advantage of updating your workstatus.csv files. It also reports much more specifically on the nature of failed jobs.

Because the job names are quite long, the standard formatting of the slurm squeue command can be unsatisfactory. You may find an alias with full-formatting to be helpful. I use this to review all my running, pending, and recently exited jobs on our cluster:

alias sq='squeue -o "%.9i %65j %.2T %.9M %.9l %R" -u mothcw'

Full documentation of the slurm squeue command is available here

Pipeline step 4: Running reports

When jobs are complete, you may generate reports.

```
% cd $UDN/UDN123456
% psb_rep.py
4 of 4  ./psb_rep.py Pipeline report generator.  -h for detailed help
Retrieving project mutations from /dors/capra_lab/projects/psb_collab/UDN/TestJED/TestJED.c
Reporting on all 2 project TestJED mutations
Reporting on SPRY3      NM_001304990 R242C
Generating html and pdf final reports for 2 of 6 complete jobs:
```

A complete portable website fileset is created, in both .zip and .tar.gz file formats.

Running a report for only one mutation

A nice thing about all the psb_*.py programs is that you can somewhat bypass the *entire* list of mutations, and run a report (or launch or monitor) just one mutation.

Once initialized, psb_rep.py can take a few minutes per mutation to report. For long mutation lists, this can be tedious, and you may want to parallelize the run. Simply add the –slurm option to do this as in:

```
psb_rep.py TestJED -u mothcw.config --slurm
```

You must manually submit the slurm file to the scheduler with the sbatch command. It is listed at the end:

> ~/psbwork $ psb_rep.py UDN525217 -u mothcw.config -c ../config/v13.config –slurm 4 of 4 ./psb_rep.py Pipeline report generator. -h for detailed help Retrieving project mutations from /dors/capra_lab/users/mothcw/UDNtests/UDN525217/UDN525217.csv
> Slurm script to run 13 reports for UDN525217 is /dors/capra_lab/users/mothcw/UDNtests/UDN525217/sl
> Generating slurm entry for DENND5B NM_144973 D849E Generating slurm entry for GOLGA6L2 NM_001304388 G567E Generating slurm entry for TMEM37 NM_183240 T70P Generating slurm entry for CLUH NM_015229 P227L Generating slurm entry for FRMD4A NM_018027 T286M Generating slurm entry for MED13L NM_015335 M323I Generating slurm entry for RAI1 NM_030665 I898V Generating slurm entry for PCDHGB4 NM_003736 P15L Generating slurm entry for TPRA1 NM_001136053 V196A Generating slurm entry for ATP2C2 NM_014861 S81W Generating slurm entry for C3orf62 NM_198562 G205S Generating slurm entry for C3orf62 NM_198562 N142D Generating slurm entry for ARHGAP6 NM_013427 L75F Created slurm script to launch all psb_rep.py processes for this case: /dors/capra_lab/users/mothcw/UDNtests/UDN525217/slurm/psb_reps.slurm

The -h option to psb_rep.py reminds you that you can also run or rerun a report for only one of the mutations.

$ psb_rep.py -u mothcw.config $UDN/UDN525217/ARHGAP6_NM_013427_L75F/ARHGAP6_NM_013427_
$UDN/UDN525217/ARHGAP6_NM_013427_L75F/ARHGAP6_NM_013427_L75F_workstatus.csv