Workflow-Based TSS Prediction

Chong Lu, Thomas Dutschmann, Minyue Qi

Eberhard Karls Universität Tübingen

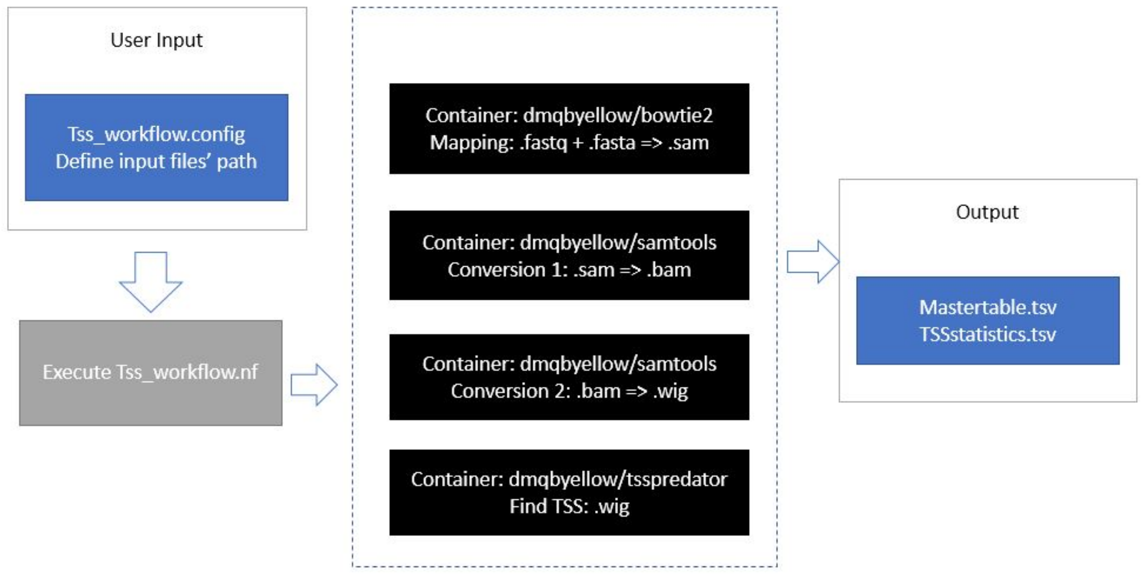
1. Why nextflow

In the first stage of our project, we evaluated three workflow languages, which are Snakemake, Nextflow and WDL. All three workflow languages require a Unix/Linux based operating system. Snakemake is a python based and domain-specific syntax workflow language. As for the management system in Snakemake, rules define how to obtain output files from input files. But Snakemake doesn't include the ability to run Docker containers, which is the reason we dropped this workflow language [1]. WDL is a domain-specific language, and Nextflow is Groovy-based. Installation of Nextflow requires Java 7, while WDL requires Java 8. Since WDL is still 'incomplete and out of date', we choose Nextflow to perform our TSS prediction workflow, even though Nextflow performed quite slow during the whole implementation [2]. In addition to the command line interface, there is a plugin in MSP (Nextflow Workbench) for Nextflow, which makes it easer to write a workflow script. Currently, it's only runnable in macOS now.

2. Project architecture

Our project architecture is shown in figure 1. We packed data processing to the black colored boxes, users merely have the task to define the path of their files in Tss_workflow.config and execute Tss_workflow.nf afterwards.

Figure 1: *Project architecture*


User Input
Tss_workflow.config
Define input files' path
Execute Tss_workflow.nf
Container: dmqbyellow/bowtie2
Mapping: .fastq + .fasta => .sam
Container: dmqbyellow/samtools
Conversion 1: .sam => .bam
Container: dmqbyellow/samtools
Conversion 2: .bam => .wig
Container: dmqbyellow/tsspredator
Find TSS: .wig
Output
Mastertable.tsv
TSSstatistics.tsv

3. Obstacles during the implementation

3.1. Mapping

We applied bowtie 2.2.7 to perform the mapping, conversion an sorting was done using samtools 1.5 [3, 4]. An issue we encountered very early is that samtools requires many libraries as prerequisites to be installed. The definition of a working Dockerfile for a samtools Docker image took more time than expected, and regarding reproducibility, it is the users' task to keep the Dockerfile up-to-date, should the samtools developers ever decide to switch to other prerequisites. Keeping in mind that we will push several read files through our workflow, the overall time may become inconvenient.

3.2. Conversion

Tsstools 1.0_beta is used for file conversion, but users should create an output folder by their own for the tool to put wig or rssc files. If the out directory doesn't exist, the workflow will break off and report 'can't find files' errors. It will be more convenient if the tsstools can create a new folder when users input a non-existent directory.

3.3. Tsspredator

We used TSSpredator 1.06 in our workflow. It requires Java 8, which is installed in our Dockerfile of container 'tsspredator'.

Tsspredator takes .wig files, a genome file and its annotation as input, then generates Mastertable.tsv, TSSstatistics.tsv as our final output files. The problem is, we can only define the path of these files by hand in the configuration of Tsspredator instead of doing it automatically. Moreover, when we tried to run it via ENTRYPOINT in our Dockerfile, it always reports the lack of space capacity for Tsspredator, which does not occur when run locally.

3.4. Overall

Normally, Nextflow has its own data stream for input and output files. It will generate all the output files in the work directory and the output files can be called without giving the file path. This feature makes file management easier and simpler, but at the same time, it can cause problems when the process is running in the Docker container or when the tool used in the process will generate output files in the assigned path. So we need to write a path for each output file to make sure they are callable in a Docker container volume.

When a Nexflow script reads files inside a Docker volume, it seems like the **file** object is invalid. The *Channel.fromPath()* method can only be used when we want to read a single file. For multiple input files, eg, '*.fasta', there is always a syntax error. So we had to give up read multiple files at the same time in a folder, and decide to let users give all the input files as a string 'FASTQ1 FASTQ2 ...'.

4. Unsolved problem

Unfortunately, we were unable to test a mapping software beside bowtie2. For further evaluation, it would be interesting to have a look on performance in terms of speed, even though bowtie2 performs well compared to most other mapping applications [5].

Due to the ambiguity of the assignment, we write the execution commands of each tool inside our Nextflow file instead of within Docker containers. In this case, containers work as environments, so that users have no need to install the programs. But the implementation of each tools still depends on Nextflow.

As an alternative, we also created Dockerfiles with ENTRYPOINT command, which can build executable containers as the alternative solution of our work that makes users free from specifying file paths manually. Since we implemtented Nextflow applications with Docker and configuration files, it could be easily modified and finished with similar processes.

References

[1] "Docker explained." `https://www.docker.com/what-docker`. Accessed 2017/6/5.

[2] "Wdl specification." `https://software.broadinstitute.org/wdl/documentation/topic?name=wdl-spec`. Accessed 2017/6/5.

[3] "Bowtie2 homepage." `http://bowtie-bio.sourceforge.net/bowtie2/index.shtml`. Accessed 2017/6/5.

[4] "Samtools." `http://samtools.sourceforge.net`. Accessed 2017/6/5.

[5] "Bowtie2, how it performs, benchmark-based." `http://www.ecseq.com/support/benchmark`. Accessed 2017/6/5.