The PanPipe Workflow Manager

Daniel Ortiz Martínez

Mathematics and Computer Science Department

University of Barcelona

Table of Contents

Introduction

Introduction

- Pipeline execution is a complex task
    - Pipeline composed of very heterogeneous tasks/processes
    - Processes may present dependencies with other ones
    - Often necessary to add or remove pipeline processes
    - Need to allocate computational resources
    - Independent processes should be executed concurrently
    - Hard to maintain and reuse code
    - ...
- `PanPipe` has been created as a highly portable, configurable and extensible solution

Package Overview

Package Dependencies

- Shell Bash
- Python
- Slurm Workload Manager (optional)

Package Installation

- Obtain the package using git:

```
git clone https://github.com/daormar/panpipe.git
```

- Change to the directory with the package's source code and type:

```
./reconf
./configure
make
make install
```

NOTE: use `--prefix` option of `configure` to install the package in a custom directory

Functionality

- `PanPipe` is an engine to execute general pipelines
- Executes only those pipeline processes that are pending
- Handles computational resources for each process
- Executes process arrays

Execution Model

- `PanPipe` follows the *flow-based programming* paradigm
  - Network of *black box* processes
  - Relations between processes are defined by the data they exchange
  - Component oriented
- `PanPipe` follows a simple execution model based on a file enumerating a list of pipeline processes to be executed
- Processes are executed simultaneously unless dependencies are specified
- Process implementation is given in module files

Main Tools and File Formats

Main Tools

- `panpipe_exec`
- `panpipe_exec_batch`
- `panpipe_check`
- `panpipe_status`

panpipe_exec

- Automates execution of general pipelines
- Main input parameters:
    - `--pfile <string>`: file with pipeline processes to be performed
    - `--outdir <string>`: output directory
    - `--sched <string>`: scheduler used for pipeline execution
    - `--showopts`: show pipeline options
    - `--checkopts`: check pipeline options
    - `--debug`: do everything except launching pipeline processes

panpipe_exec: Output

- Content of output directory:
  - `scripts`: directory containing the scripts used for each pipeline process
  - `<pipeline_process_name>`: directory containing the results of the pipeline process of the same name
- Additional directories may be created depending on the pipeline

`panpipe_exec`: Available Schedulers

- **Built-in Scheduler**
  - Allows to execute pipelines locally
  - Incorporates a basic resource allocation mechanism
- **Slurm Scheduler**
  - Allows to exploit large computational resources
  - Usage transparent to the user
  - Slurm behavior influenced by pipeline description

panpipe_exec_batch

- Automates execution of pipeline batches
- Main input parameters:
  - `-f <string>`: file with a set of `panpipe_exec` commands
  - `-m <string>`: Maximum number of concurrently executed pipelines
  - `-o <string>`: Output directory to move output of each pipeline

panpipe_check

- Checks correctness of pipelines and converts them to other formats
- Main input parameters:
  - `-p <string>`: pipeline file
  - `-g`: print pipeline in `graphviz` format

`panpipe_status`

- Checks execution status of a given pipeline
- Main input parameters:
  - `-d <string>`: directory where the pipeline processes are stored
  - `-s <string>`: process name whose status should be determined (optional)

The `panpipe_lib.sh` Library

- Shell library with functions used by the previously described tools
- Functions can be classified as follows:
  - Implementation of the package execution model
  - Automated creation of scripts executing pipeline processes
  - Helper functions to implement pipeline processes

File Formats

- **Pipeline file**: file enumerating all of the pipeline processes to be carried out when processing a normal-tumor sample
- **Module file**: file defining the code of the pipeline processes
- **Pipeline automation script**: file with a sequence of `panpipe_exec` commands automating the analysis of a dataset

Pipeline File

- **Module import** (module names separated by commas)
- **Entry format** (one entry per line)

  Process name, Slurm account, Slurm partition, CPUs, Memory limit, Time limit, Dependencies, ...

- **Dependency types:** `none, after, afterok, afternotok, afterany`

```
#import panpipe_software_test
#
process_a  cpus=1 mem=32 time=00:01:00 processdeps=none
process_b  cpus=1 mem=32 time=00:01:00 processdeps=afterok:process_a
process_c  cpus=1 mem=32 time=00:01:00 throttle=2 processdeps=afterok:process_a
```

Module File

- Contains the definition of the different processes
- Written in `bash`
- Three `bash` functions should be defined for each process:
    - `processname_explain_cmdline_opts()`
    - `processname_define_opts()`
    - `processname()`

Module File: `processname_explain_cmdline_opts()`

- This function documents the command line options that the process needs to work
- The aggregated documentation for the different processes is shown when executing `panpipe_exec --showopts`
- Whenever two processes share the same option, it is important to give it the same name

Module File: `processname_explain_cmdline_opts()`

```
process_a_explain_cmdline_opts()
{
    # -a option
    description="Sleep time in seconds for process_a (required)"
    explain_cmdline_opt "-a" "<int>" "$description"
}
```

Module File: `processname_define_opts()`

- This function should create a string containing the options that are specific to the process
- The main idea is to map command line options to process options
- The package provides multiple built-in functions to make the implementation of this function easier

Module File: `processname_define_opts()`

```
processname_define_opts()
{
    # Initialize variables
    local cmdline=$1
    local process_spec=$2
    local optlist=""

    # Use built-in functions to add options to optlist variable
    ...

    # Save option list
    save_opt_list optlist
}
```

Module File: `processname_define_opts()`

```
process_a_define_opts()
{
    # Initialize variables
    local cmdline=$1
    local process_spec=$2
    local optlist=""

    # -a option
    define_cmdline_opt "$cmdline" "-a" optlist || exit 1

    # Save option list
    save_opt_list optlist
}
```

Module File: `processname()`

- Implements the process
- The function should incorporate code at the beginning to read the options defined by `processname_define_opts()`

Module File: `processname()`

```
process_a()
{
    # Initialize variables
    local sleep_time=`read_opt_value_from_line "$*" "-a"`

    # Sleep some time
    sleep ${sleep_time}
}
```

Pipeline Automation Script

- Automates the analysis of a whole dataset
- At each entry (one per line), `panpipe_exec` tool is used to execute a whole pipeline
- Can be used as input for `panpipe_exec_batch`
- Entry example:

```
panpipe_exec --pfile example.ppl --outdir outdir1 --sched SLURM -opt1 <opt1_val> -opt2 <opt2_val> ...
panpipe_exec --pfile example.ppl --outdir outdir2 --sched SLURM -opt1 <opt1_val> -opt2 <opt2_val> ...
panpipe_exec --pfile example.ppl --outdir outdir3 --sched SLURM -opt1 <opt1_val> -opt2 <opt2_val> ...
...
panpipe_exec --pfile example.ppl --outdir outdirn --sched SLURM -opt1 <opt1_val> -opt2 <opt2_val> ...
```

Extending Modules

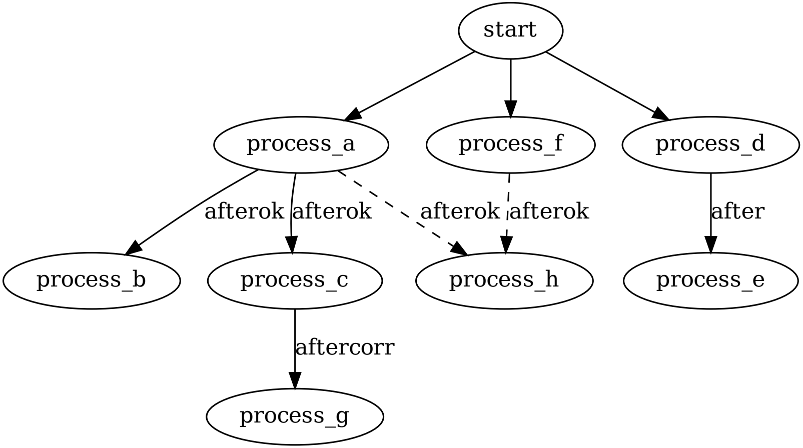
- Since multiple imports are permitted, a new module may contain process definitions missing in another one
- The order in which modules are imported is relevant
    - if two modules define the same function, the definition in the module imported last will prevail
    - the previous property can be used to modify a specific process without repeating the code of the whole module

Toy Pipeline Example

Pipeline File

```
#import panpipe_software_test
#
process_a cpus=1 mem=32 time=00:01:00 processdeps=none
process_b cpus=1 mem=32 time=00:01:00 processdeps=afterok:process_a
process_c cpus=1 mem=32 time=00:01:00 throttle=2 processdeps=afterok:process_a
process_d cpus=1 mem=32 time=00:01:00 processdeps=none
process_e cpus=1 mem=32 time=00:01:00 processdeps=after:process_d
process_f cpus=1 mem=32 time=00:01:00 processdeps=none
```

Pipeline Representation


start
process_a
process_f
process_d
afterok
afterok
afterok
afterok
after
process_b
process_c
process_h
process_e
aftercorr
process_g