Flux

Release 0.1.0

This page is maintained by the Flux community.

Mar 11, 2021

CONTENTS

The examples contained here demonstrate and explain some simple use-cases with Flux, and make use of Flux's command-line interface (CLI), Flux's C library, and the Python and Lua bindings to the C library. The entire set of examples can be downloaded by cloning the Github repo.

The examples assume that you have installed:

1. A recent version of Flux
2. Python 3.6+
3. Lua 5.1+

CHAPTER

ONE

CLI: JOB SUBMISSION

Launch a flux instance and schedule/launch compute and io-forwarding jobs on separate nodes using the CLI

CHAPTER

TWO

PYTHON: JOB SUBMISSION

Schedule/launch compute and io-forwarding jobs on separate nodes using the Python bindings

CHAPTER

THREE

PYTHON: JOB SUBMIT/WAIT

Submit jobs and wait for them to complete using the Flux Python bindings

CHAPTER

FOUR

PYTHON: ASYNCHRONOUS BULK JOB SUBMISSION

Asynchronously submit jobspec files from a directory and wait for them to complete in any order

CHAPTER

FIVE

PYTHON: TRACKING JOB STATUS AND EVENTS

Submit job bundles and wait until all jobs complete

CHAPTER

SIX

PYTHON: JOB CANCELLATION

Cancel a running job

CHAPTER

SEVEN

LUA: USE EVENTS

Use events to synchronize compute and io-forwarding jobs running on separate nodes

CHAPTER

EIGHT

PYTHON: SIMPLE KVS EXAMPLE

Use KVS Python interfaces to store user data into KVS

CHAPTER

NINE

CLI/LUA: JOB ENSEMBLE SUBMITTED WITH A NEW FLUX INSTANCE

Submit job bundles, print live job events, and exit when all jobs are complete

CHAPTER

TEN

CLI: HIERARCHICAL LAUNCHING

Launch a large number of sleep 0 jobs

CHAPTER

ELEVEN

C/LUA: USE A FLUX COMMS MODULE

Use a Flux Comms Module to communicate with job elements

CHAPTER

TWELVE

C/PYTHON: A DATA CONDUIT STRATEGY

Attach to a job that receives OS time data from compute jobs

12.1 Job Submit CLI

To run the following examples, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/job-submit-cli
```

12.1.1 Part(a) - Partitioning Schedule

Description: Launch a flux instance and schedule/launch compute and io-forwarding jobs on separate nodes

1. `salloc -N3 -ppdebug`
2. `srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out`
3. `flux mini submit --nodes=2 --ntasks=4 --cores-per-task=2 ./compute.lua 120`
4. `flux mini submit --nodes=1 --ntasks=1 --cores-per-task=2 ./io-forwarding.lua 120`
5. List running jobs:

`flux jobs`

```
JOBID     USER     NAME       ST NTASKS NNODES   RUNTIME RANKS
ƒ3ETxsR9H moussa1  io-forward  R      1      1    2.858s 2
ƒ38rBqEWT moussa1  compute.lu  R      4      2     15.6s [0-1]
```

1. Get information about job:

`flux job info ƒ3ETxsR9H R`

```
{"version":1,"execution":{"R_lite":[{"rank":"2","children":{"core":"0-1"}}]}}
```

`flux job info ƒ38rBqEWT R`

```
{"version":1,"execution":{"R_lite":[{"rank":"0-1","children":{"core":"0-3"}}]}}
```

12.1.2 Part(b) - Overlapping Schedule

Description: Launch a flux instance and schedule/launch both compute and io-forwarding jobs across all nodes

1. `salloc -N3 -ppdebug`
2. `srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out`
3. `flux mini submit --nodes=3 --ntasks=6 --cores-per-task=2 ./compute.lua 120`
4. `flux mini submit --nodes=3 --ntasks=3 --cores-per-task=1 ./io-forwarding.lua 120`
5. List jobs in KVS:

`flux jobs`

```
JOBID     USER     NAME       ST NTASKS NNODES  RUNTIME RANKS
ƒ3ghmgCpw moussa1  io-forward  R      3      3   16.91s [0-2]
ƒ3dSybfQ3 moussa1  compute.lu  R      6      3    24.3s [0-2]
```

1. Get information about job:

`flux job info ƒ3ghmgCpw R`

```
{"version":1,"execution":{"R_lite":[{"rank":"0-2","children":{"core":"4"}}]}}
```

`flux job info ƒ3dSybfQ3 R`

```
{"version":1,"execution":{"R_lite":[{"rank":"0-2","children":{"core":"0-3"}}]}}
```

12.2 Job Submit API

To run the following examples, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/job-submit-api
```

12.2.1 Part(a) - Using a direct job.submit RPC

Description: Schedule and launch compute and io-forwarding jobs on separate nodes

1. Allocate three nodes from a resource manager:

`salloc -N3 -p pdebug`

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

`srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out`

1. Run the submitter executable:

`./submitter.py`

1. List currently running jobs:

```
flux jobs
```

```
JOBID    USER     NAME       ST NTASKS NNODES  RUNTIME RANKS
ƒ5W8gVwm moussa1  io-forward  R      1      1   19.15s 2
ƒ5Vd2kJs moussa1  compute.py  R      4      2   19.18s [0-1]
```

1. Information about jobs, such as the submitted job specification, an eventlog, and the resource description format **R** are stored in the KVS. The data can be queried via the `job-info` module via the `flux job info` command. For example, to fetch **R** for a job which has been allocated resources:

```
flux job info ƒ5W8gVwm R
```

```
{"version":1,"execution":{"R_lite":[{"rank":"2","children":{"core":"0"}}]}}
```

```
flux job info ƒ5Vd2kJs R
```

```
{"version":1,"execution":{"R_lite":[{"rank":"0-1","children":{"core":"0-3"}}]}}
```

12.2.2 Part(b) - Using a direct job.submit RPC

Description: Schedule and launch both compute and io-forwarding jobs across all nodes

1. Allocate three nodes from a resource manager:

```
salloc -N3 -p pdebug
```

1. Launch another Flux instance on the current allocation:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Run the second submitter executable:

```
./submitter2.py
```

1. List currently running jobs:

```
flux jobs
```

```
JOBID    USER     NAME       ST NTASKS NNODES  RUNTIME RANKS
ƒctYadhh moussa1  io-forward  R      3      3   3.058s [0-2]
ƒct1StnT moussa1  compute.py  R      6      3   3.086s [0-2]
```

1. Fetch **R** for the jobs that have been allocated resources:

```
flux job info ƒctYadhh R
```

```
{"version":1,"execution":{"R_lite":[{"rank":"0-2","children":{"core":"0-3"}}]}}
```

```
flux job info ƒct1StnT R
```

```
{"version":1,"execution":{"R_lite":[{"rank":"0-2","children":{"core":"0-3"}}]}}
```

12.2.3 Notes

- `f = flux.Flux()` creates a new Flux handle which can be used to connect to and interact with a Flux instance.
- The following constructs a job request using the **JobspecV1** class with customizable parameters for how you want to utilize the resources allocated for your job:

```
compute_jobreq = JobspecV1.from_command(
    command=["./compute.py", "120"], num_tasks=4, num_nodes=2, cores_per_task=2
)
compute_jobreq.cwd = os.getcwd()
compute_jobreq.environment = dict(os.environ)
```

- `flux.job.submit(f, compute_jobreq)` submits the job to be run, and returns a job ID once it begins running.

12.3 Python Job Submit/Wait

To run the following examples, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/job-submit-wait
```

12.3.1 Part(a) - Python Job Submit/Wait

Description: Submit jobs asynchronously and wait for them to complete in any order

1. Allocate three nodes from a resource manager:

```
salloc -N3 -ppdebug
```

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Submit the **submitter_wait_any.py** script, along with the number of jobs you want to run (if no argument is passed, 10 jobs are submitted):

```
./submitter_wait_any.py 10
```

```
submit: 46912591450240 compute_jobspec
submit: 46912591450912 compute_jobspec
submit: 46912591451080 compute_jobspec
submit: 46912591363152 compute_jobspec
submit: 46912591362984 compute_jobspec
submit: 46912591451360 bad_jobspec
submit: 46912591451528 bad_jobspec
submit: 46912591451696 bad_jobspec
submit: 46912591451864 bad_jobspec
submit: 46912591452032 bad_jobspec
wait: 46912591451528 Error: job returned exit code 1
wait: 46912591451864 Error: job returned exit code 1
wait: 46912591451360 Error: job returned exit code 1
```

(continues on next page)

(continued from previous page)

```
wait: 46912591451696 Error: job returned exit code 1
wait: 46912591452032 Error: job returned exit code 1
wait: 46912591450240 Success
wait: 46912591363152 Success
wait: 46912591450912 Success
wait: 46912591451080 Success
wait: 46912591362984 Success
```

12.3.2 Part(b) - Python Job Submit/Wait (Sliding Window)

Description: Asynchronously submit jobs and keep at most a number of those jobs active

1. Allocate three nodes from a resource manager:

```
salloc -N3 -ppdebug
```

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Submit the **submitter_sliding_window.py** script, along with the number of jobs you want to run and the size of the window (if no argument is passed, 10 jobs are submitted and the window size is 2 jobs):

```
./submitter_sliding_window.py 10 3
```

```
submit: 5624175788032
submit: 5624611995648
submit: 5625014648832
wait: 5624175788032 Success
submit: 5804329533440
wait: 5624611995648 Success
submit: 5804648300544
wait: 5625014648832 Success
submit: 5805084508160
wait: 5804329533440 Success
submit: 5986144223232
wait: 5804648300544 Success
submit: 5986462990336
wait: 5805084508160 Success
submit: 5986882420736
wait: 5986144223232 Success
submit: 6164435697664
wait: 5986462990336 Success
wait: 5986882420736 Success
wait: 6164435697664 Success
```

12.3.3 Part(c) - Python Job Submit/Wait (Specific Job ID)

Description: Asynchronously submit jobs, block/wait for specific jobs to complete

1. Allocate three nodes from a resource manager:

```
salloc -N3 -ppdebug
```

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Submit the **submitter_wait_in_order.py** script, along with the number of jobs you want to run (if no argument is passed, 10 jobs are submitted):

```
./submitter_wait_in_order.py 10
```

```
submit: 46912593818008 compute_jobspec
submit: 46912593818176 compute_jobspec
submit: 46912593818344 compute_jobspec
submit: 46912593818512 compute_jobspec
submit: 46912593738048 compute_jobspec
submit: 46912519873816 bad_jobspec
submit: 46912593818792 bad_jobspec
submit: 46912593818960 bad_jobspec
submit: 46912593819128 bad_jobspec
submit: 46912593819296 bad_jobspec
wait: 46912593818008 Success
wait: 46912593818176 Success
wait: 46912593818344 Success
wait: 46912593818512 Success
wait: 46912593738048 Success
wait: 46912519873816 Error: job returned exit code 1
wait: 46912593818792 Error: job returned exit code 1
wait: 46912593818960 Error: job returned exit code 1
wait: 46912593819128 Error: job returned exit code 1
wait: 46912593819296 Error: job returned exit code 1
```

12.3.4 Notes

- The following constructs a job request using the **JobspecV1** class with customizable parameters for how you want to utilize the resources allocated for your job:

```
# create jobspec for compute.py
compute_jobspec = JobspecV1.from_command(command=["./compute.py", "15"], num_tasks=4,␣
↪num_nodes=2, cores_per_task=2)
compute_jobspec.cwd = os.getcwd()
compute_jobspec.environment = dict(os.environ)
```

- Using the executor as a context manager (`with FluxExecutor() as executor`) ensures it shuts down properly.
- `executor.submit(jobspec)` returns a future which completes when the job is done.
- `future.exception()` blocks until the future is complete and returns (not raises) an exception if the job was canceled or was otherwise prevented from execution. Otherwise the method returns `None`.

- `future.result()` blocks until the future is complete and returns the return code of the job. If the job succeeded, the return code will be 0.

12.4 Python Asynchronous Bulk Job Submission

Parts (a) and (b) demonstrate different implementations of the same basic use-case—submitting large numbers of jobs to Flux. For simplicity, in these examples all of the jobs are identical.

In part (a), we use the `flux.job.submit_async` and `flux.job.wait` functions to submit jobs and wait for them. In part (b), we use the `FluxExecutor` class, which offers a higher-level interface. It is important to note that these two different implementations deal with very different kinds of futures. The executor's futures fulfill in the background and callbacks added to the futures may be invoked by different threads; the `submit_async` futures do not fulfill in the background, callbacks are always invoked by the same thread that added them, and sharing the futures among threads is not supported.

12.4.1 Setup - Downloading the Files

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/async-bulk-job-submit
```

12.4.2 Part (a) - Using `submit_async`

Description: Asynchronously submit jobspec files from a directory and wait for them to complete in any order

1. Allocate three nodes from a resource manager:

```
salloc -N3 -ppdebug
```

1. Make a **jobs** directory:

```
mkdir jobs
```

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Store the jobspec of a `sleep 0` job in the **jobs** directory:

```
flux mini run --dry-run -n1 sleep 0 > jobs/0.json
```

1. Copy the jobspec of **job0** 1024 times to create a directory of 1025 `sleep 0` jobs:

```
for i in `seq 1 1024`; do cp jobs/0.json jobs/${i}.json; done
```

1. Run the **bulksubmit.py** script and pass all jobspec in the **jobs** directory as an argument with a shell glob `jobs/*.json`:

```
./bulksubmit.py jobs/*.json
```

```
bulksubmit: Starting...
bulksubmit: submitted 1025 jobs in 3.04s. 337.09job/s
bulksubmit: First job finished in about 3.089s
|| 100.0% (29.4 job/s)
bulksubmit: Ran 1025 jobs in 34.9s. 29.4 job/s
```

12.4.3 Notes to Part (a)

- `h = flux.Flux()` creates a new Flux handle which can be used to connect to and interact with a Flux instance.
- `job_submit_async(h, jobspec.read(), waitable=True).then(submit_cb)` submits a jobspec, returning a future which will be fulfilled when the submission of this job is complete.

`.then(submit_cb)`, called on the returned future, will cause our callback `submit_cb()` to be invoked when the submission of this job is complete and a jobid is available. To process job submission RPC responses and invoke callbacks, the flux reactor for handle `h` must be run:

```
if h.reactor_run() < 0:
    h.fatal_error("reactor start failed")
```

The reactor will return automatically when there are no more outstanding RPC responses, i.e., all jobs have been submitted.

- `job.wait(h)` waits for any job submitted with the `FLUX_JOB_WAITABLE` flag to transition to the **INACTIVE** state.

12.4.4 Part (b) - Using FluxExecutor

Description: Asynchronously submit a single command repeatedly

If continuing from part (a), skip to step 3.

1. Allocate three nodes from a resource manager:

```
salloc -N3 -ppdebug
```

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Run the **bulksubmit_executor.py** script and pass the command (`/bin/sleep 0` in this example) and the number of times to run it (default is 100):

```
./bulksubmit_executor.py -n200 /bin/sleep 0
```

```
bulksubmit_executor: submitted 200 jobs in 0.45s. 441.15job/s
bulksubmit_executor: First job finished in about 1.035s
|| 100.0% (24.9 job/s)
bulksubmit_executor: Ran 200 jobs in 8.2s. 24.4 job/s
```

12.5 Using Flux Job Status and Control API

12.5.1 Description: Submit job bundles, get event updates, and wait until all jobs complete

Setup

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/job-status-control
```

Execution

1. Allocate three nodes from a resource manager:

```
salloc -N3 -p pdebug
```

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Run the bookkeeper executable along with the number of jobs to be submitted (if no size is specified, 6 jobs are submitted: 3 instances of **compute.py**, and 3 instances of **io-forwarding,py**):

```
./bookkeeper.py 2
```

```
bookkeeper: all jobs submitted
bookkeeper: waiting until all jobs complete
job 39040581632 triggered event 'submit'
job 39040581633 triggered event 'submit'
job 39040581632 triggered event 'depend'
job 39040581632 triggered event 'priority'
job 39040581632 triggered event 'alloc'
job 39040581633 triggered event 'depend'
job 39040581633 triggered event 'priority'
job 39040581633 triggered event 'alloc'
job 39040581632 triggered event 'start'
job 39040581633 triggered event 'start'
job 39040581632 triggered event 'finish'
job 39040581633 triggered event 'finish'
job 39040581633 triggered event 'release'
job 39040581633 triggered event 'free'
job 39040581633 triggered event 'clean'
job 39040581632 triggered event 'release'
job 39040581632 triggered event 'free'
job 39040581632 triggered event 'clean'
bookkeeper: all jobs completed
```

12.5.2 Notes

- The following constructs a job request using the **JobspecV1** class with customizable parameters for how you want to utilize the resources allocated for your job:

```
compute_jobreq = JobspecV1.from_command(
    command=["./compute.py", "10"], num_tasks=4, num_nodes=2, cores_per_task=2
)
compute_jobreq.cwd = os.getcwd()
compute_jobreq.environment = dict(os.environ)
```

- `with FluxExecutor() as executor:` creates a new `FluxExecutor` which can be used to submit jobs, wait for them to complete, and get event updates. Using the executor as a context manager (`with ... as ...:`) ensures it is shut down properly.
- `executor.submit(compute_jobreq)` returns a `concurrent.futures.Future` subclass which completes when the underlying job is done. The jobid of the underlying job can be fetched with the `.jobid([timeout])` method (which waits until the jobid is ready).
- Throughout the course of a job, various events will occur to it. `future.add_event_callback(event, event_callback)` adds a callback which will be invoked when the given event occurs.

12.6 Job Cancellation

12.6.1 Description: Cancel a running job

Setup

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/job-cancel
```

Execution

1. Launch the submitter script:

`./submitter.py $(flux resource list -no {ncores} --state=up)`

note: for older versions of Flux, you might need to instead run: `./submitter.py $(flux hwloc info | awk '{print $3}')`

```
Submitted 1st job: 2241905819648
Submitted 2nd job: 2258951471104

First submitted job status (2241905819648) - RUNNING
Second submitted job status (2258951471104) - PENDING

Canceled first job: 2241905819648

First submitted job status (2241905819648) - CANCELED
Second submitted job status (2258951471104) - RUNNING
```

12.6.2 Notes

- `f = flux.Flux()` creates a new Flux handle which can be used to connect to and interact with a Flux instance.
- `flux.job.submit(f, sleep_jobspec, waitable=True)` submits a jobspec, returning a job ID that can be used to interact with the submitted job.
- `flux.job.cancel(f, jobid)` cancels the job.
- `flux.job.wait_async(f, jobid)` will wait for the job to complete (or in this case, be canceled). It returns a Flux future, which can be used to process the result later. Only jobs submitted with `waitable=True` can be waited for.

12.7 Using Events with Separate Nodes

12.7.1 Description: Using events to synchronize compute and io-forwarding jobs running on separate nodes

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/synchronize-events
```

1. `salloc -N3 -ppdebug`
2. `srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out`
3. `flux mini submit --nodes=2 --ntasks=4 --cores-per-task=2 ./compute.lua 120`

Output - `225284456448`

1. `flux mini submit --nodes=1 --ntasks=1 --cores-per-task=2 ./io-forwarding.lua 120`

Output - `344889229312`

1. List running jobs:

`flux jobs`

```
JOBID    USER     NAME       ST NTASKS NNODES  RUNTIME RANKS
ƒA4TgT7d moussa1  io-forward  R      1      1   4.376s 2
ƒ6vEcj7M moussa1  compute.lu  R      4      2   11.51s [0-1]
```

1. Attach to running or completed job output:

`flux job attach ƒ6vEcj7M`

```
Block until we hear go message from the an io forwarder
Block until we hear go message from the an io forwarder
Recv an event: please proceed
Recv an event: please proceed
Will compute for 120 seconds
Will compute for 120 seconds
Block until we hear go message from the an io forwarder
Block until we hear go message from the an io forwarder
Recv an event: please proceed
```

(continues on next page)

(continued from previous page)

```
Recv an event: please proceed
Will compute for 120 seconds
Will compute for 120 seconds
```

12.8 KVS Python Binding Example

12.8.1 Description: Use the KVS Python interface to store user data into KVS

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/kvs-python-bindings
```

1. Launch a Flux instance by running `flux start`, redirecting log messages to the file `out` in the current directory:

```
flux start -s 1 -o,-S,log-filename=out
```

1. Submit the Python script:

```
flux mini submit -N 1 -n 1 ./kvsput-usrdata.py
```

```
6705031151616
```

1. Attach to the job and view output:

```
flux job attach 6705031151616
```

```
hello world
hello world again
```

1. Each job is run within a KVS namespace. `FLUX_KVS_NAMESPACE` is set, which is automatically read and used by the KVS operations in the handle. To take a look at the job's KVS, convert its job ID to KVS:

```
flux job id --from=dec --to=kvs 6705031151616
```

```
job.0000.0619.2300.0000
```

1. The keys for this job will be put at the root of the namespace, which is mounted under "guest". To get the value stored under the first key "usrdata":

```
flux kvs get job.0000.0619.2300.0000.guest.usrdata
```

```
"hello world"
```

1. Get the value stored under the second key "usrdata2":

```
flux kvs get job.0000.0619.2300.0000.guest.usrdata2
```

```
"hello world again"
```

12.8.2 Notes

- `f = flux.Flux()` creates a new Flux handle which can be used to connect to and interact with a Flux instance.
- `kvs.put()` places the value of *udata* under the key **"usrdata"**. Once the key-value pair is put, the change must be committed with `kvs.commit()`. The value can then be retrieved with `kvs.get()`
- `kvs.get()` on a directory will return a KVSDir object which supports the `with` compound statement. `with` guarantees a commit is called on the directory.

12.9 Job Ensemble Submitted with a New Flux Instance

12.9.1 Description: Launch a flux instance and submit one instance of an io-forwarding job and 50 compute jobs, each spanning the entire set of nodes.

12.9.2 Setup

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/job-ensemble
```

12.9.3 Execution

1. `salloc -N3 -ppdebug`
2. `cat ensemble.sh`

```
#!/usr/bin/env sh

NJOBS=10
MAXTIME=$(expr ${NJOBS} + 2)
JOBIDS=""

JOBIDS=$(flux mini submit --nodes=1 --ntasks=1 --cores-per-task=2 ./io-forwarding.lua
↪${MAXTIME})
for i in `seq 1 ${NJOBS}`; do
    JOBIDS="${JOBIDS} $(flux mini submit --nodes=2 --ntasks=4 --cores-per-task=2 ./
↪compute.lua 1)"
done

flux jobs
flux queue drain

# print mock-up prevenance data
for i in ${JOBIDS}; do
    echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    echo "Jobid: ${i}"
    KVSJOBID=$(flux job id --from=dec --to=kvs ${i})
    flux kvs get ${KVSJOBID}.R | jq
done
```

1. `srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out ./ensemble.sh`

```
JOBID          USER      NAME       STATE     NTASKS NNODES   RUNTIME RANKS
1721426247680 fluxuser  compute.lu RUN            4      2    0.122s [1-2]
1718322462720 fluxuser  compute.lu RUN            4      2    0.293s [0,2]
1715201900544 fluxuser  compute.lu RUN            4      2    0.481s [0-1]
1712299442176 fluxuser  compute.lu RUN            4      2    0.626s [1-2]
1709296320512 fluxuser  compute.lu RUN            4      2    0.885s [0,2]
1706293198848 fluxuser  compute.lu RUN            4      2    1.064s [0-1]
1691378253824 fluxuser  io-forward RUN            1      1    1.951s 0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Jobid: 1691378253824
{
  "version": 1,
  "execution": {
    "R_lite": [
      {
        "rank": "0",
        "children": {
          "core": "0-1"
        }
      }
    ]
  }
}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Jobid: 1694414929920
{
  "version": 1,
  "execution": {
    "R_lite": [
      {
        "rank": "1-2",
        "children": {
          "core": "0-3"
        }
      }
    ]
  }
}
.
.
.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Jobid: 1721426247680
{
  "version": 1,
  "execution": {
    "R_lite": [
      {
        "rank": "1-2",
        "children": {
          "core": "8-11"
        }
      }
    ]
  }
}
```

12.10 Hierarchical Launching

12.10.1 Description: Launch an ensemble of sleep 0 tasks

Setup

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/hierarchical-launching
```

Execution

1. `salloc -N3 -ppdebug`
2. `srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out`
3. `./parent.sh`

```
Mon Nov 18 15:31:08 PST 2019
13363018989568
13365166473216
13367095853056
First Level Done
Mon Nov 18 15:34:13 PST 2019
```

12.10.2 Notes

- You can increase the number of jobs by increasing `NCORES` in `parent.sh` and `NJOBS` in `ensemble.sh`.

12.11 Using a Flux Comms Module

12.11.1 Description: Use a Flux comms module to communicate with job elements

Setup

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/comms-module
```

Execution

1. `salloc -N3 -ppdebug`
2. Point to `flux-core`'s `pkgconfig` directory:

| Shell | Command | | --- | ------- | | tcsh | `setenv PKG_CONFIG_PATH <FLUX_INSTALL_PATH>/lib/pkgconfig` | | bash/zsh | `export PKG_CONFIG_PATH='<FLUX_INSTALL_PATH>/lib/pkgconfig'` |

1. `make`
2. Add the directory of the modules to `FLUX_MODULE_PATH`; if the module was built in the current dir:

```
export FLUX_MODULE_PATH=${FLUX_MODULE_PATH}:$(pwd)
```

1. `srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out`
2. `flux submit -N 2 -n 2 ./compute.lua 120`
3. `flux submit -N 1 -n 1 ./io-forwarding.lua 120`

12.12 A Data Conduit Strategy

12.12.1 Description: Use a data stream to send packets through

Setup

If you haven't already, download the files and change your working directory:

```
$ git clone https://github.com/flux-framework/flux-workflow-examples.git
$ cd flux-workflow-examples/data-conduit
```

Execution

1. Allocate three nodes from a resource manager:

```
salloc -N3 -ppdebug
```

1. Point to `flux-core`'s `pkgconfig` directory:

| Shell | Command | | --- | ------- | | tcsh | `setenv PKG_CONFIG_PATH <FLUX_INSTALL_PATH>/lib/pkgconfig` | | bash/zsh | `export PKG_CONFIG_PATH='<FLUX_INSTALL_PATH>/lib/pkgconfig'` |

1. `make`
2. Add the directory of the modules to `FLUX_MODULE_PATH`, if the module was built in the current directory:

```
export FLUX_MODULE_PATH=${FLUX_MODULE_PATH}:$(pwd)
```

1. Launch a Flux instance on the current allocation by running `flux start` once per node, redirecting log messages to the file `out` in the current directory:

```
srun --pty --mpi=none -N3 flux start -o,-S,log-filename=out
```

1. Submit the **datastore** script:

```
flux submit -N 1 -n 1 ./datastore.py
```

1. Submit and resubmit five **compute** scripts to send time data to **datastore**:

```
flux submit -N 1 -n 1 ./compute.py 1
flux submit -N 1 -n 1 ./compute.py 1
flux submit -N 1 -n 1 ./compute.py 1
flux submit -N 1 -n 1 ./compute.py 1
flux submit -N 1 -n 1 ./compute.py 1
```

1. Attach to the **datastore** job to see the data sent by the **compute.py** scripts:

```
flux job attach 1900070043648
```

```
Starting....
Module was loaded successfully...
finished initialize...
starting run()
Waiting for a packet
{u'test': 101}
Waiting for a packet
{u'test': 101, u'1578431137': u'os.time'}
Waiting for a packet
{u'test': 101, u'1578431137': u'os.time', u'1578431139': u'os.time'}
Waiting for a packet
{u'test': 101, u'1578431140': u'os.time', u'1578431137': u'os.time', u'1578431139': u
↪'os.time'}
Waiting for a packet
{u'test': 101, u'1578431140': u'os.time', u'1578431137': u'os.time', u'1578431139': u
↪'os.time', u'1578431141': u'os.time'}
Bye bye!
run finished...
```

12.12.2 Notes

- `f = flux.Flux()` creates a new Flux handle which can be used to connect to and interact with a Flux instance.
- `kvs.put()` places the value of *udata* under the key **"conduit"**. Once the key-value pair is put, the change must be committed with `kvs.commit()`. The value can then be retrieved with `kvs.get()`.
- `f.rpc()` creates a new RPC object consisting of a specified topic and payload (along with additional flags) that are exchanged with a Flux service.