Running Julia AGC on coffea.af@uchicago

Dec, 2023

Jerry Ling (Harvard University / ATLAS Experiment)

Pointers

- Talk by Atell-Yehor Krasnopolski
- Atell developed most of the code as a IRIS-HEP 2023 Fellow
- Atell was supervised by Alex Held and I

Step 0: Get Julia

General comment: Similar to Rustup, juliaup is the preferred way to manage Julia binary (when it doesn't come as part of the environment). It ensures official binary & serves as version multiplexer. CERN LCG is another option, when cvmfs is available.

AF@UChicago specific:

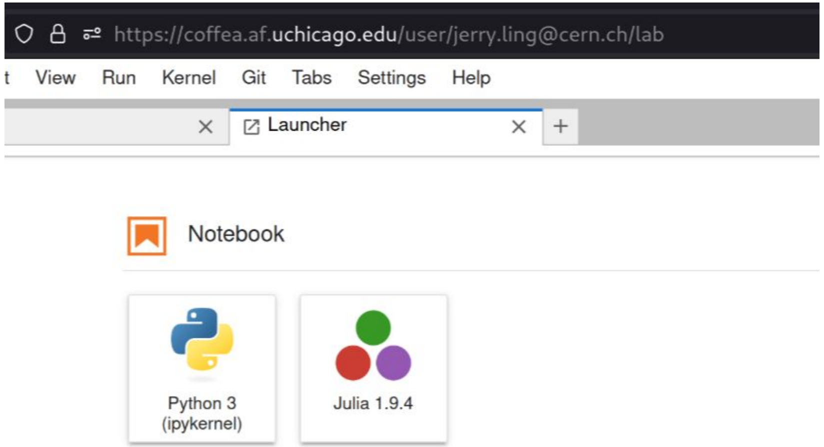
```
~$ mamba install juliaup
```

Optional Step 0.5: Set up Jupyter

General comment: The `IJulia.jl` package can re-use existing Jupyter(lab) instance by adding a kernel spec, or it will download its own Jupyterlab for you.

AF@UChicago specific:

```
$ juliaup add 1.9
$ julia
# press ] to enter Pkg mode
(@v1.9) pkg> add IJulia
```


https://coffea.af.uchicago.edu/user/jerry.ling@cern.ch/lab

Step 1: Set up Julia AGC

General comment: One can either `add` or `dev` from the Github url. Or clone first and do it locally. We anticipate many hotfixes needed, so we will just `dev` it.

AF@UChicago specific:

- Manifest.toml records exact deps
- If we need to fix things, we can `]dev …` from here to update dependency

```
$ git clone https://github.com/Moelf/LHC_AGC.jl
$ cd LHC_AGC.jl
$ julia --project=.

(LHC_AGC) pkg> st
Project LHC_AGC v0.1.0
Status `~/.julia/dev/LHC_AGC/Project.toml`
  [6e4b80f9] BenchmarkTools v1.3.2
^ [13f3f980] CairoMakie v0.10.12
  [34f1f09b] ClusterManagers v0.4.5
  [861a8166] Combinatorics v1.0.2
  [31c24e10] Distributions v0.25.103
  ...
```

Outline for Step 2+

We encountered a few problems with the current coffea.af@uchicago setup, some problems have workaround, some are actually “blocking” right now.

I will first briefly describe how Julia’s built-in `Distributed.jl` communication model and quickly introduce the `ClusterMannagers.jl` package, which is a thin wrapper to facilitate main-worker communication on various HPC scheduler.

Then I will describe the assumptions CM.jl makes for a HTCondor, some workaround, and why it doesn’t work all the way. Finally, I will side step the “broken” steps by using login nodes on af@uchicago.

Distributed.jl and ClusterManagers.jl

General comment: Julia has memory-shared multi-threading. Distributed.jl, on the other hand, is a standard library for supporting multi-processing (both local and remote processes). These worker processes can communicate with the main process in various ways, including local FIFO file, SSH/Sockets, telnet etc. The ClusterManagers.jl is a thin wrapper to facilitate spawning and connecting remote worker process back to main process, on various HPC scheduler (qsub, htcondor, slurm etc.)

Of course, if you're real HPC, you probably want to use MPI.jl directly.

State of ClusterManagers.jl

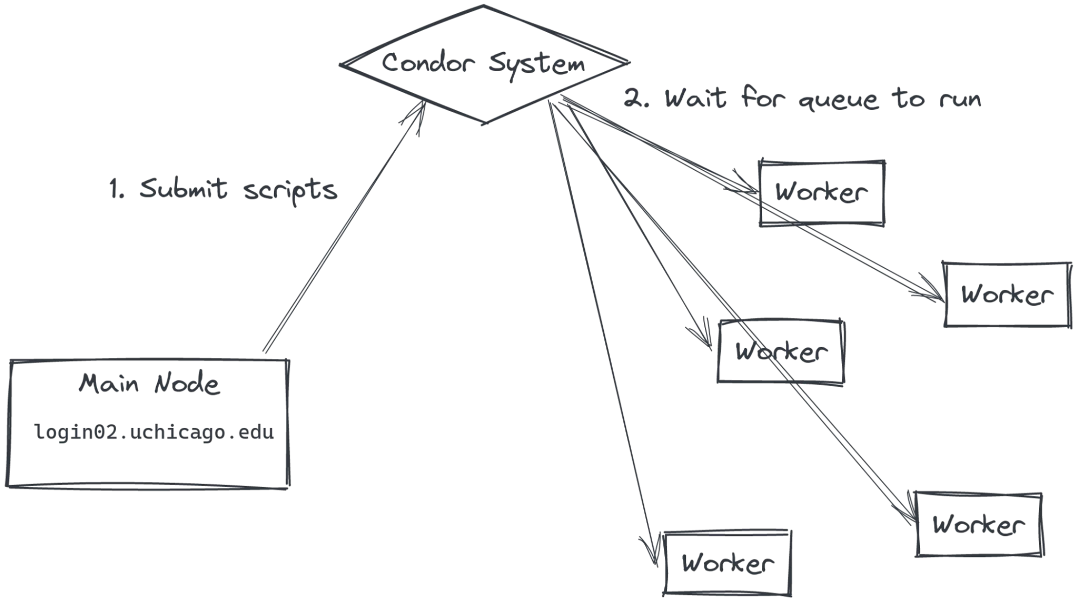
General comment: Basically, think of CM.jl as a tiny subset of Dask, its sole purpose is to provide a ~uniform interface to spawn remote process and connect back to main process, nothing else. It is also undermaintained – it's hard to "integration test" a dozen of different HPC setup, especially given no two HTCondor setups are identical.

Historically, HEP people have contributed to various fixes on HPC setup they happen to use (#184, #160, #157). But it's nowhere near the robustness Dask has and has ~0 dedicated resource.

Communication model of ClusterManagers.jl

Specifically to HTCondor, it will generate `.sh` and `.sub` files according to your Julia environment and user provided options.

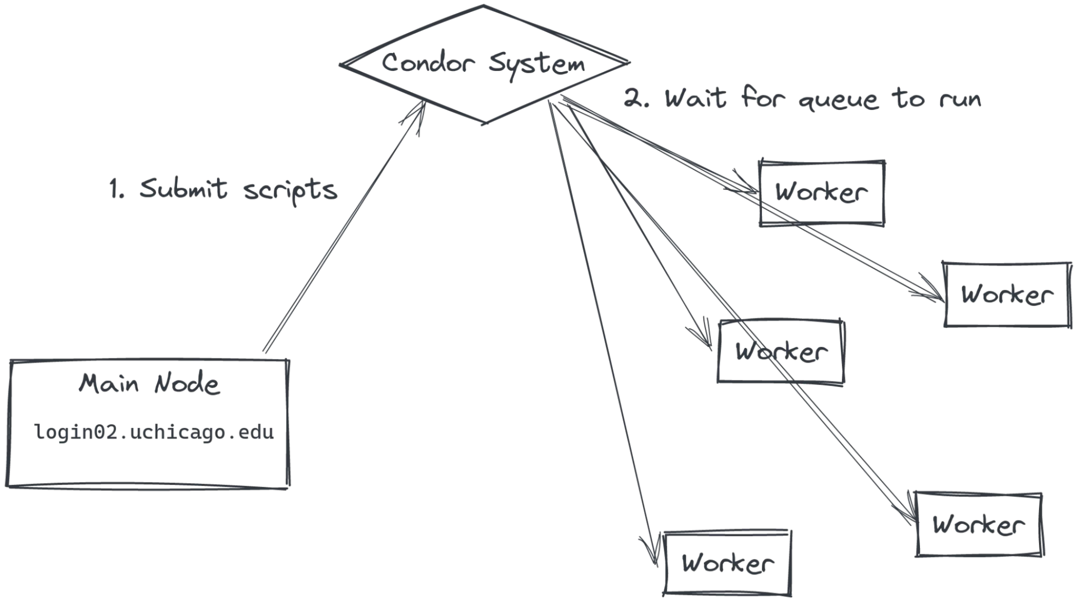
Here comes the assumptions this package currently make, for HTCondor setup.



Communication model of ClusterManagers.jl

Assumption: The same Julia executable is accessible from worker nodes.

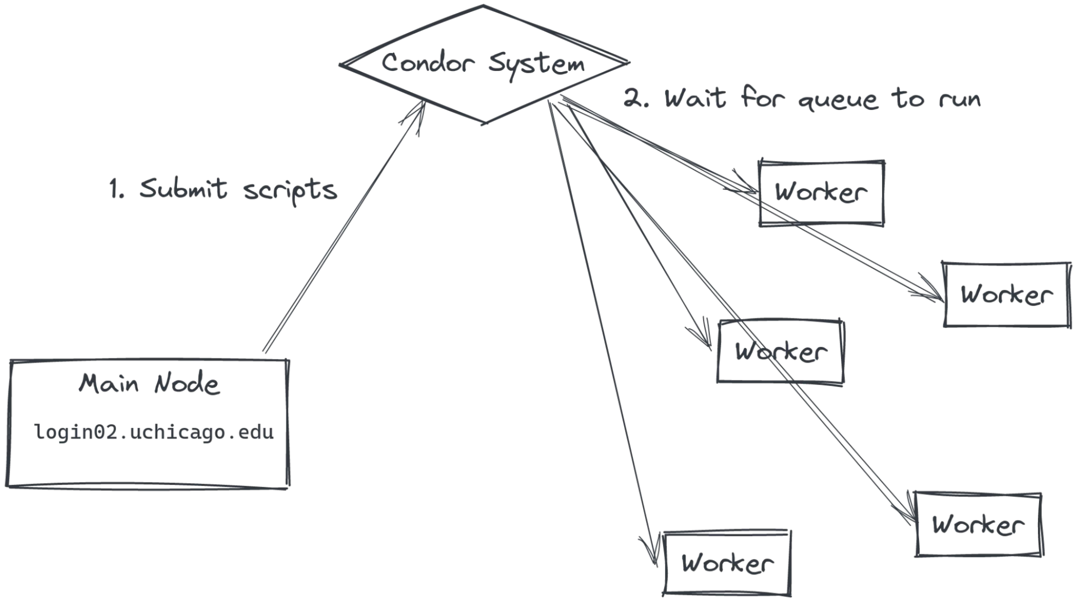
❌; the workaround is to create a mirror `.juliaup` setup in `/data/jiling`. A follow-up problem is now remote Julia worker, each needs to JIT to a different location.



Communication model of ClusterManagers.jl

Assumption: The directory job submitted from can be `cd` to from remote worker.

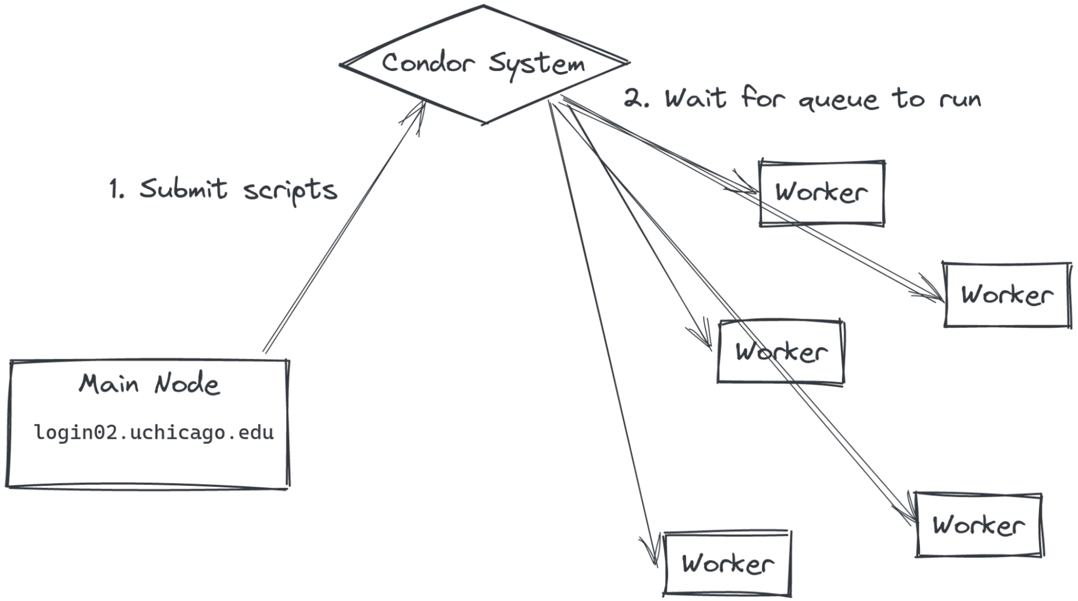
❌; similar to first assumption, the workaround is to inject `initialdir=/data/jiling/` into the `.sub` file.



Communication model of ClusterManagers.jl

Assumption: `transfer_input_files` functions.

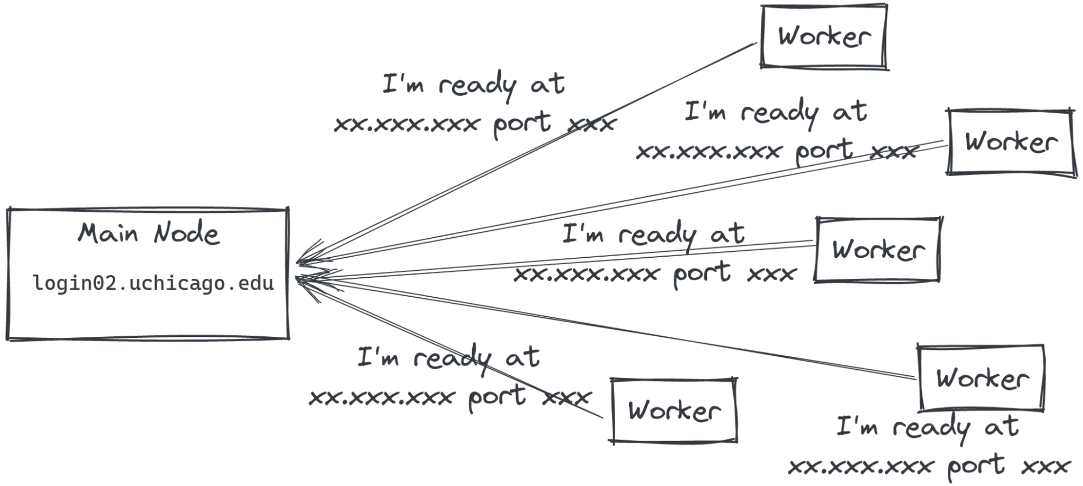
Currently, does not work our particular HTCondor system. Expected to be fixed soon.



Communication model of ClusterManagers.jl

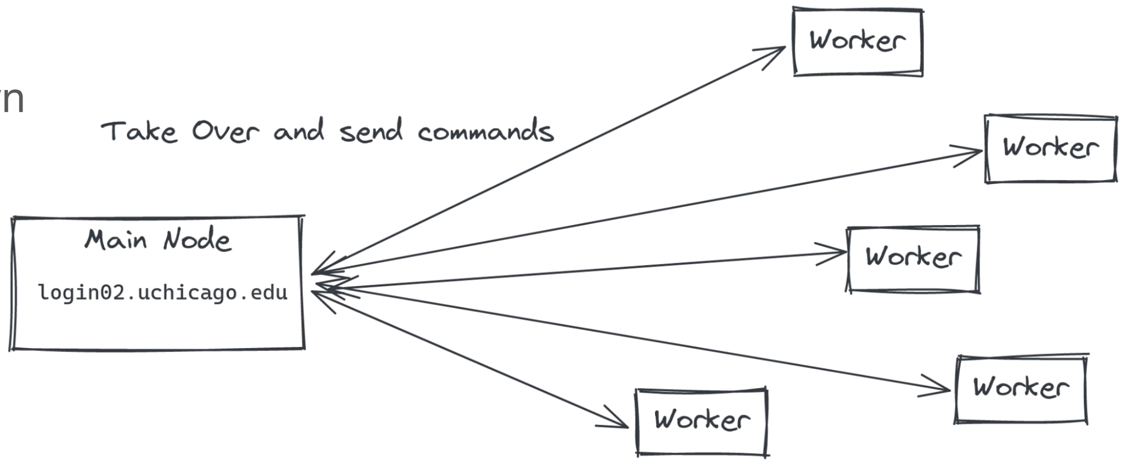
The next step in a “normal” workflow is for worker processes to connect back to the main node.

This communication can happen in one of many ways. HTCondor setup uses telnet, which isn’t ideal, it also expects *some* port to open, which may not be the case.



Communication model of ClusterManagers.jl

The last step in a normal workflow is just when main node takes over and can perform code loading and spawn tasks on them.



“Normal” Step 2

Because submitting from coffea.casa is broken, we demonstrate normal workflow from login nodes:

```
julia> using Revise, Distributed, ClusterManagers, LHC_AGC
[ Info: Precompiling Revise [295af30f-e4ad-537b-8983-00126c2a3abe]
[ Info: Precompiling LHC_AGC [7d6523ac-594c-471e-83f0-329b402061cb]

julia> addprocs(HTCManager(2); extrajdl=["+queue=\"short\""]);
Waiting for 2 workers: 1 2 .

julia> @fetchfrom 1 gethostname()
"login01.af.uchicago.edu"

julia> @fetchfrom 2 gethostname()
"c001.af.uchicago.edu"
```

Step 3: Prepare tasks

Like many HEP analyses, AGC is naively parallelizable, and an obvious way to partition the tasks is using the combination of <process>-<file path>-<variation>.

```
julia> LHC_AGC.get_tasks(:ttbar; n_files_max_per_sample=1)
5-element Vector{LHC_AGC.AnalysisTask}:
 LHC_AGC.AnalysisTask(:ttbar, "/data/alheld/AGC/datasets/nanoAOD/TT_TuneEE5C_13TeV-powheg-herwigpp
/cmsopendata2015_ttbar_19999_PU25nsData2015v1_76X_mcRun2_asymptotic_v12-v1_10000_0000.root", 1.88165112743392,
:PS_var)
 LHC_AGC.AnalysisTask(:ttbar, "/data/alheld/AGC/datasets/nanoAOD/TT_TuneCUETP8M1_13TeV-powheg-pythia8
/cmsopendata2015_ttbar_19980_PU25nsData2015v1_76X_mcRun2_asymptotic_v12_ext3-v1_00000_0000.root", 1.8475328155584265,
:nominal)
 LHC_AGC.AnalysisTask(:ttbar, "/data/alheld/AGC/datasets/nanoAOD/TT_TuneCUETP8M1_13TeV-powheg-scaledown-pythia8
/cmsopendata2015_ttbar_19983_PU25nsData2015v1_76X_mcRun2_asymptotic_v12_ext3-v1_00000_0000.root", 1.9439411850048256,
:scaledown)
 LHC_AGC.AnalysisTask(:ttbar, "/data/alheld/AGC/datasets/nanoAOD/TT_TuneCUETP8M1_13TeV-powheg-scaleup-pythia8
/cmsopendata2015_ttbar_19985_PU25nsData2015v1_76X_mcRun2_asymptotic_v12_ext3-v1_10000_0000.root", 1.9280590914956264,
:scaleup)
 LHC_AGC.AnalysisTask(:ttbar, "/data/alheld/AGC/datasets/nanoAOD/TT_TuneCUETP8M1_13TeV-amcatnlo-pythia8
/cmsopendata2015_ttbar_19978_PU25nsData2015v1_76X_mcRun2_asymptotic_v12_ext1-v1_60000_0000.root", 1.8170692216981132,
:ME_var)
```

Step 4: Parallel map-reduce

Given the collection of tasks (over all process, files, variations), we simply want to run all of them. `pmap` is built-in and will use all processes. For real application, you probably want to:

- ❖ Use `Parallelism.robust_pmap()` to guard against worker nodes dying
- ❖ Add progress bar / logging

```
julia> pmap(task_to_hists, all_tasks)
```

Step 5: Merge

Conceptually, each “task” will return a collection of histograms, keyed by

`<process>_<signal region>_<variation>`

The “merging” rules of these dictionaries are captured nicely by (as long as + is defined for histograms):

```
julia> a = Dict(:ttbar_var1_SR1 => 1.0, :ttbar_var1_SR2=>1.0);
julia> b = Dict(:ttbar_nominal_SR1 => 1.1, :ttbar_nominal_SR2=>1.0);
julia> c = Dict(:ttbar_var1_SR1 => 2.0, :ttbar_var1_SR2=>2.0);

julia> reduce(mergewith(+), [a,b,c])
Dict{Symbol, Float64} with 4 entries:
  :ttbar_nominal_SR2 => 1.0
  :ttbar_nominal_SR1 => 1.1
  :ttbar_var1_SR1    => 3.0
  :ttbar_var1_SR2    => 3.0
```

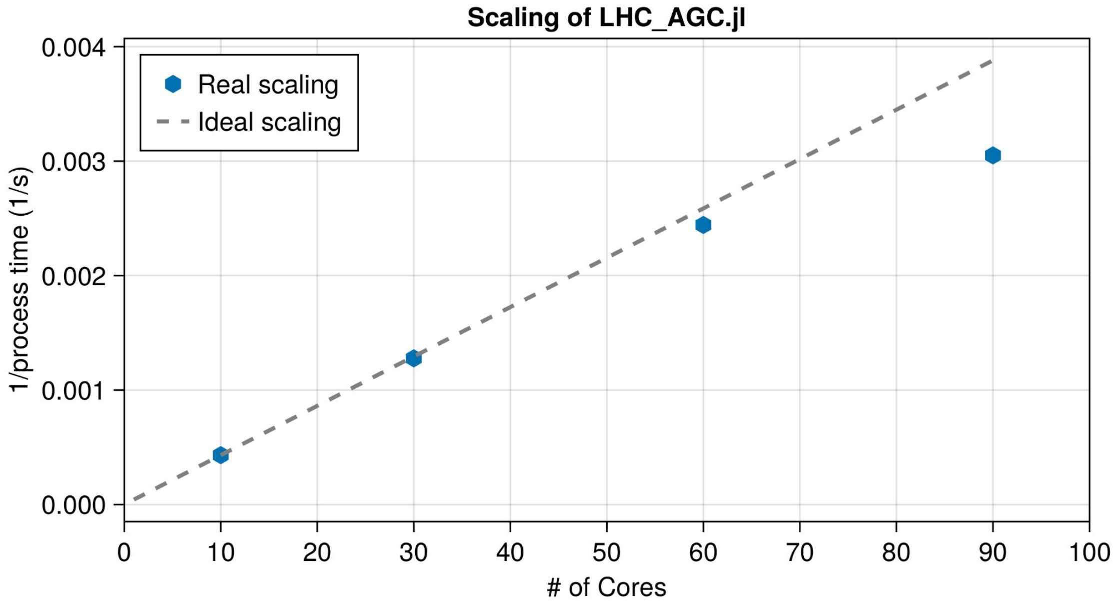
Step 6: Form “workspace” and inference

Currently, hard-coded, dumps the dictionary of histograms to a pyhf-compatible JSON.

```
help?> LHC_AGC.generate_workspace_file
  generate_workspace_file(all_hists::Dict, filename, real_data; rebin2=true, systematics=false)
```

Interactive distributed analysis

- Embarrassingly parallel workload scales nicely
- AF UChicago has **25** physical nodes, fall off when network/storage bottle necked.



Backup

Backup

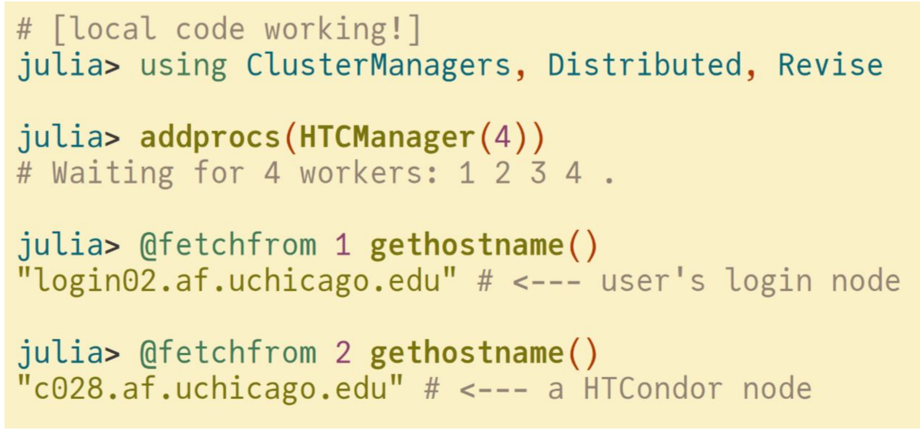
```
#!/bin/sh
/data/jiling/.juliaup/bin/julia  -e 'using Distributed; start_worker("0D6ZuoPE1950P3wf")' | /usr/bin/telnet
jupyter-jerry-2eling-40cern-2ech 8799
```

```
executable = /bin/bash
arguments = ./julia-1412.sh
universe = vanilla
should_transfer_files = yes
transfer_input_files = /home/atlas-coffea/.julia-htc/julia-1412.sh
Notification = Error
initialdir=/data/jiling/
+queue="short"
output = /home/atlas-coffea/.julia-htc/julia-1412-1.o
error= /home/atlas-coffea/.julia-htc/julia-1412-1.e
queue
```

Interactive distributed analysis

End users' partial wish list for running analysis on cluster:

- Smooth local session -> cluster
- No wait for compilation
- Revise code without re-submitting



```
# [local code working!]
julia> using ClusterManagers, Distributed, Revise

julia> addprocs(HTCManager(4))
# Waiting for 4 workers: 1 2 3 4 .

julia> @fetchfrom 1 gethostname()
"login02.af.uchicago.edu" # <--- user's login node

julia> @fetchfrom 2 gethostname()
"c028.af.uchicago.edu" # <--- a HTCondor node
```

Interactive distributed analysis

End users' partial wish list for running analysis on cluster:

- Smooth local session -> cluster
- No wait for compilation
- Revise code without re-submitting

```
# [local code working!]
julia> using ClusterManagers, Distributed, Revise

julia> addprocs(HTCManager(4))
# Waiting for 4 workers: 1 2 3 4 .

julia> @fetchfrom 1 gethostname()
"login02.af.uchicago.edu" # <--- user's login node

julia> @fetchfrom 2 gethostname()
"c028.af.uchicago.edu" # <--- a HTCondor node

julia> @everywhere using WVZAnalysis
```

Interactive distributed analysis

End users' partial wish list for running analysis on cluster:

- ❖ Smooth local session -> cluster
- ❖ No wait for compilation
- ❖ Revise code without re-submitting

```
# [local code working!]
julia> using ClusterManagers, Distributed, Revise

julia> addprocs(HTCManager(4))
# Waiting for 4 workers: 1 2 3 4 .

julia> @fetchfrom 1 gethostname()
"login02.af.uchicago.edu" # <--- user's login node

julia> @fetchfrom 2 gethostname()
"c028.af.uchicago.edu" # <--- a HTCondor node

julia> @everywhere using WVZAnalysis

julia> run_analysis(..)

# Result looks wrong!
```

Interactive distributed analysis

End users' partial wish list for running analysis on cluster:

- ❖ Smooth local session -> cluster
- ❖ No wait for compilation
- ❖ Revise code without re-submitting

```
# [local code working!]
julia> using ClusterManagers, Distributed, Revise

julia> addprocs(HTCManager(4))
# Waiting for 4 workers: 1 2 3 4 .

julia> @fetchfrom 1 gethostname()
"login02.af.uchicago.edu" # <--- user's login node

julia> @fetchfrom 2 gethostname()
"c028.af.uchicago.edu" # <--- a HTCondor node

julia> @everywhere using WVZAnalysis

julia> run_analysis(..)

# Result looks wrong!

# [Edit source code]

julia> run_analysis(..)
```

Modified code re-compiled