GSPDA Mini-Software Workshop (Part 2)

“Hands-on” SWIF2 Farm Submission Example

Sept. 6, 2024

Brad Sawatzky

"Scientific Workflow Indefatigable Factotum"

- Or SWIF, to save time...
  - → Swif is a workflow manager designed at JLab
  - → Interfaces intelligently with the JLab tape system and slurm, the job scheduler
    - » will efficiently pre-stage file dependencies
    - » will efficiently ensure output files are placed on tape
  - → Command line tool: swif2
  - → Web monitoring

- SWIF == Workflow manager
  - → provides tools to let you to submit, run, and monitor groups of jobs
    - » ie. a "workflow"
  - → provides information on individual jobs as well as aggregate information
  - → tracks success and failures and allows you to modify and resubmit failed jobs
    - » ie. outlier jobs that need more time, RAM, disk, etc

Simple Linux Utility For Resource Management

- Or Slurm, to save time…
  - → Open source compute cluster resource manager and job scheduler
  - → JLab Slurm Docs
    - » sinfo
    - » srun
    - » Useful env-vars
- Jobs run under a particular
  - → Slurm 'Account' (ie. group)
    - » Ask helpdesk@jlab.org or Compute Coordinator
  - → Slurm 'Partition'
    - » usually 'production' or 'priority'

slurm
workload manager

- Cluster/Job Scheduling
  - → takes care of nitty-gritty of actually running jobs on a given compute node
  - → tracks and enforces resource usage and requirements (RAM, disk, time, etc)
  - → enforces a fair balance of allocated farm time across our user base (Division, experiment, Hall, etc)
    - » "Fairshare" allocations are managed by Compute Coordinators, SciComp and Physics Division

General Farm Access

- See "New users start here"
  - → At top of scicomp.jlab.org
  - → See this excellent walkthrough for BRIC by Cameron Clarke too!
- You will need (in general):
  - → JLab CUE account
  - → Get a 2-factor token (ssh access)
  - → Get access to Farm/ifarm
    - » talk to Project lead and/or Compute Coordinator
    - » learn where to put your files (unix group access)
    - » get attached to a slurm group
  - → SciComp user certificate

Steps to getting started using the Physics Farm system at Jefferson Lab are listed below:

1. Obtain Farm/ifarm access.
2. Generate a SciComp user certificate for your account.
3. Please join the email list `jlab-scicomp-briefs@jlab.org` to get email about the status and updates of resources. This is highly recommended.
4. Considerable information about the status of the computing resources is available at the SciComp Portal: scicomp.jlab.org/scicomp/. On the entry page you can find information about the status of the various systems, as well as important announcements about new capabilities, current problems or planned outages. From the menu on the left you can get more detailed information about status and utilization, including reports by user or project for any arbitrary time interval.
5. For all other account questions please contact your Hall Compute Coordinator, the Physics Computing Coordinator Brad Sawatzky, or the Computer Center Help Desk

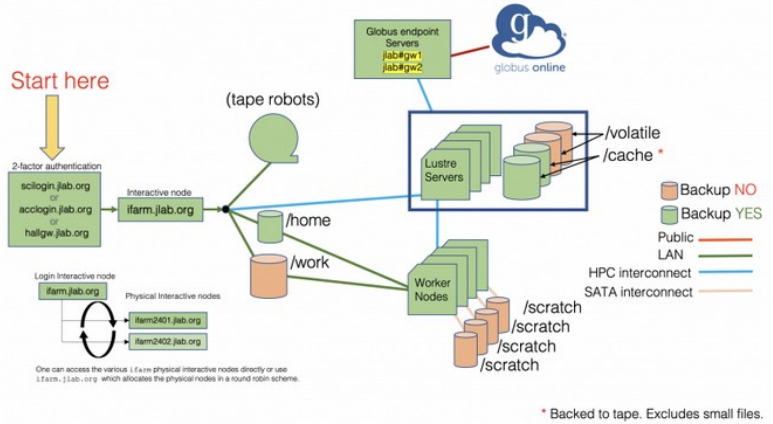


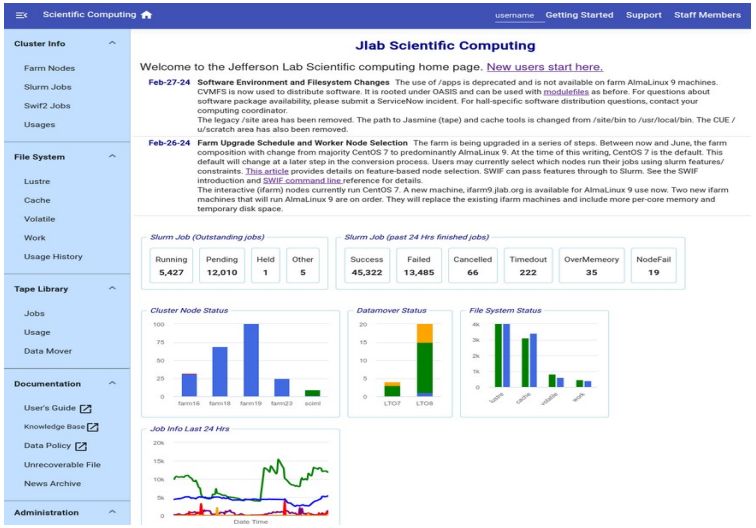
Figure 1. Jefferson Lab Farm Cluster Layout

- Access
- Create custom kernel for Jupyter Hub
- Interactive login machines
- Login to SciComp GPUs
- Start using JLab Jupyter Hub
- Network certificate
- File system layout

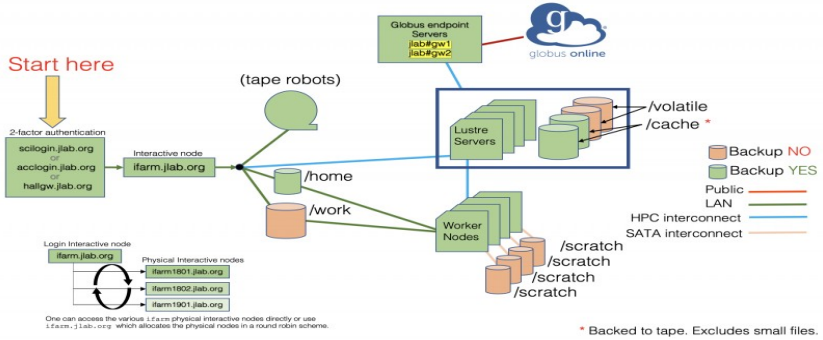
‹ Experimental Physics User's Guide up Access ›

Other Information Resources

- scicomp.jlab.org
  → SciComp web page
- scicomp-briefs
  → mailing list for JLab Scientific Computing



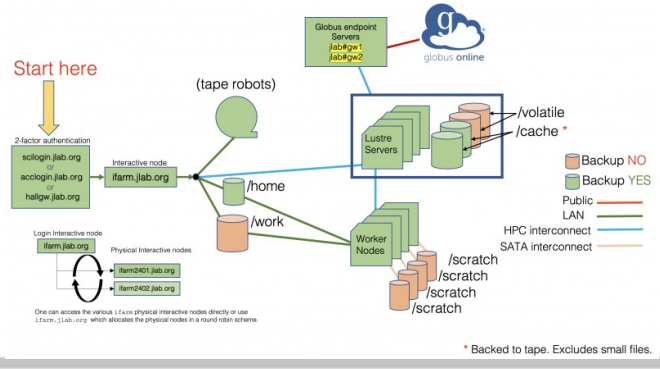
- Documentation links
  → Getting Started
  → SciComp Knowledge Base
  → CST User Portal
  → JLab Helpdesk
    » helpdesk@jlab.org
    » Incident Request



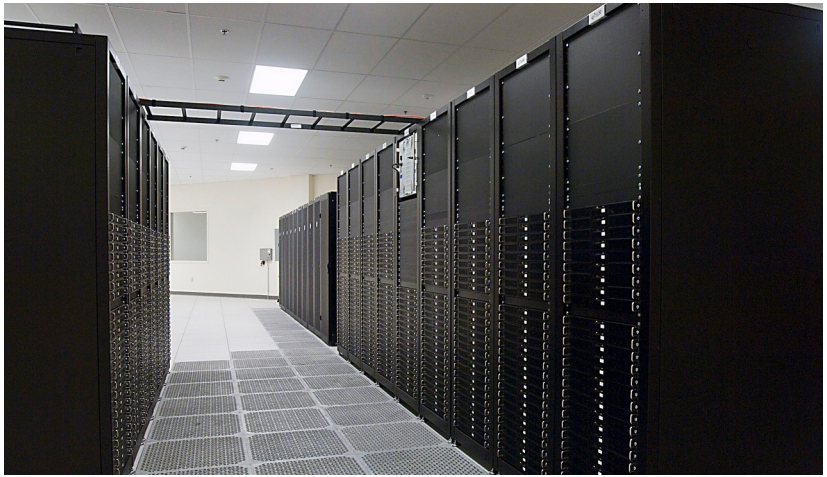
The Plan (TM)

- Develop and test an example compute job
- Develop and test a ‘steering script’ that you will use to
  - → setup your environment
  - → stage your code/exe,
  - → and run your code

  as a non-interactive farm job

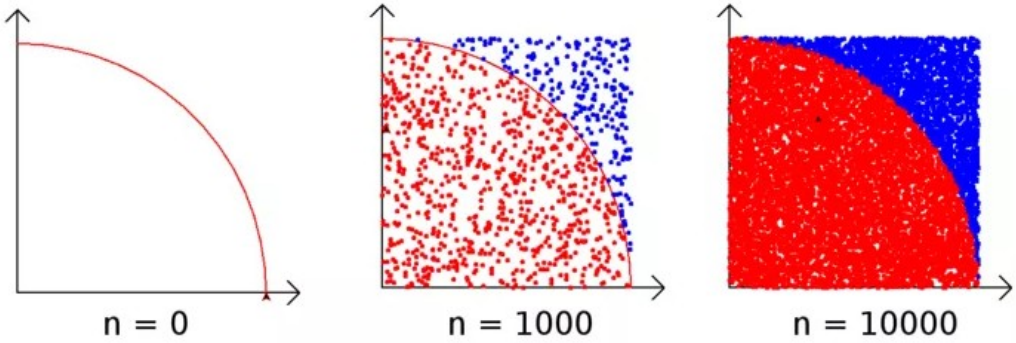


- Submit a few example jobs to the Farm using SWIF2
  - → monitor those jobs with some swif and slurm commands
  - → modify and re-run any failed jobs



The Plan (TM)

- We are going to compute pi using a simple (but not super efficient) Monte Carlo program
  - → Pick random points in a 1x1 square area and count the number that land within a radius of 1 from the origin
  - → Area of the square is 1x1
  - → Area of the ¼ circle is ¼ π 1^2
  - → So the ratio of the areas is ¼ π
  - → So lets throw N_{sample} points at the square and see what fraction of them are also in the circle



- $A_{circle} / A_{square} = ¼ \pi$
  - → $\pi = 4 \times hits_{circle} / hits_{square}$
  - → $\pi = 4 \times hits_{circle} / N_{samples}$
- More samples
  - → better coverage
  - → better accuracy (but takes longer)

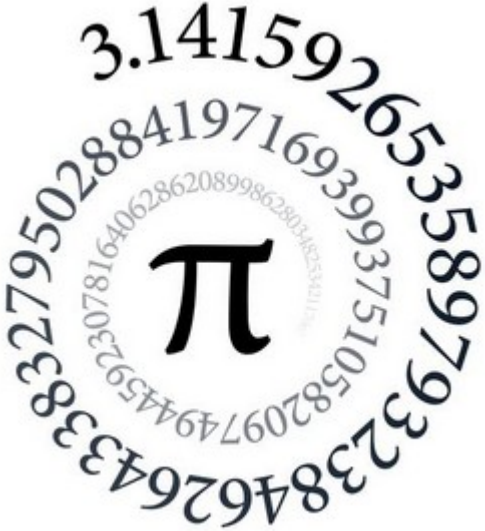
Grab the Example Code

- ssh <you>@login.jlab.org
- ssh ifarm

```
brads@aether 1021% git clone https://code.jlab.org/brads/workshop-sept2024.git
Cloning into 'workshop-sept2024'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), 4.47 KiB | 4.47 MiB/s, done.
brads@aether 1022% cd workshop-sept2024/
brads@aether [git:master] 1023% ls -lF
total 16
-rwxr-xr-x 1 brads brads 4321 Sep  4 00:01 job-script.sh*
-rwxr-xr-x 1 brads brads 2186 Sep  4 00:01 piece-of.py*
-rwxr-xr-x 1 brads brads 2053 Sep  4 00:01 swif-submission.sh*
brads@aether [git:master] 1024%
```

Dev Process: Test your executable!

- Test your code on the ifarm
  - → piece-of.py
    - » this does the actual calculation




```
brads@ifarm2401 1006% cd workshop-sept2024/
brads@ifarm2401 1007% ls
job-script.sh  output  output-work  piece-of.py  swif-submission.sh  SWS-pitest-brads
brads@ifarm2401 1008% ./piece-of.py -v 100000000
Running 100.0 M samples...
..................................................................................... 10.0 M : 3.1418976
..................................................................................... 20.0 M : 3.1410738
..................................................................................... 30.0 M : 3.141404
..................................................................................... 40.0 M : 3.1414287
..................................................................................... 50.0 M : 3.14158672
..................................................................................... 60.0 M : 3.1416396
..................................................................................... 70.0 M : 3.1417185142857145
..................................................................................... 80.0 M : 3.14172715
..................................................................................... 90.0 M : 3.1417816
..................................................................................... 100.0 M : 3.14185776

Number of samples:                  100000000 (100 M)
Number inside unit quarter-circle:   78546444
Pi is roughly:                      3.141858
brads@ifarm2401 1009%
```

Dev Process: Test your job-script!

- Create and test a ‘job-script’ wrapper on ifarm
  - → job-script.sh <arguments>
  - → this ‘wrapper’ will be invoked for every farm job
  - → its purpose is to:
    - » definitively set-up your working environment
    - » load software dependencies
    - » check directories, etc
    - » configure and run your executable for each discrete job in your workflow
      - – update run number
      - – update config files
      - – update event count
      - – etc…
- Test this script in a clean ‘working dir’ to replicate farm node environment
  - → we will use /scratch/$USER/workshop-test/

```
brads@ifarm2401 1009% cd /scratch/$USER
brads@ifarm2401 1010% mkdir test
brads@ifarm2401 1011% cd test
brads@ifarm2401 1012% ~/workshop-sept2024/job-script.sh 100000000

-- Environment --------------------------------------------
SHELL=/bin/bash
LESS=-x4 -j4 -i -w  -e -M -X -R -P%t?f%f :stdin .?pb%pb\%:?lbLine %lb:?bbByte %bb:-...
HISTCONTROL=ignoreboth
HOSTNAME=ifarm2401.jlab.org
HISTSIZE=1000
PERLLIB=/home/brads/lib/perl
BSTINPUTS=:/home/brads/LaTeX/Bibstyles/
DEBFULLNAME=Brad Sawatzky
EDITOR=vim
SIM_HOME=/cvmfs/oasis.opensciencegrid.org/jlab/geant4
PWD=/scratch/brads/test
```

```
Number of samples:                    100000000 (100 M)
Number inside unit quarter-circle:     78534714
Pi is roughly:                        3.141389

-- Copy the file to /work/hallc/c-polhe3/brads/workshop-temp --------------------------
Nothing to do here.  Let's handle this using swif2 '-output ....' option

-- List Working Directory (END OF JOB) --------------------------------------------
total 28
drwx------. 3 brads a-phy 4096 Sep  3 22:36 ./
drwx------. 5 brads a-phy   83 Sep  3 22:37 ../
drwx------. 7 brads a-phy  155 Sep  2 20:11 .git/
-rw-------. 1 brads a-phy  160 Sep  2 19:59 .gitignore
-rwxr-xr-x. 1 brads a-phy 4435 Sep  3 20:13 job-script.sh*
lrwxrwxrwx. 1 brads a-phy   40 Sep  3 19:08 output-work -> /work/hallc/c-polhe3/brads/workshop-temp/
-rw-r--r--. 1 brads a-phy 1409 Sep  3 22:38 pi-calc-output-100000000_test-2913113.out
-rwxr-xr-x. 1 brads a-phy 2186 Sep  3 13:48 piece-of.py*
-rwxr-xr-x. 1 brads a-phy 2224 Sep  3 20:54 swif-submission.sh*
```

Dev Process: Test your ‘swif submission’ script

- Develop and test a script that can generate the ‘swif2 add-job …’ command for each job in your workflow
  - » swif-submission.sh
  - » Update **SLURM_ACCT**
  - → This can/will loop over runs / run numbers (from directory or file), etc and create a swif job submission for each case.
    - – Declare (tape) -outputs
    - – Declare (tape) -inputs
  - » **NOTE**: mss:// (tape) files will be copied to working dir!
    - – Access them as *./<filename>* in your analysis script *not* */cache/…./<filename>*
  - → Also sets ‘defaults’ like:
  - » slurm partition/queue
  - » slurm ‘account’
  - » RAM, disk space, CPU needs

```
#!/bin/bash

# - Use *your* slurm account
#   Find your name here: https://scicomp.jlab.org/scicomp/slurmJob/slurmAccount
SLURM_ACCT='hallc'  # USE YOUR SLURM ACCOUNT/group here!
WORKFLOW="SWS-pitest-$USER"

SRCDIR="$HOME/workshop-sept2024"        # This is where your job code/exe will be
DESTDIR="/farm_out/$USER/workshop-temp" # This is where your output will be copied for this test
OUTPUT_GLOB='match:pi-calc-output-*'    # output file(s) copied to $DESTDIR by swif

JOB_SCRIPT="${SRCDIR}/job-script.sh"
JOB_ARGS="5000000 50000000 5000000000" # 5M, 50M, 5000M

ECHO="echo"   ## Used to echo swif commands for testing
if [ ${1:-unset} == "submit" ]; then
  ECHO=""  ## disables the 'echo' and actually runs the swif commands
fi

# NOTES:
# - The -input file is a semi-random file just used as an example of how to
#   pull from tape and demonstrate the file lands in the working directory on
#   the farm node.  For example:
#      -input 'shms_all_09450.dat' 'mss:/mss/hallc/c-polhe3/raw/shms_all_09450.dat' \
#
# - For 'real' jobs, you would use the 'production' partition/queue:
#      -partition 'production'
#   The 'priority' partitin/queue has limited run time and resource restrictions
#   but will schedule rapidly -- intended for quickly testing a short job
#   or two before switching to 'production' and submitting a complete workflow

for N in $JOB_ARGS; do
  $ECHO \
  swif2 add-job \
    -create \
    -workflow "$WORKFLOW" \
    -account "$SLURM_ACCT" \
    -partition 'priority' \
    -constraint 'el9'  \
    -cores '1'  \
    -ram '100M'  \
    -time 5min  \
    -shell '/bin/bash'  \
    -output "$OUTPUT_GLOB" "$DESTDIR/" \
    "$JOB_SCRIPT" "$N"
done
```

Dev Process: Test your ‘swif add-job …’ script

- Comments / Notes

  -cores <N>

  → know what your job can actually use

  -ram <size>

  → shoot for < 2GB/core if you can

  -disk <size>

  → disk space needed on the farm node itself

  → If set, then <size> == size for both

  » ‘-input’ files, *AND*

  » ‘-output’ files on the local disk

  → If unset, then Swif will use

  » SizeOf(input files) * 1.1

  -scratch_disk <size> Coming Soon!

  → <size> only has to account for output files on local disk (space for input files is added to this automatically)

- “Gotchas” to remember:

  → Do *not* modify your executable or job-script.sh until your workflow is fully completed

  » no recompiling; no ‘tweaks’

  » (It will change what runs on the farm mid-workflow!)

  → Remember many jobs will run at the exact same time!

  » If you run your job from within a network mounted space (ie. /group, /work) then you *must* ensure input and output files have unique names, ‘temp’ files don’t step on each other, etc.

  → Access ‘-input mss://….’ files from job’s working directory (*not from /cache*)

Submit your Jobs to SWIF

- Check lines from
  → swif-submission.sh
  ~ Then ~
- Run ‘swif-submission.sh submit’

```
brads@ifarm2401 1084% ./swif-submission.sh submit
id                  = 32969382
name                = SWS-pietest-brads-0

id                  = 32969383
name                = SWS-pietest-brads-1

id                  = 32969384
name                = SWS-pietest-brads-2

-- Active SWIF workflows --------
swif2 list

-- SWIF jobs for workflow SWS-pietest-brads --------
swif2 status -workflow SWS-pietest-brads -jobs

NOTE: IF the workflow looks good, then tell SWIF to execute it with:
  swif2 run -workflow 'SWS-pietest-brads'
```

```
brads@ifarm2401 1020% cd ~/workshop-sept2024/
brads@ifarm2401 1021% ./swif-submission.sh
swif2 add-job -create -workflow SWS-pitest-brads -account h
allc -partition priority -constraint el9 -cores 1 -ram 100M
 -time 5min -shell /bin/bash -output match:pi-calc-output-*
 /work/hallc/c-polhe3/brads/workshop-temp/ /home/brads/work
shop-sept2024/job-script.sh 5000000
swif2 add-job -create -workflow SWS-pitest-brads -account h
allc -partition priority -constraint el9 -cores 1 -ram 100M
 -time 5min -shell /bin/bash -output match:pi-calc-output-*
 /work/hallc/c-polhe3/brads/workshop-temp/ /home/brads/work
shop-sept2024/job-script.sh 50000000
swif2 add-job -create -workflow SWS-pitest-brads -account h
allc -partition priority -constraint el9 -cores 1 -ram 100M
 -time 5min -shell /bin/bash -output match:pi-calc-output-*
 /work/hallc/c-polhe3/brads/workshop-temp/ /home/brads/work
shop-sept2024/job-script.sh 5000000000
brads@ifarm2401 1022%
```

Don’t forget this step!!

Useful SWIF2 Commands

- swif2 list
  → see all of your workflows
- swif2 status -workflow $WORKFLOW
  → see summary details for $WORKFLOW
- swif2 status -workflow "$WORKFLOW" -jobs | less
  → see a list of jobs associated with $WORKFLOW (can be big list!)
  → find SWIF job_ids, (*etc*)
- watch -d -n30 -- “swif2 status $WORKFLOW”
  → if you’re impatient like me…
- swif2 show-job -jid <swif_job_id>
  → get info on a particular SWIF2 job
    » job_status, slurm_id, etc

Useful Slurm Commands

NOTE: Slurm commands take slurm job-ids, not SWIF job-ids!

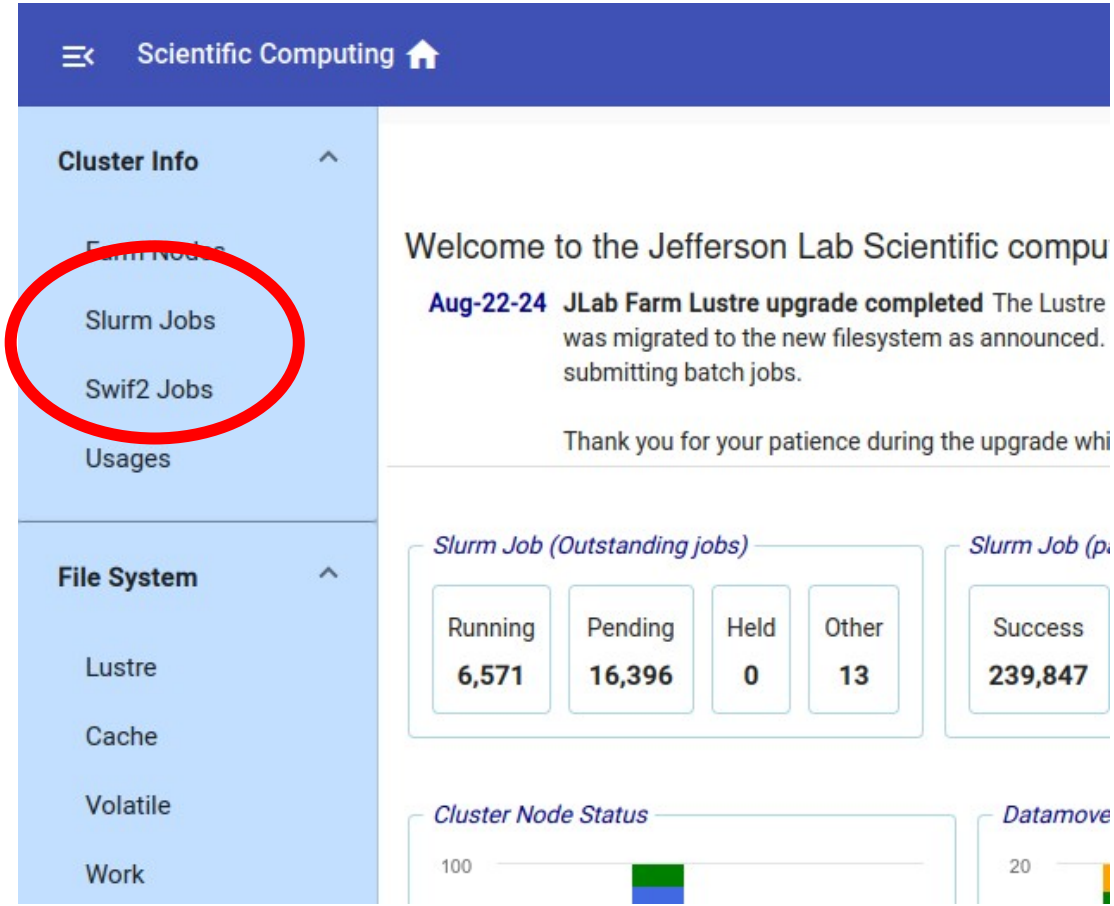
- sacct
  - → queries *all the things* (ie. slurm DB access)
  - → https://slurm.schedmd.com/sacct.html
- sacct --long -j <slurm_id>
- sacct -- env-vars -j <slurm_id>
  - → env vars set during a slurm job
  - → SWIF_JOB_ID, SWIF_JOB_ATTEMPT_ID ← map back to SWIF job!
- seff <slurm_id>
  - → wrapper around sacct that reports job efficiency numbers
  - → RAM, CPU usage

Don’t forget about /farm_out/$USER/swif/...

- cd /farm_out/$USER/swif

- ls
  → Should see a list of stdout and stderr for jobs that have run (or are currently running!)
- less <job-name-id>.out
  → shows everything you dumped in your job-script.sh and what your executable wrote to stdout
- less <job-name-id>.err
  → shows text written to stderr (usually ‘errors’) and/or debug info

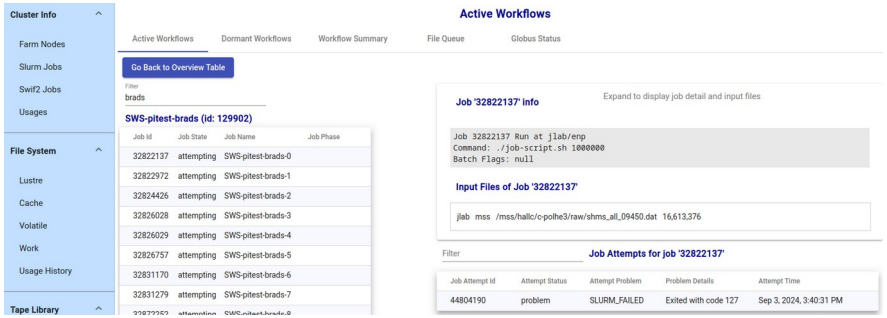
Web-based SWIF/Slurm Reporting

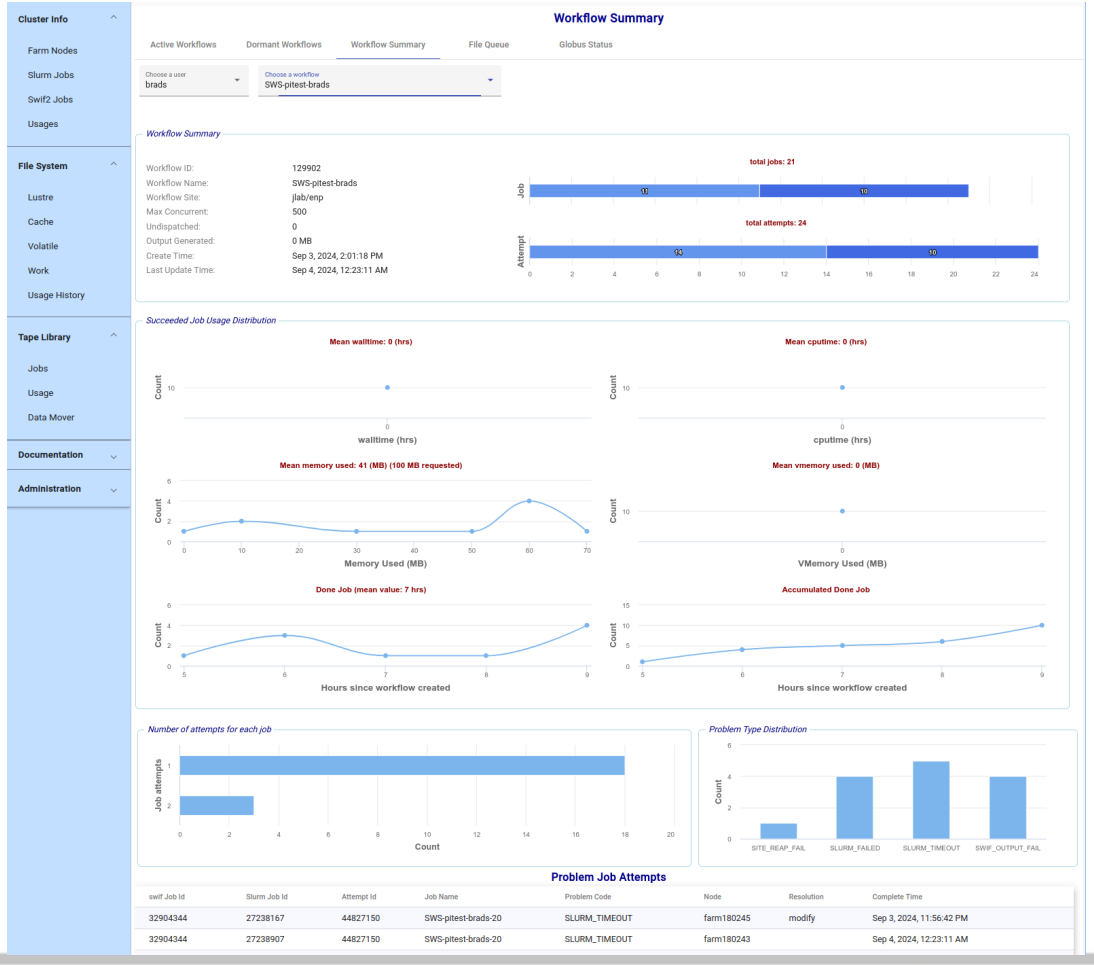


- Most of the command-line info from SWIF and slurm can be found on scicomp.jlab.org web page
  - → See Swif2 Jobs
  - → See Slurm Jobs

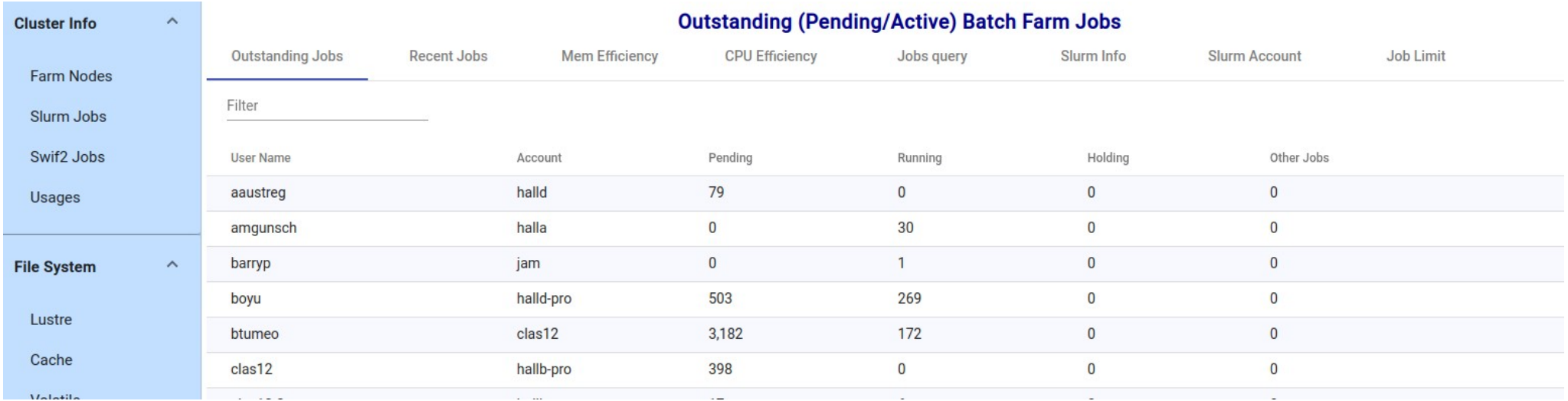
Web-based SWIF2 Reporting

- SWIF2 Jobs
  - → Workflow Summary
    - » aggregate job info
    - » failed jobs at bottom
  - → Active (Dormant) Workflows
    - » What is running/waiting
    - » Filter to find your workflow and select it for job list
      - – click on job for job info





Web-based Slurm Reporting



- Inefficient Job information
  - → Mem / CPU efficiency tabs
  - → Will get an email if you hit this list
- Jobs Query
  - → look up slurm job information

- "Slurm info" tab
  - → Find 'feature' tags for use with job 'constraints'
  - → See a list of partitions (job queues)
  - → Slurm Account
    - » Find your name; use the correct slurm account for your job!

Fix and Resubmit failed job(s) with Swif2

- If we get this far and you are monitoring your jobs, you will find that 1 of the 3 jobs we submitted failed:
  - → SLURM_TIMEOUT error
  - → The 500M event job took longer than the 5min we declared in swif
- Fix this and resubmit:
  - → swif2 modify-jobs $WORKFLOW -time 20min -problems SLURM_TIMEOUT

```
brads@ifarm2401 1117% swif2 modify-jobs $WORKFLOW -time set 20min -problems SLURM_TIMEOUT
Found 1 matching jobs
Modified 1 job
Resolving 1 problem job
brads@ifarm2401 1118%
```

Any Wisdom/Hints for Others?

- What issues have you run into?
  - → Raise them here so others can learn from your experience!
- Any quirks that need better documentation?
- Let me know so we have a shot at fixing the problems/issues?

- 'swif2 notify' (as is) not super useful
  - → only reports if all jobs in a workflow are tagged 'DONE'
  - → a single pending/broken will keep this from firing
- 'swif2 notify' is being tweaked to allow for email notification if there are no more jobs capable of running
  - → could be successful or stalled
  - → *much more useful*
    - » *Coming Soon!*