Running persistent jobs in Condor

Derek Weitzel & Brian Bockelman

Holland Computing Center

University of Nebraska Lincoln

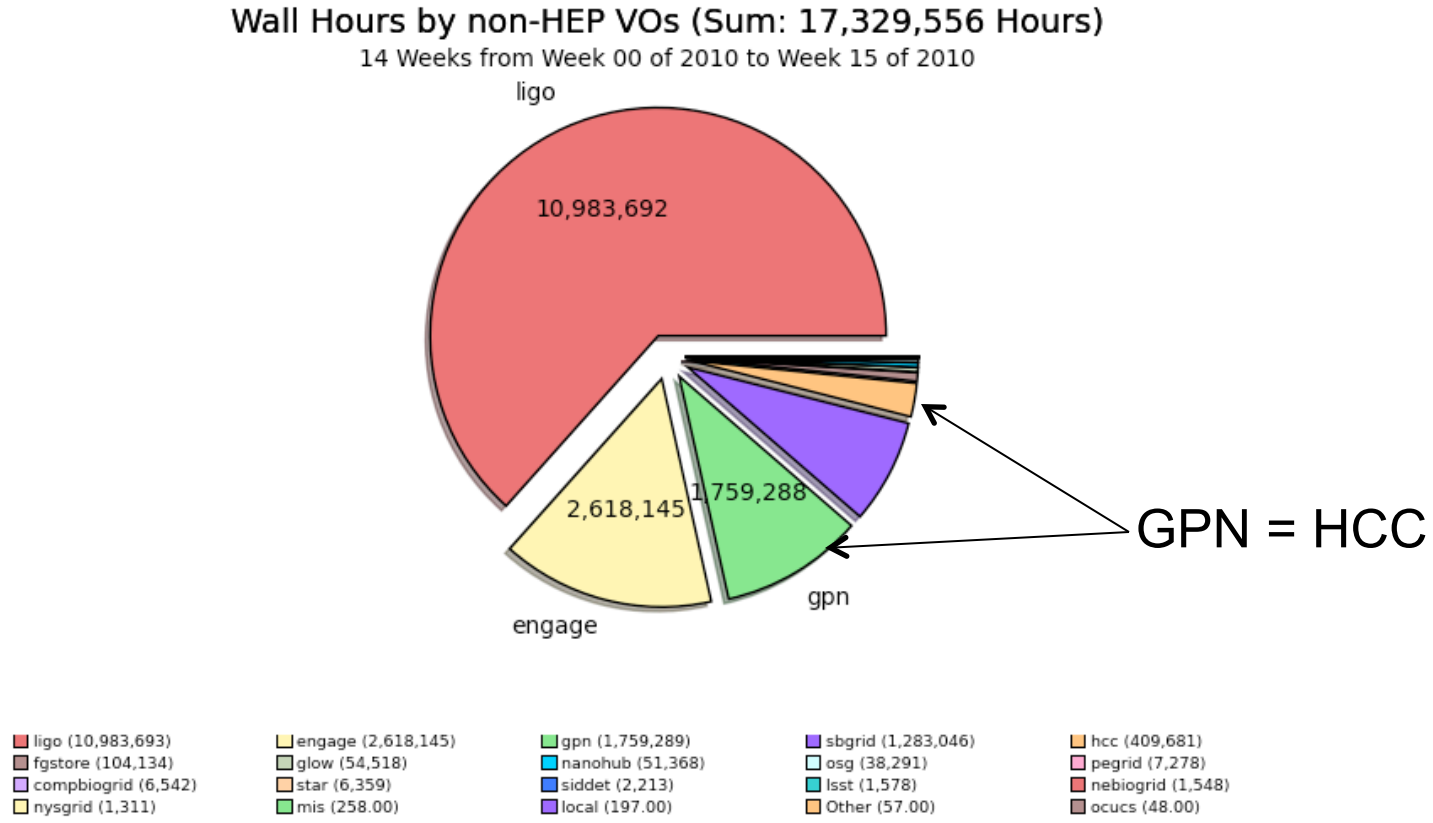
Introduction – HCC

- University of Nebraska
  Holland Computing Center – 2009
- 162 Research groups
  - Not all of them are using Condor… wide variety of users.
- Condor Applications:
  - Bioinformatics: CS-Rosetta, Autodock…
  - Math: Graph Transversal
  - Physics: CMS and Dzero

Introduction – HCC

- 7,000 Cores, all opportunistic for grid
- HCC's Use of the Open Science Grid
  - 2.1 Million hours this year
- Grid usage is based off GlideinWMS & Engage's OSG-MM

Open Science Non-HEP Grid Usage



Persistent Jobs

- Today, I'll be discussing a unique way we create backfill at HCC using what we call "persistent jobs"
- We are sharing it because it's an unexpected combination of several unique-to-Condor capabilities to solve a problem that usually is solved by writing a lot of other code.

Persistent Jobs - Motivation

- What is a persistent job?
  - A single task which lasts longer than a single batch job can run.
  - It is thus necessary to keep state between runs
- If your cluster limits batch jobs to 12 hours, a 24 hour task would be a persistent job.
- At a minimum, between runs, the persistent job must keep some state – what job am I?
  - This could also be a quite large checkpoint!

Persistent Jobs - Examples

- A few examples:
  - Traditional Condor standard universe job that uses checkpoints.
  - Processing streams of data, where there is one job per stream.
- Taken to the extreme, a persistent job *might never exit*.
- We will show an example of persistent jobs for grid backfill.

Persistent Jobs - Difficulties

- Persistent jobs are difficult because:
  - We want no more than one instance running,
  - Prefer more than zero instances running.
- To satisfy the first, we need some sort of locking and heartbeat mechanisms.
- To satisfy the second, we need something to automatically re-queue.
  - And then something to monitor the automatic submission framework.
- Oh, and deadlock detection!

“Traditional Approach”

- Keep the state in a database.
  - Have a locking check-in/check-out mechanisms to make sure checkpoints are used by one job.
- Have a server listening for job heartbeats and update the DB.
- Have a submit daemon.
- A lot of infrastructure! At least three services you have to write, care, and feed.

Yuck!

- We just invented a ton of architecture.
  - Too much for a grad student to write, much less operate.
- Distributed architectures are ripe for bugs.
  - You can take a semester long course in this and still not be very good at it.
- Having a cron/daemon that submits jobs is asking for a bug that submit an infinite # of jobs or deadlocks and submits 0.

Persistent Jobs – the Condor Way!

- Can we come up with a solution that:
  - Uses Condor only
  - Runs on grid (Grid Universe or Glide-in).
  - Allows us to checkpoint.
- Insight: ***Use the Condor Global Job ID as the unique state!***
  - And Condor can re-run completed jobs!
  - Don't do book-keeping, let Condor do it for you.

Persistent Jobs – the Condor Way!

- The state kept by each job is the Condor Job ID.
- Use the fact Condor can resubmit completed jobs.
  - Set “OnExitRemove = FALSE”
  - For Grid:
    - Globus_Resubmit = TRUE
    - Globus_Rematch = TRUE
- Use the job ID to uniquely identify a checkpoint file kept in a storage element.
  - Periodically overwrite with new checkpoints.
- Sit back, and use the fact that Condor will keep trying to re-run the job indefinitely and will make sure that the job only has one running instance.

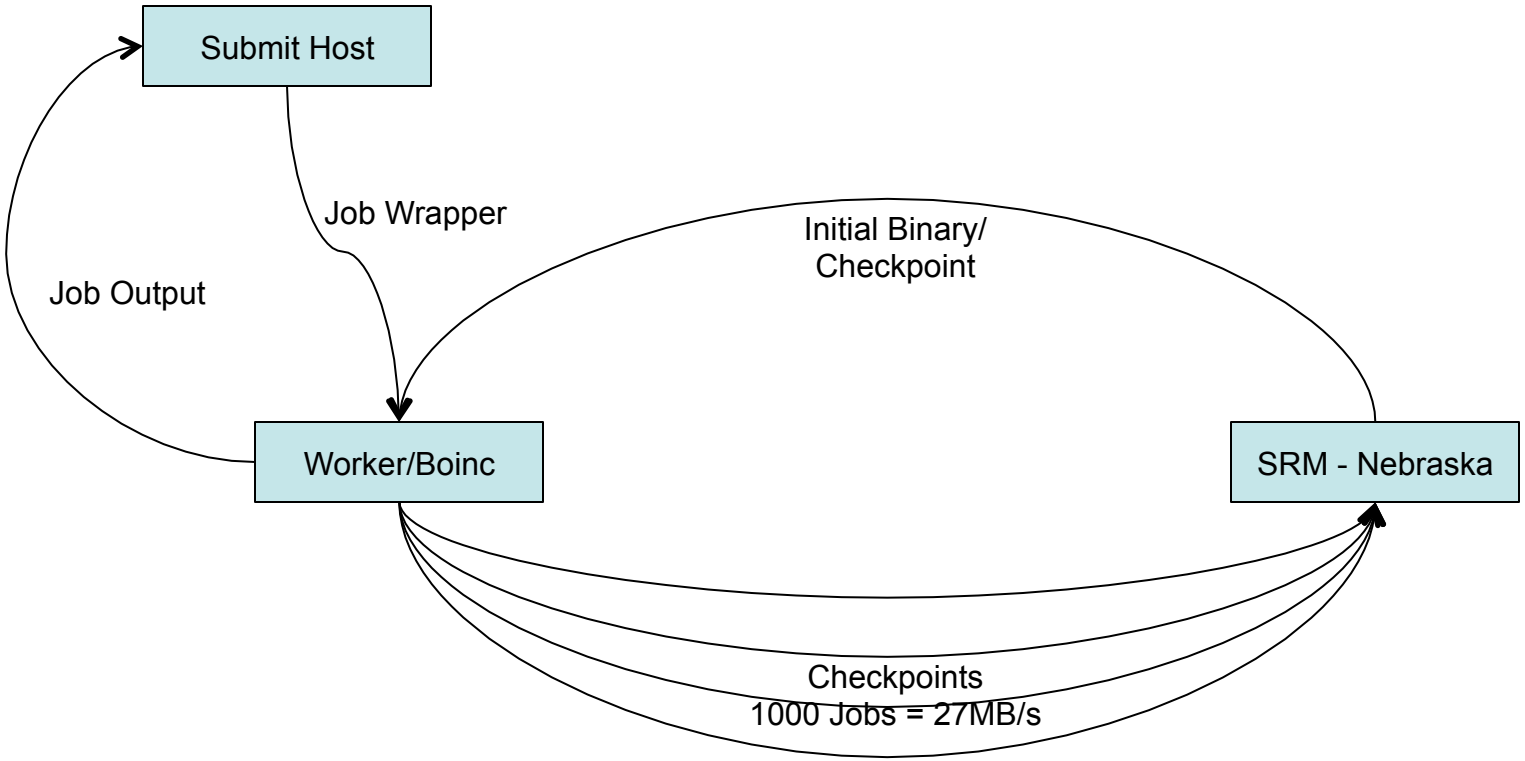
Our Use Case

- We want to have an infinite amount of work for our clusters.
  - Yet people will be mad if we just do sleep()
- Solution: BOINC
  - Einstein@Home is an acceptable use of our resources.
  - BOINC is a persistent job! As long as you feed it the same checkpoint directory(*), it will continue to run indefinitely!
    - Actually, it gets fed jobs internally, but we don't know anything about the job boundaries.

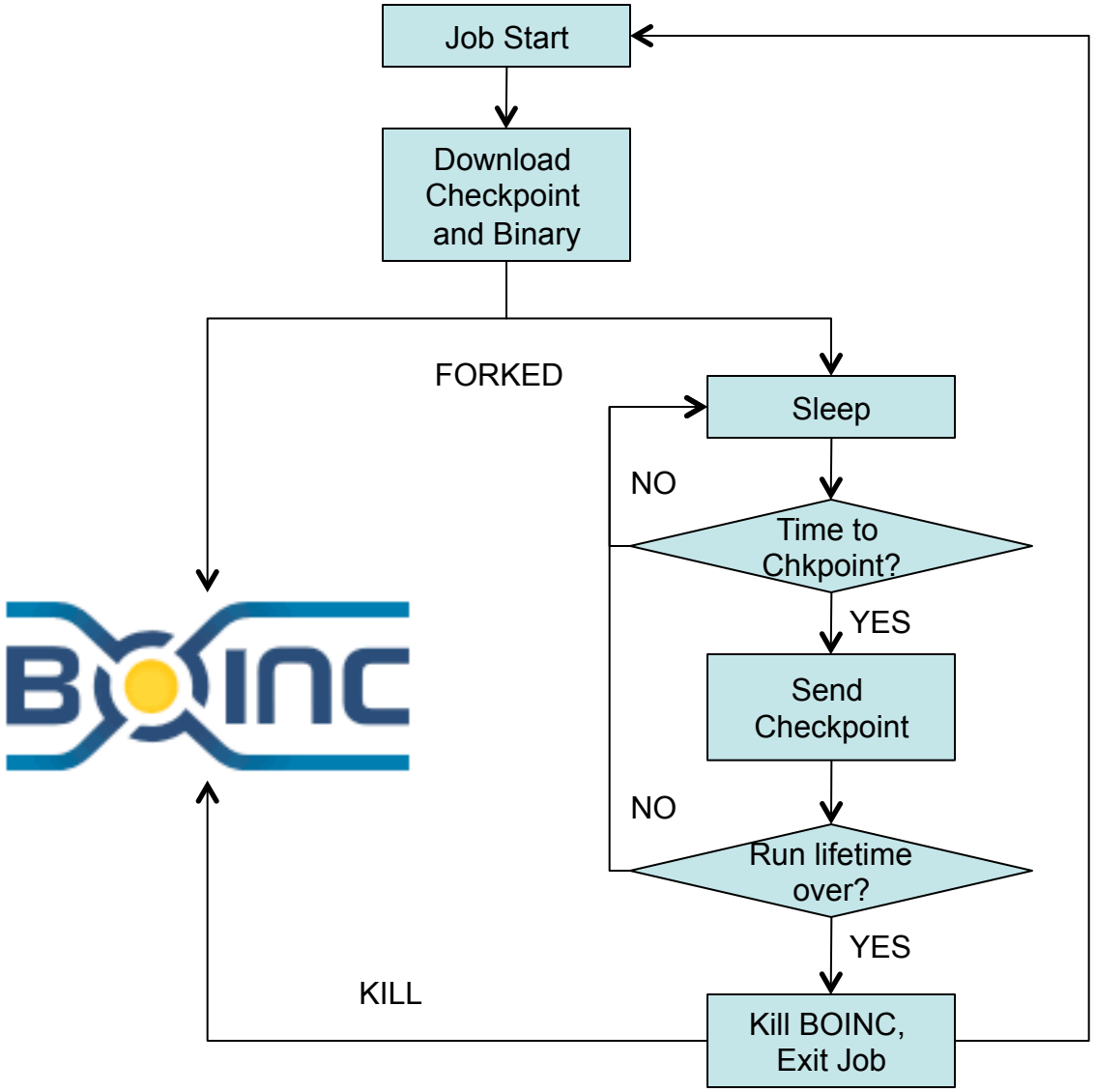
Our Solution

- We take the basic BOINC binary, statically compiled for RHEL4/32-bit for maximum compatibility.
- BOINC uses the md5sum of the host's mac address to identify the checkpoint directory is correct between executions.
  - Replace use of mac address with the md5sum of the Condor JobID!
  - Now, regardless of the worker node running the job, the correct checkpoint is used.
- Checkpoints (100MB each) kept on SRM storage. No NFS used!

Data Flow

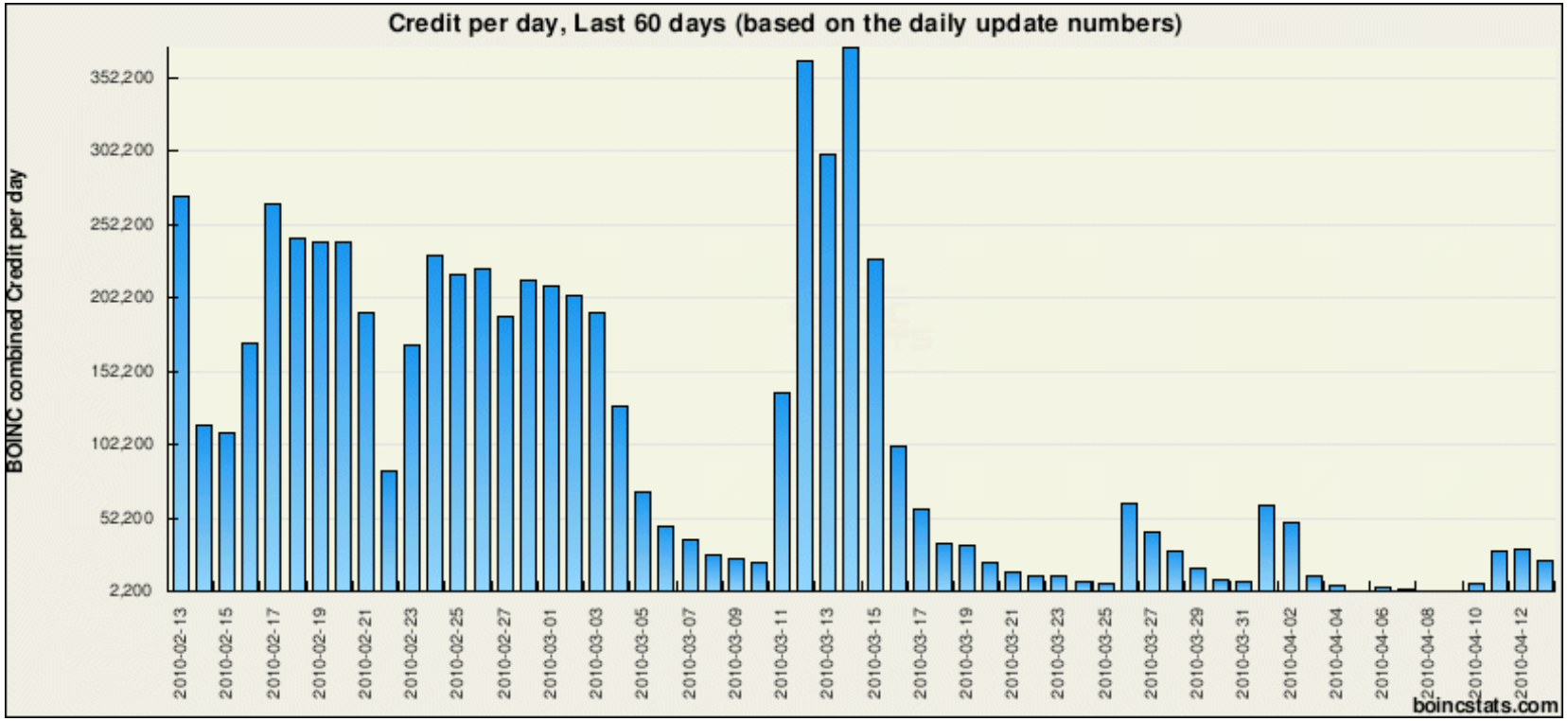

Submit Host
Job Wrapper
Initial Binary/
Checkpoint
Job Output
Worker/Boinc
SRM - Nebraska
Checkpoints
1000 Jobs = 27MB/s

Workflow



Example Values:
Checkpoint = 1hr
Lifetime = 12hrs

Boinc Results



Why not Condor backfill?

- Why this and not use Condor backfill?
  - Not all of our sites run Condor. We want to be able to utilize an arbitrary site.
    - Lots of exercise for operating grid components!
  - Standard backfill tied to physical worker node (MAC). No clue when you will see that node again on the grid, so you might just be throwing away work. BOINC progress expires in 3 days.
  - Can't push out backfill jobs using OSG-MM.
    - Possibly could do this for gWMS; I don't know.

GlideinWMS Backfill

- Keeping a Glide-in instance busy with backfill until you get user jobs.
  - Preempt backfill jobs for user jobs.
  - Removes the user ‘ramp up’ time.
- Use Flocking to GlideinWMS
  - GlideinWMS is resource hungry, need interactive nodes for deployment.
  - Run Backfill to have running glideins for the flocked jobs.

Future Uses

- Where else could persistent jobs be useful?
  - Glidein Pilot Factory. Removes the need to have processes that submit jobs.
    - If the thing stops working, then it's a Condor bug, and you have "someone else" to yell at. ☺

Conclusions

- “Persistent jobs” can be a nightmare due to the necessary bookkeeping.
- Condor jobs *can be made persistent*.
- Don’t go inventing things Condor does for you.
- By utilizing basic Condor *and nothing else* (entire project is 2 python scripts), we can generate an arbitrary amount of backfill for our site or the OSG!

Questions?

Code location:
svn://t2.unl.edu/brian/osg_boinc

University of Nebraska Lincoln

Backup Slides

Issues found during implementation

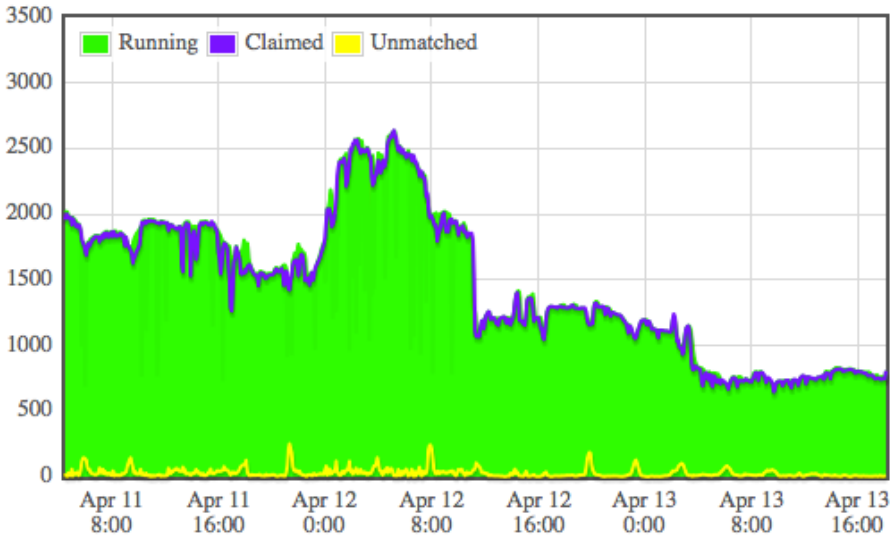
- Rematching in grid universe
  - Using matchmaking in grid universe
  - Need special keyword:
    - GlobusRematch = TRUE
  - Was a huge pain to get right.
- Job accounting
  - Job accounting collects data when Condor job finishes, not for each run.
  - Not solved, relying on grid resource accounting

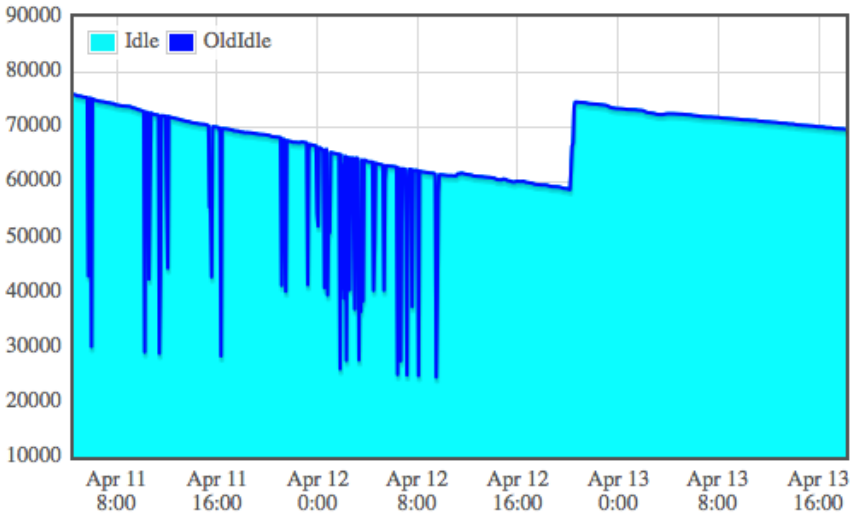
Issues found during implementation

- Need to easily modify existing jobs
  - In ‘Arguments’, use special variables:
    - $$([Variable])
  - Variables evaluated at time of match
  - Used for job lifetime, checkpoint location, binary location

Results

- Glideins running / Jobs Queued





Glidein Usage by Site