A Year of HTCondor Monitoring

Lincoln Bryant
Suchandra Thapa

HTCondor Week 2015
May 21, 2015

Computation Institute

Analytics vs. Operations

- Two parallel tracks in mind:
  - Operations
  - Analytics
- Operations needs to:
  - Observe trends.
  - Be alerted and respond to problems.
- Analytics needs to:
  - Store and retrieve the full ClassAds of every job.
  - Perform deep queries over the entire history of jobs.

Motivations

- For OSG Connect, we want to:
  - track HTCondor status
  - discover job profiles of various users to determine what resources users are need
  - determine when user jobs are failing and help users to correct failures
  - create dashboards to provide high-level overviews
  - open our monitoring up to end-users
  - be alerted when there are problems
- Existing tools like Cycleserver, Cacti, Ganglia, etc.. did not adequately cover our use case

Operations

Graphite

- Real-time graphing for time series data
- Open source, developed by Orbitz
- Three main components:
  - Whisper - Data format, replacement for RRD
  - Carbon - Listener daemon
  - Front-end - Web interface for metrics
- Dead simple protocol
  - Open socket, fire off metrics in the form of:

```
path.to.metric <value> <timestamp>
```

Graphite home page: http://graphite.wikidot.com/

Sending HTCondor stats

- To try it out, you can just parse the output of “condor_q” into the desired format
- Then, simply use netcat to send it to the Graphite server

```
#!/bin/bash

metric=”htcondor.running”
value=$(condor_q | grep R | wc -l)
timestamp=$(date +%s)

echo ”$metric $value $timestamp” | nc \
 graphite.yourdomain.edu 2003
```

A simple first metric

- Run the script with cron:

Problems with parsing

- Parsing condor_q output is a heavyweight and potentially fragile operation
  - Especially if you do it once a minute
  - Be prepared for gaps in the data if your schedd is super busy
- What to do then?
  - Ask the daemons directly with the Python bindings!

Collecting Collector stats via Python

- Here we ask the collector for slots in “claimed” state and sum them up:

```
import classad, htcondor


coll = htcondor.Collector("htcondor.domain.edu")
slotState = coll.query(htcondor.AdTypes.Startd,
"true",['Name','JobId','State','RemoteOwner','COLLECTOR_HOST_STRI
NG'])


for slot in slotState[:]:
        if (slot['State'] == "Claimed"):
                slot_claimed += 1


print "condor.claimed "+ str(slot_claimed) + " " + str(timestamp)
```

Sample HTCondor Collector summary

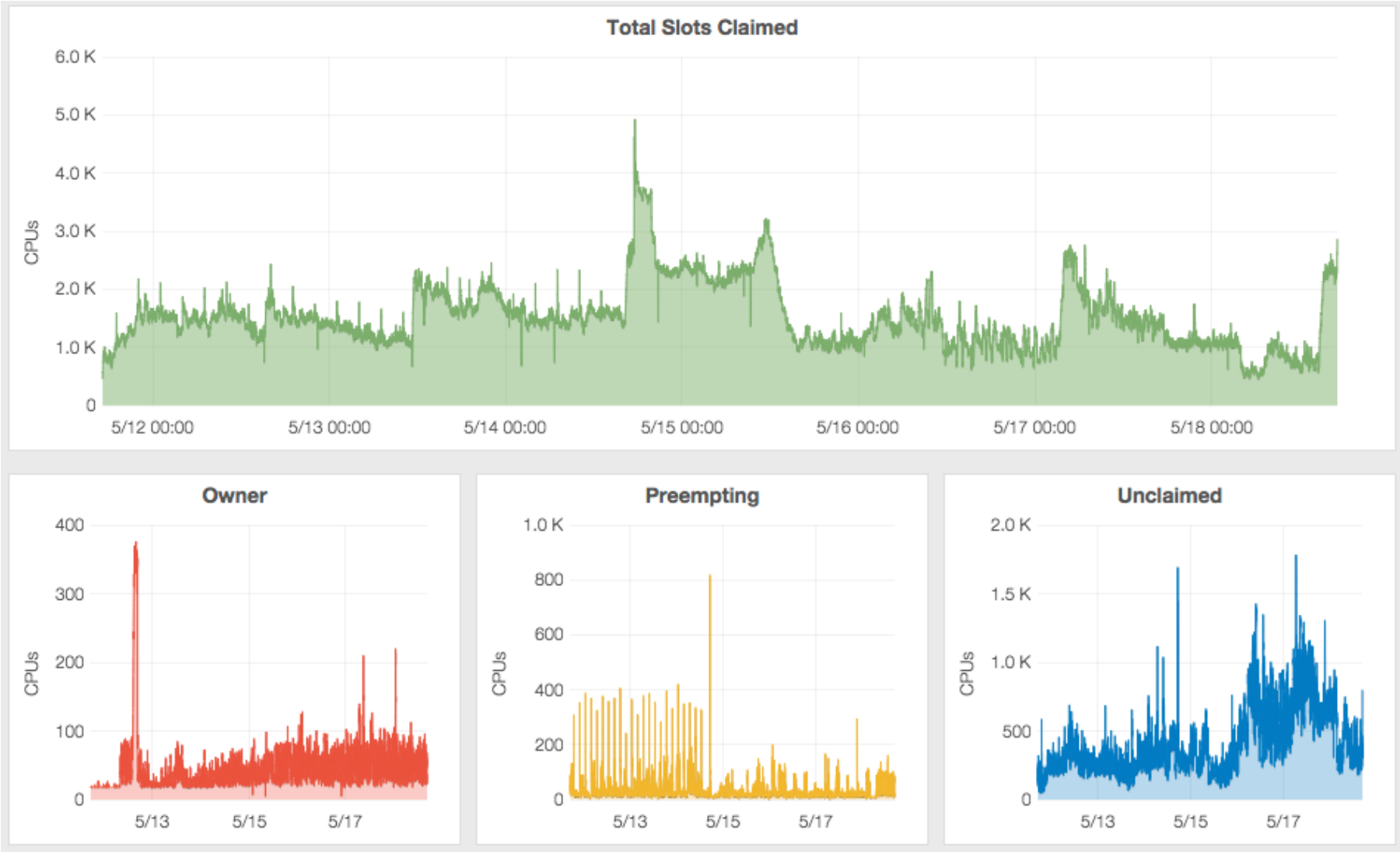
- Fire off a cron & come back to a nice plot:

Grafana

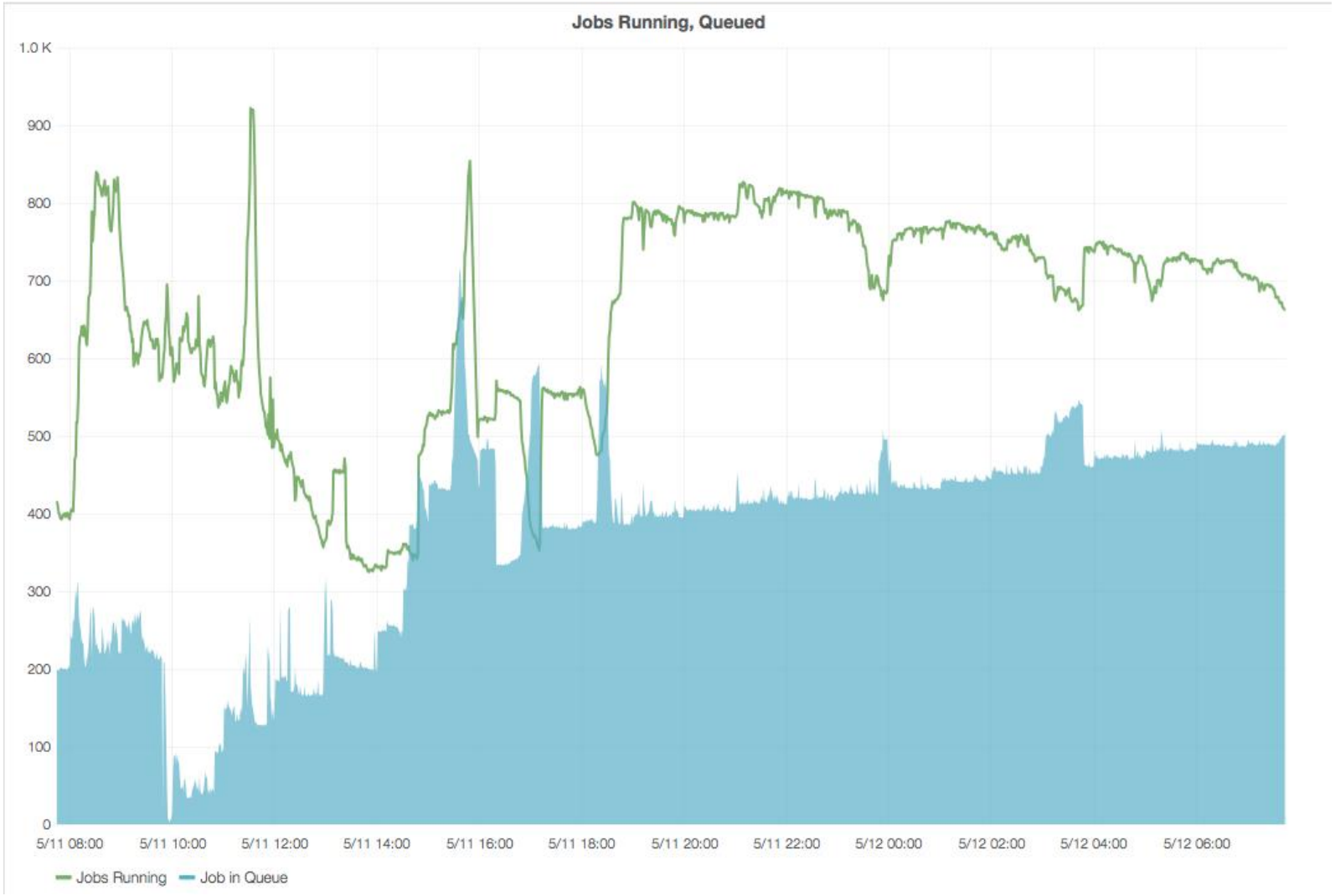
- Graphite is nice, but static PNGs are so Web 1.0
- Fortunately, Graphite can export raw JSON instead
- Grafana is an overlay for Graphite that renders the JSON data using flot
  - Nice HTML / Javascript-based graphs
  - Admins can quickly assemble dashboards, carousels, etc
  - Saved dashboards are backed by nosql database
- All stored Graphite metrics should work out of the box

Grafana home page: http://grafana.org/
flot home page: http://www.flotcharts.org/

Sample dashboard



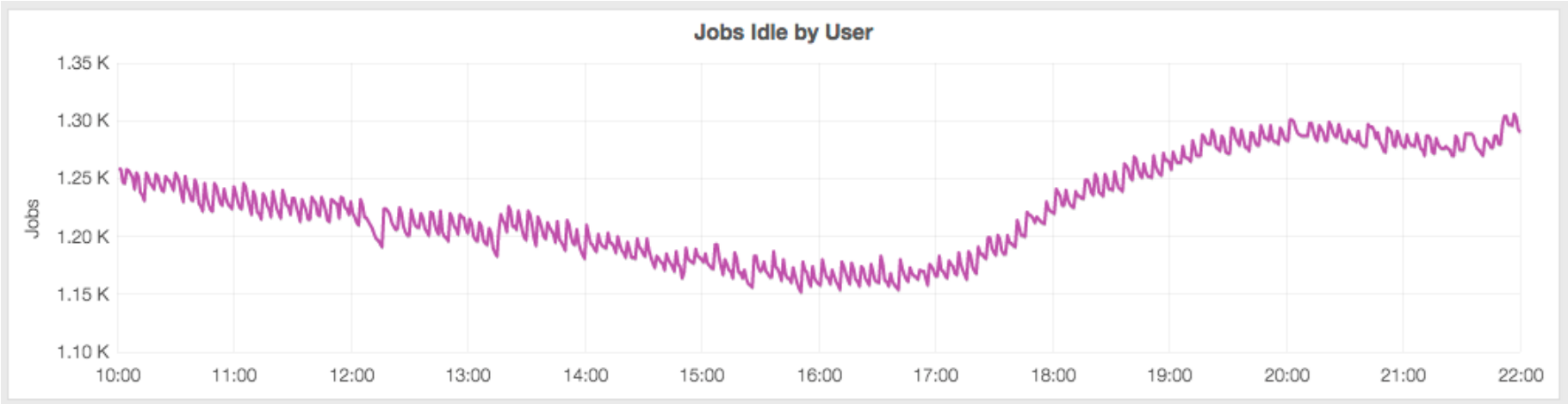
Sample dashboard



An example trend

- A user's jobs were rapidly flipping between idle and running. Why?
- Turns out to be a problematic job with an aggressive periodic release:

```
periodic_release = ((CurrentTime - EnteredCurrentStatus) > 60)
```



(Credit to Mats Rynge for pointing this out)

Active monitoring with Jenkins

- Nominally a continuous integration tool for building software
- Easily configured to submit simple Condor jobs instead
- Behaves similar to a real user
  - Grabs latest functional tests via git
  - Runs “condor_submit” for a simple job
  - Checks for correct output, notifies upon failure
- Plethora of integrations for notifying sysops of problems:
  - Email, IRC, XMPP, SMS, Slack, etc.

Jenkins monitoring samples

- Dashboard gives a reassuring all-clear:

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Submission to OrangeGrid	50 min - #399	6 days 8 hr - #247	18 min
		Submission to OSG	13 min - #1559	2 days 22 hr - #1277	55 sec
		Submission to UC3	6 min 22 sec - #405	N/A	1 min 21 sec

- Slack integration logs & notifies support team of problems:

jenkins BOT 12:57 AM
Submission to OSG - #1219 Failure after 3 min 33 sec (Open)

jenkins BOT 1:09 AM
Submission to OSG - #1220 Back to normal after 50 sec (Open)

Operations - Lessons learned

- Using the HTCondor Python bindings for monitoring is just as easy, if not easier, than scraping condor_{q,status}
- If you plan to have a lot of metrics, the sooner you move to SSD(s), the better
- Weird oscillatory patterns, sudden drops in running jobs, huge spikes in idle jobs can all be indicative of problems
- Continuous active testing + alerting infrastructure is key for catching problems before end-users do

Analytics

In the beginning...

- We started with a summer project where students would be visualizing HTCondor job data
- To speed up startup, we wrote a small python script (~50 lines) that queried HTCondor for history information and added any new records to a MongoDB server
  - Intended so that students would have an existing data source
  - Ended up being used for much longer

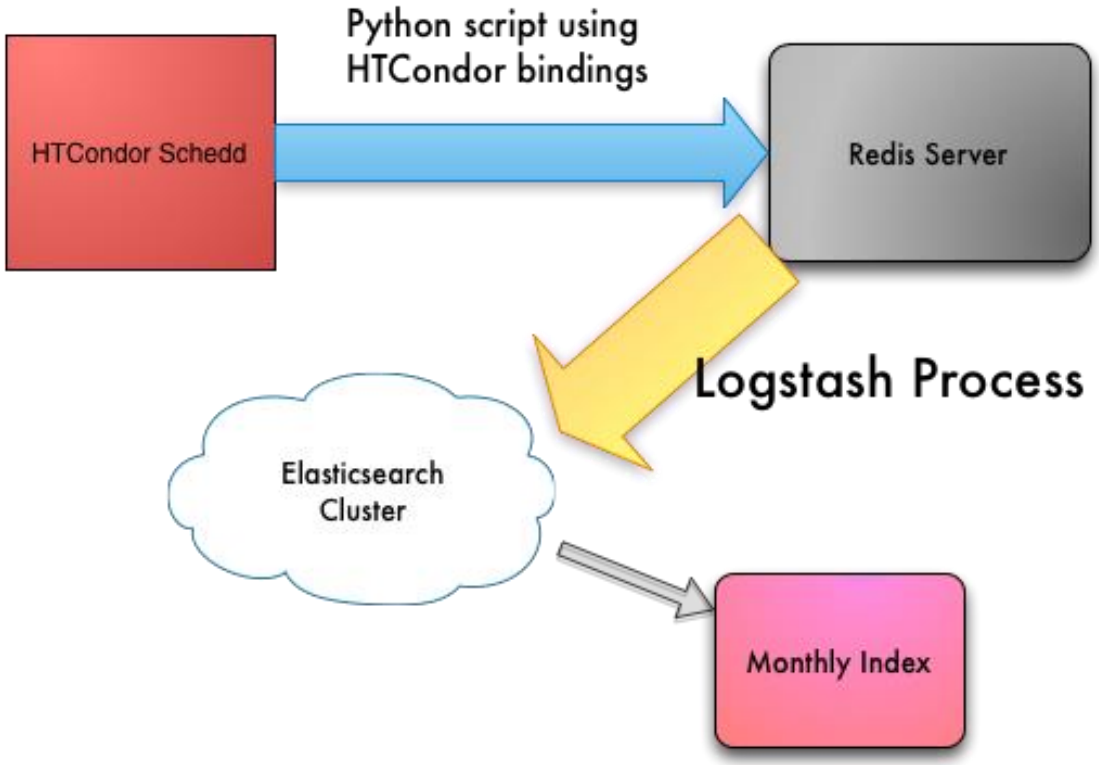
Initial data visualization efforts

- Had a few students working on visualizing data over last summer
  - Generated a few visualizations using MongoDB and other sources
  - Tried importing data from MongoDB to Elasticsearch using Kibana
  - Used a few visualizations for a while but eventually stopped due to maintenance required
- Created a homebrew system using MongoDB, python, highcharts and cherrypy

Current setup

- Probes to collect condor history from log file and to check the schedd every minute
- Redis server for pub/sub channels for probes
- Logstash to follow Redis channels and to insert data into Elasticsearch
- Elasticsearch cluster for storage and queries
- Kibana for user and operational dashboards, RStudio/python scripts for more complicated analytics

Indexing job history information

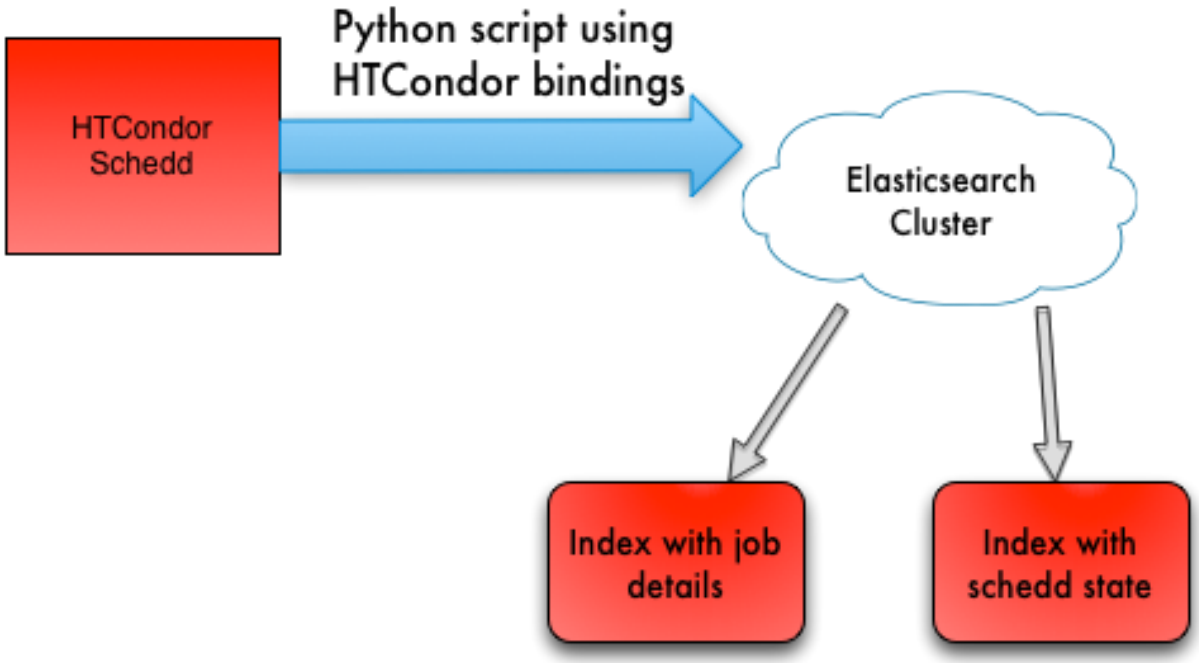


Python script polls the history logs periodically for new entries and publishes this to a Redis channel.

Classads get published to a channel on the Redis server and read by Logstash

Due to size of classads on Elasticsearch and because ES only works on data in memory, data goes into a new index each month

Indexing schedd data



Python script is run every minute by a cronjob and collects classads for all jobs. The complete set of job classads is put into an ES index for that week. Script also inserts a record with number of jobs in each state into another ES index.

Querying/Visualizing information

- All of this information is good, but need a way of querying and visualizing it
- Luckily, Elasticsearch integrates with Kibana which provides a web interface for querying/visualization

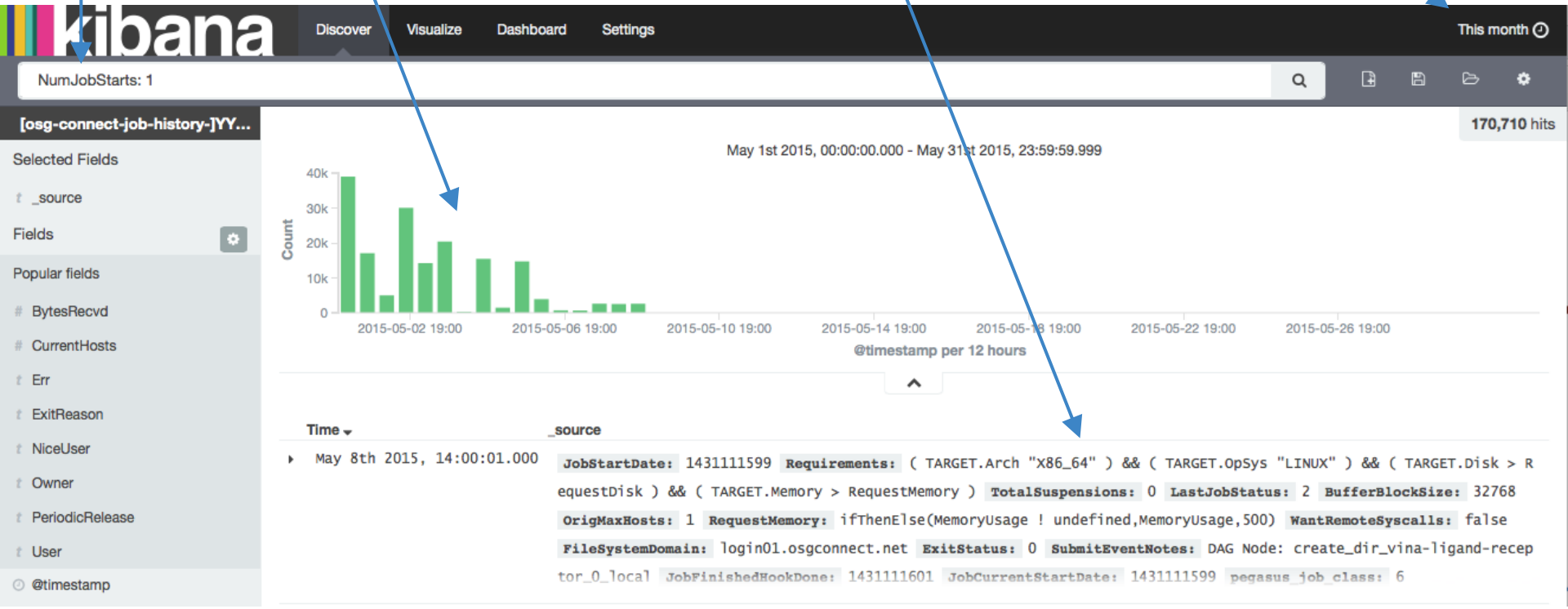
Kibana Overview

Query field

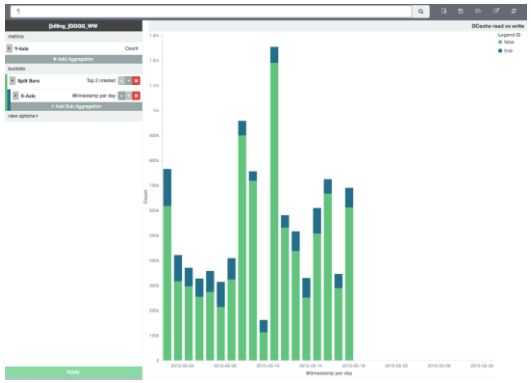
Hits over time

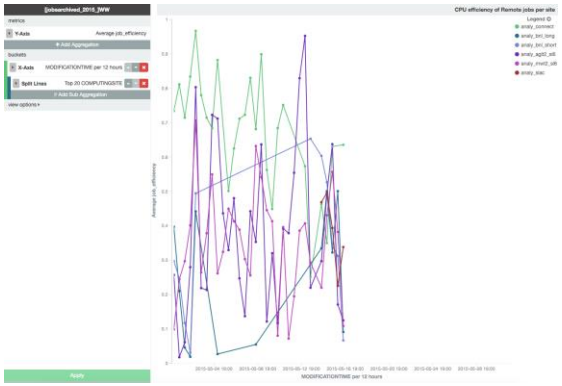
Sampling of documents found

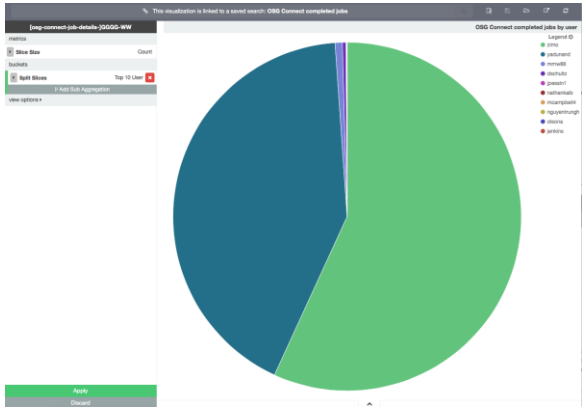
Time range selector

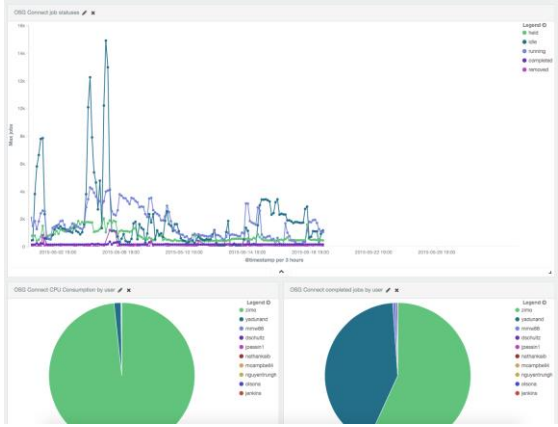


Plotting/Dashboards with Kibana









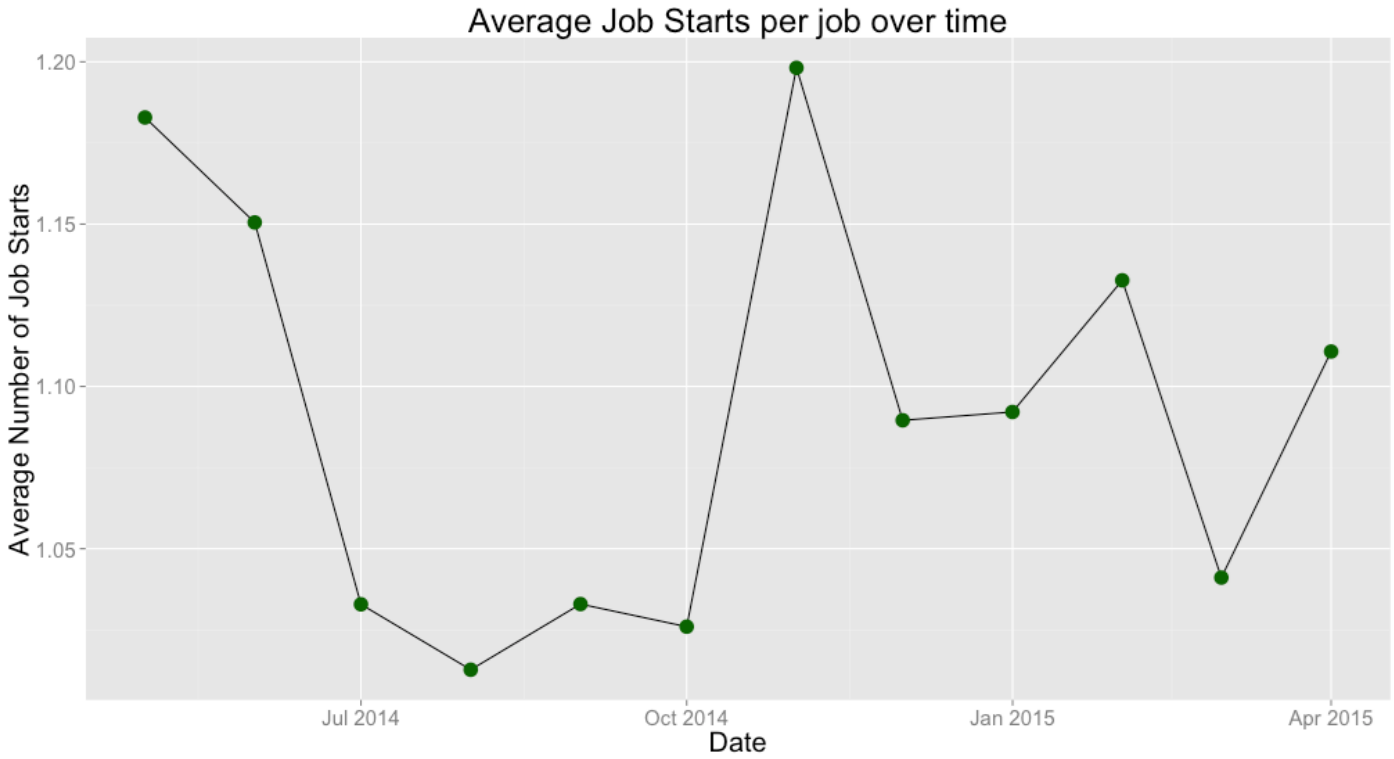
Can also generate plots of data with Kibana as well as dashboards

Plots and dashboards can be exported!

Some (potentially interesting) plots

- Queries/plots developed in Kibana, data exported as csv file and plotted using RStudio/ggplot

Average number of job starts over time

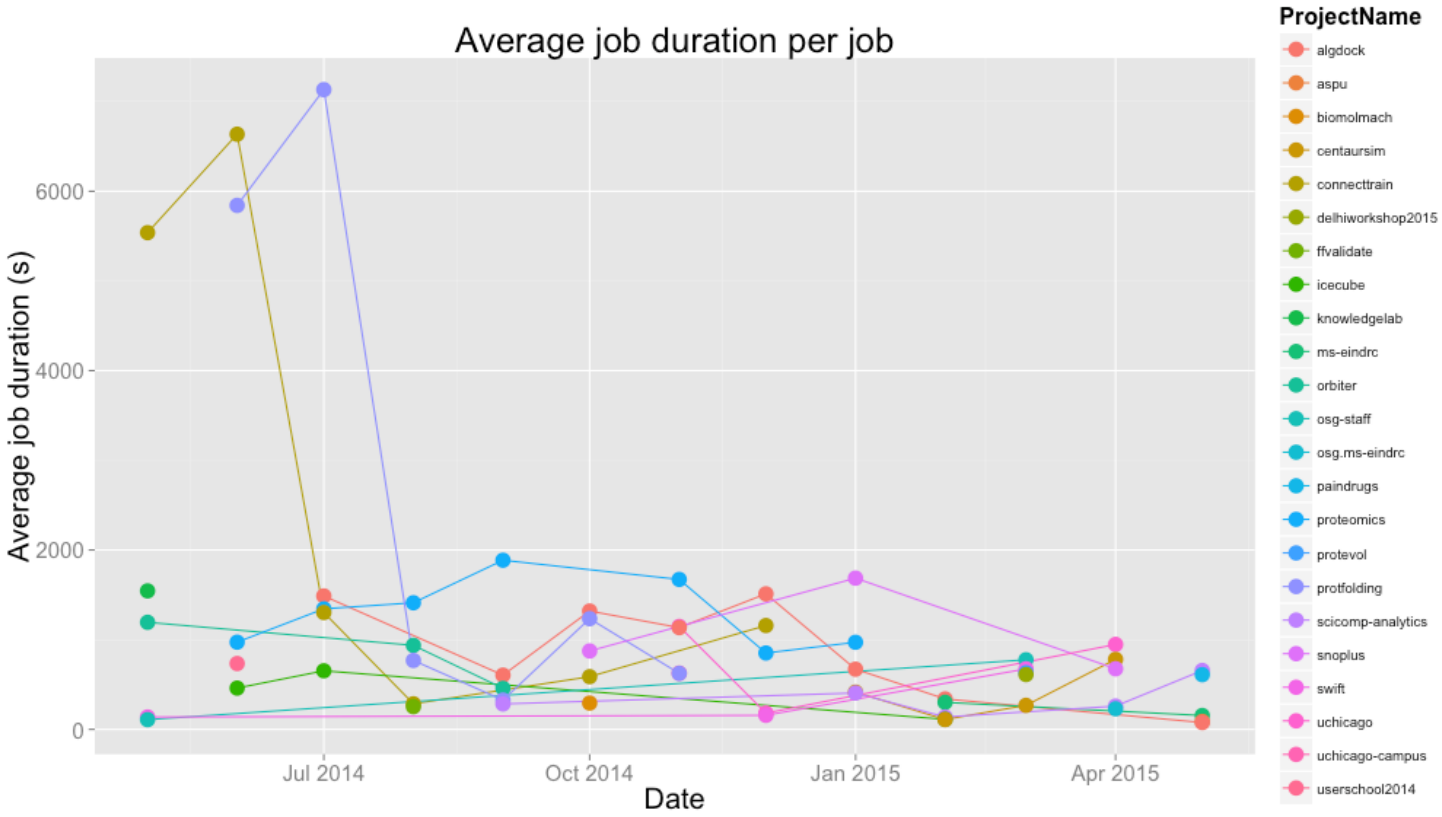


94% of jobs succeed on first attempt

Note: most jobs use periodic hold and period release ClassAds to retry failed jobs

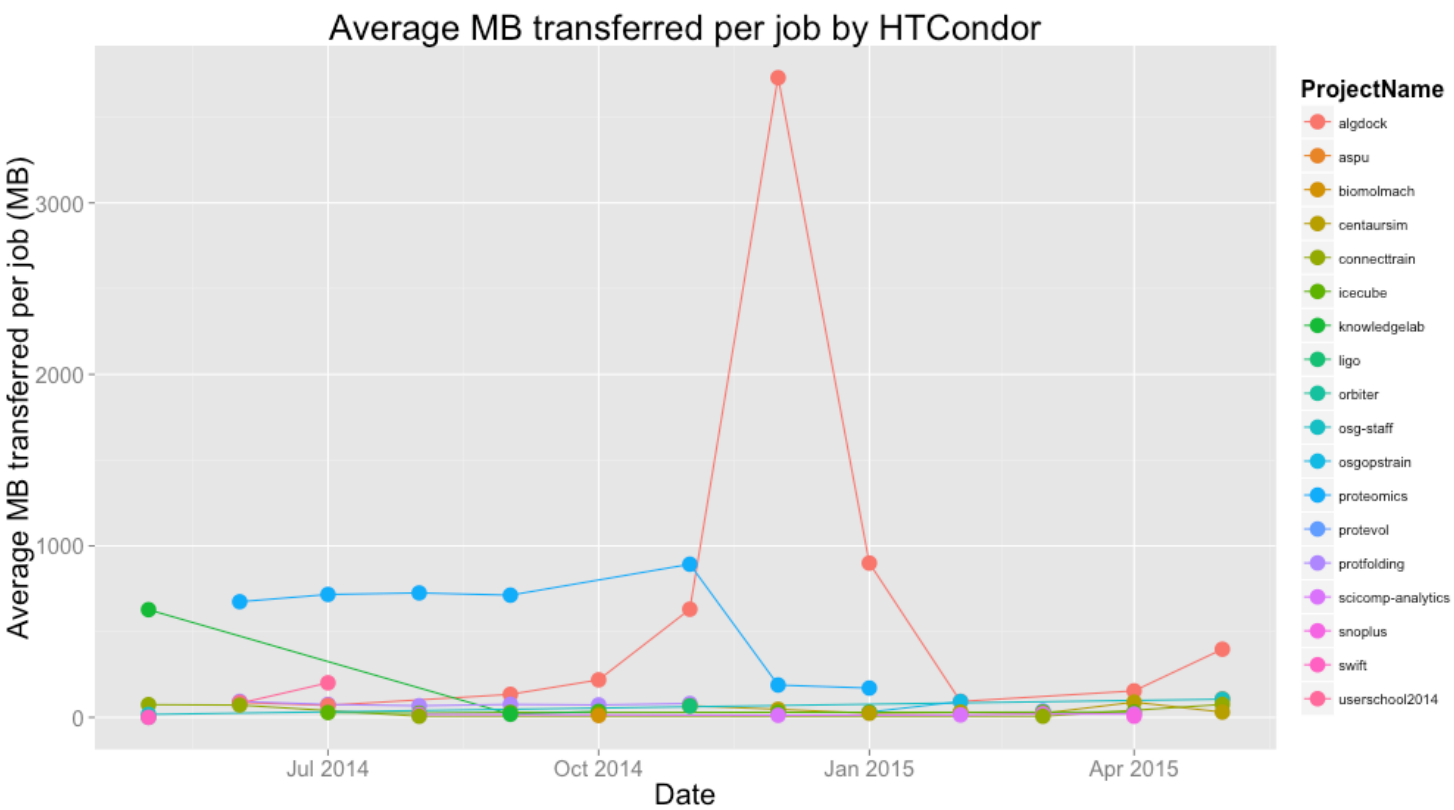
Thus invalid submissions may result in multiple restarts, inflating this number

Average job duration on compute node



Plot of the average job duration on compute nodes, majority of jobs complete within an hour with a few projects having outliers (mean: 1050s, median: 657s)

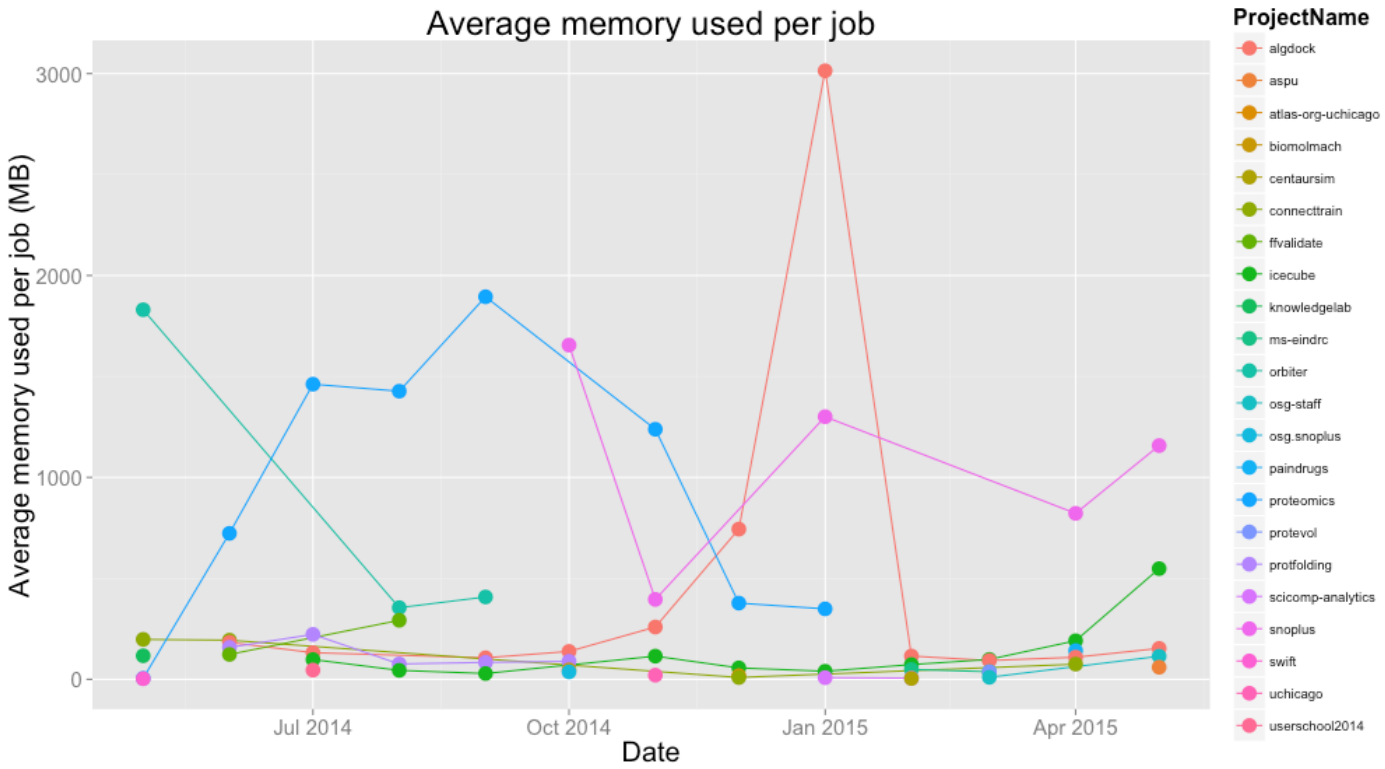
Average bytes transferred per job using HTCondor



Most jobs transfer significantly less than 1GB per job through HTCondor (mean: 214MB, median: 71MB)

Note: this does not take into account data transferred by other methods (e.g. wget)

Memory usage



Most jobs use less than 2GB of memory with proteomics and snoplus projects having the highest average memory utilization

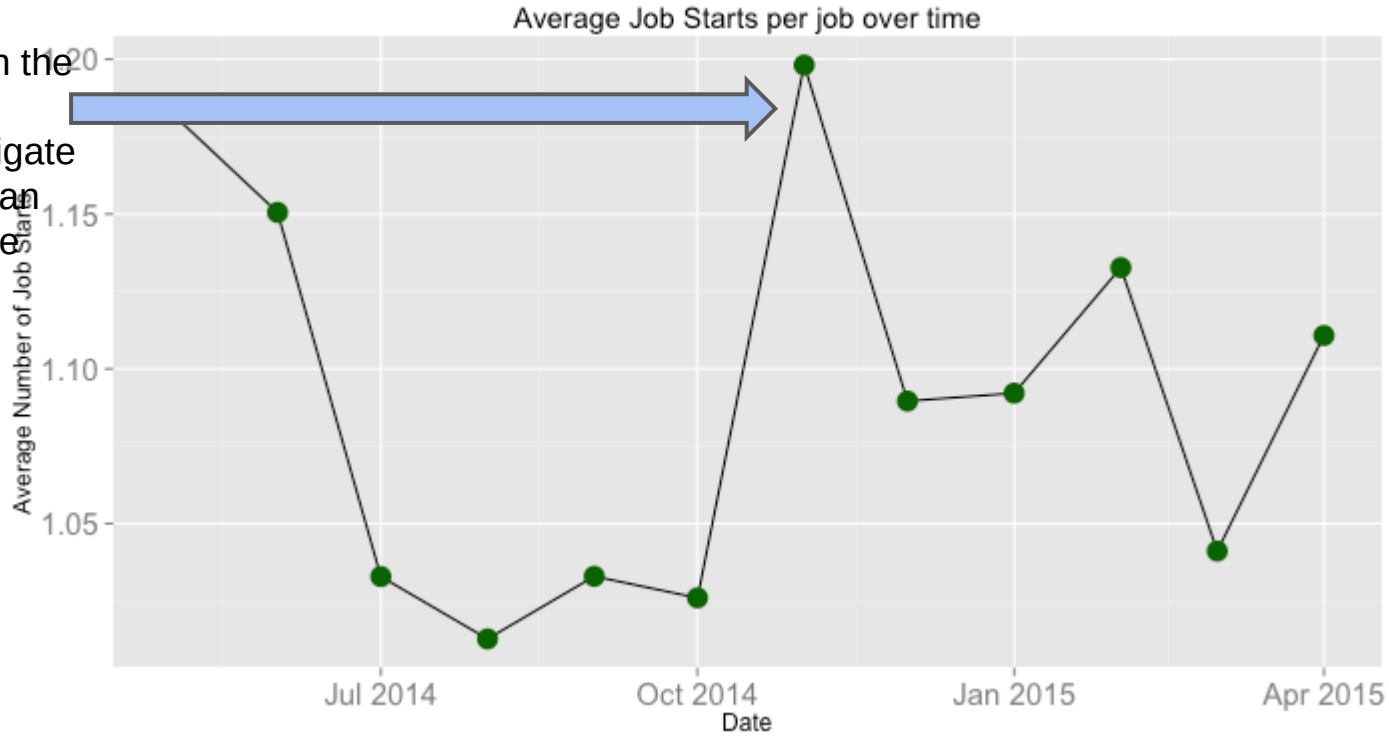
Mean use: 378 MB
Median use: 118 MB

Other uses

- Analytics framework is not just for dashboards and pretty plots
- Can also be used to troubleshoot issues

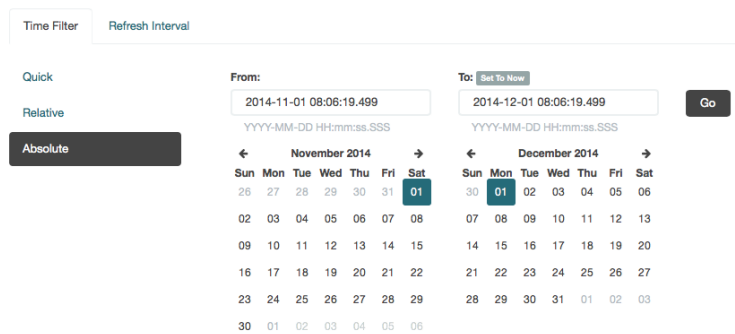
Troubleshooting a job

In November 2014, there was a spike in the number of restarts, would like to investigate this and see if we can determine the cause and fix



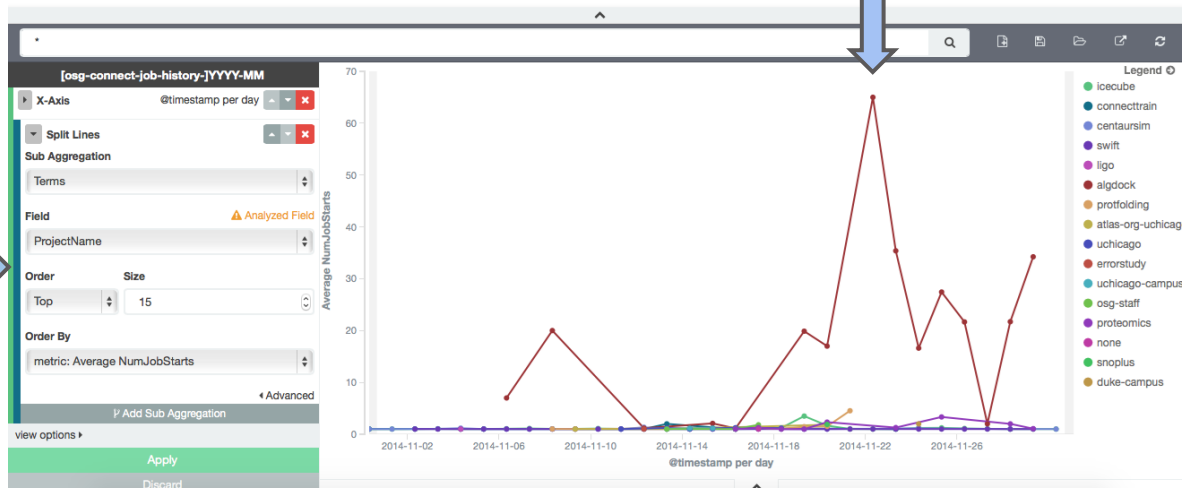
Troubleshooting part 2

Select the time range in question



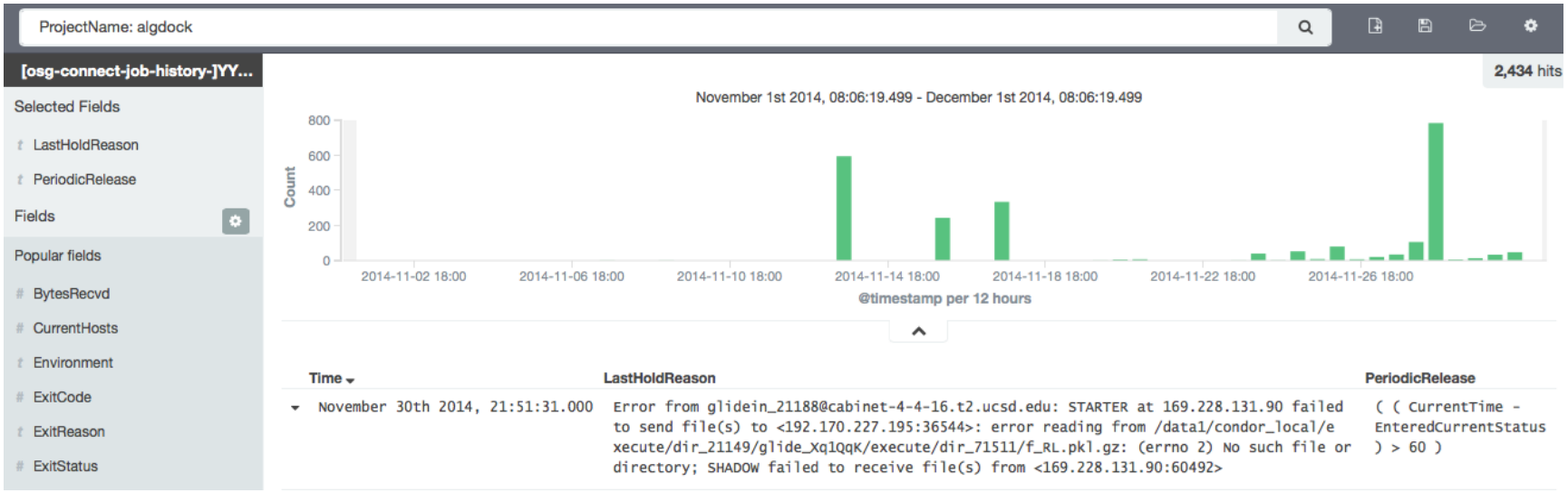
algdock project seems to be responsible for most of the restarts

Split out the projects with the most restarts



Troubleshooting concluded

Next, we go to the discover tab in Kibana and search for records in the relevant time frame and look at the classads, it quickly becomes apparently that the problem is due to a missing output file that results in the job being held combined with a PeriodicRelease for the job



Future directions

- Update schedd probe to use LogReader API to get classad updates rather than querying schedd for all classads
- More and better analytics:
  - Explore data more and pull out analytics that are useful for operational needs
  - Update user dashboards to present information that users are interested in

Links

- Github repo - https://github.com/DHTC-Tools/logstash-confs/tree/master/condor

Questions? Comments?