An opportunistic HTCondor pool inside an interactive-friendly Kubernetes cluster
Outline

• Where do I come from?
• What we did?
• How is it working?
• Looking head
The Pacific Research Platform

- The PRP originally created as a regional networking project
  - Establishing end-to-end links between 10Gbps and 100Gbps
The Pacific Research Platform

• The PRP originally created as a regional networking project
  • Establishing end-to-end links between 10Gbps and 100Gbps

• Expanded nationally since
  • And beyond, too
Recently the PRP evolved in a major resource provider, too
  • Because scientists really need more than bandwidth tests
  • They need to share their data at high speed and compute on it, too

The PRP now also provides
  • Extensive compute power – About 330 GPUs and 3.5k CPU cores
  • A large distributed storage area - About 2 PBytes

Select user communities now directly use all the resources PRP has to offer
  • Still doing all the network R&D in the same setup, too
  • We call it the Nautilus cluster
Kubernetes as a resource manager

- Industry standard
  - Large and active development and support community
- Container based
  - More freedom for users
- Flexible scheduling
  - Allows for easy mixing of service and user workloads
Designed for interactive use

Users expect to get what they need when they need it

- Makes for very happy users

Congestion happens only very rarely

- And is typically short in duration
Opportunistic use

No congestion → Idle compute resources → Time for opportunistic use
### Kubernetes priorities

- **Priorities natively supported in Kubernetes**
  - Low priority pods only start if no demand from higher priority ones

- **Preemption out of the box**
  - Low priority pods killed the moment a high priority pod needs the resources

- **Perfect for opportunistic use**
  - Just keep enough low-priority pods in the system

---

https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/

---

HTCondor Week, May 2019
HTCondor as the OSG helper

PRP wanted to give opportunistic resources to Open Science Grid (OSG) users
  • Since they can tolerate preemption

But OSG does not have native support for Kubernetes
  • Supports only resources provided by batch systems

We thus instantiated an HTCondor pool
  • As a fully Kubernetes/Containerized deployment
HTCondor in a (set of) container(s)

- Just create an image with HTCondor binaries in it!
- Configuration injected through Kubernetes pod config

Putting HTCondor in a set of containers is not hard

HTCondor deals nicely with ephemeral IPs

- The Collector must be discoverable – Kubernetes service
- Everything else just works from there

Persistency needed for the Schedd(s)

- And potentially for the Negotiator, if long term accounting desired
- Everything else can live off ephemeral storage
Service vs Opportunistic

**Collector and Schedd(s) deployed as high priority service pods**
- Should be running at all times
- Few pods, not high CPU or GPU users, so OK
- Using Kubernetes Deployment to re-start the pods in case of HW problems and/or maintenance
- Kubernetes Service used to get a persistent routing IP to the collector pod

**Startds deployed as low priority pods**
- Hundreds of pods in the Kubernetes queue at all times, many in Pending state
- HTCondor Startd configured to accept jobs as soon as it starts and forever after
- If pod preempted, HTCondor gets a SIGTERM and has a few seconds to go away

Pure opportunistic
Then came the users

- Everything was working nicely, until we let in real users
- OSG users got used to rely on Containers
- But HTCondor Startd already running inside a container!
- Well, until we had more than a single user
- So they can use any weird software they like
- Cannot launch a user-provided container

Not without elevated privileges
Then came the users

- Well, until we had more than a single user
- So they can use any weird software they like
- Cannot launch a user-provided container
- How many of each kind?

Everything was working nicely, until we let in real users

OSG users got used to rely on Containers

But HTCondor Startd already running inside a container!

So I need to provide user-specific execute pods
Dealing with many opportunistic pod types

| Having idle Startd pods not OK anymore | - A different kind of pod could use that resource  
|                                       | - A glidein-like setup would solve that |
| Keeping pods without users not OK anymore | - They will just terminate without ever running a job  
|                                           | - Who should regulate the “glidein pressure”? |
| How do I manage fair share between different pod types? | - Kubernetes scheduling is basically just priority-FIFO |
| How am I to know what Container images users want? | - Ideally, HTCondor should have native Kubernetes support |
Dealing with many opportunistic pod types

- A different kind of pod could use that resource
- A glidein-like setup would solve that

Having idle Startd pods not OK anymore

- They will just terminate without ever running a job
- Who should regulate the “glidein pressure”?

Keeping pods without users not OK anymore

- Kubernetes scheduling is not only just priority-FIFO

How do I manage fair share between different pod types?

- Ideally, HTCondor should have native Kubernetes support

How am I to know what Container images users want?

I know how to implement this.
Dealing with many opportunistic pod types

| Having idle Startd pods not OK anymore | • A different kind of pod could use that resource  
|                                         | • A glidein-like setup would solve that |
| Keeping pods without users not OK anymore | • They will just terminate without ever running a job  
|                                         | • Who should regulate the “glidein pressure”? |
| How do I manage fair share between different pod types? | • Kubernetes scheduling is priority-FIFO |
| How am I to know what Container images users want? | • Ideally, HTCondor should have native Kubernetes support |

I was told this is coming.
Dealing with many opportunistic pod types

| Having idle Startd pods not OK anymore | • A different kind of pod could use that resource  
|                                         | • A glidein-like setup would solve that |
| Keeping pods without users not OK anymore | • They will just terminate without ever running a job  
|                                         | • Who should regulate the “glidein pressure”? |
| How do I manage fair share between different pod types? | • Kubernetes scheduling  
| How am I to know what Container images users want? | • Ideally, HTCondor should have native Kubernetes support  

In OSG-land, glideinWMS solves this for me.
Dealing with many opportunistic pod types

Having idle Startd pods not OK anymore
- A different kind of pod could use that resource
- A glidein-like setup would solve that

Keeping pods without users not OK anymore
- They will just terminate without ever running a job
- Who should regulate the “glidein pressure”?

How do I manage fair share between different pod types?
- Kubernetes scheduling is basically just priority-FIFO

How am I to know what Container images users want?
- Ideally, HTCondor should have native Kubernetes support

No concrete plans on how to address these yet.
Dealing with many opportunistic pod types

For now, I just periodically adjust the balance
- A completely manual process

Currently supporting only a few, well behaved users
- Maybe not optimal, but good enough

But looking forward to a more automated future
Are side-containers an option?

Ideally, I do want to use user-provided, per-job Containers

- Running HTCondor and user jobs in separate pods not an option due to opportunistic nature

But Kubernetes pods are made of several Containers

- Could I run HTCondor in a dedicated Container?
- Then start the user pod in a side-container?

Pretty sure currently not supported

- But, at least in principle, fits the architecture
- Would also need HTCondor native support
Will nested Containers be a reality soon?

It has been pointed out to me that latest CentOS supports unprivileged Singularity.

Have not tied it out

• Probably I should

Cannot currently assume all of my nodes have a recent-enough kernel

• But eventually will get there
Looking ahead

Looking forward to a more automated future

Will do what I have to myself

Would be happier if I could use off-the-shelf solutions
A final picture

- Opportunistic GPU usage over the past few months
Summary

• We created an opportunistic HTCondor pool in the PRP Kubernetes cluster
  • OSG users can now use any otherwise-unused cycles
• The lack of nested containerization forces us to have multiple execute pod types
• Some micromanagement currently needed, hoping for more automation in the future
Acknowledgments

This work was partially funded by US National Science Foundation (NSF) awards CNS-1456638, CNS-1730158, ACI-1540112, ACI-1541349, OAC-1826967, OAC 1450871, OAC-1659169 and OAC-1841530.