2024 Summer Fellows
Where In The World Am I?

Neha Talluri
Mentor: Jason Patton
What is a Glidein?

- A glidein runs on shared resources to create more execution points
- Glideins assess the worker node they are on by gathering information and utilizing pre-configured data
- Location of a Glidein is currently not discovered or provided
  - Institutions may not be co-located with the resource
IP Geolocation Problematic

Kansas problem
“Where Am I?”

• Location: is the distance to some known entity
• Distance: Network latency and hops
• Answer “Where are glideins in relation to known entities?”
  • Entities = OSDF Caches
What I’ve Been Doing

- Learning the glideinWMS system through running OSPool jobs and kicking off glideins
- Developing glidein scripts to gather network information and advertise this information to the machine ad and glidein logs
- Figuring out how to answer “Where am I?”
  - Testing IP geolocation using different IP addresses
  - Figuring out how to use tracepath to figure out latency and hops
Questions?
Machine Learning for OSPool Failure Classification

Thinh Nguyen
Mentor: Justin Hiemstra
An Example

- Lifecycle of a job
- During a job’s execution it can go on hold for various reasons
- User’s discretion to release or remove it
Using AI to Make the **Inference**

- **Why AI?**
  - OSPool is a Dynamic system
  - Continual learning
Usage

[tdnguyen25@ap2002 logs]$ inference 1741531 ./job.log
How?

- Each job has a log file describing its lifecycle
- Format these logs into a time-series structure
- Use model that accounts for temporal patterns
  - e.g. Long Short-Term Memory (LSTM) neural network
Bonus

- Can the model provide information as to why the job went on hold?
Thank you,

Questions?

https://github.com/super10099/Machine-Learning-for-OSPool-Failure-Classification
Expanding Pelican Origin Monitoring

Patrick Brophy
Who Am I?

- My name is Patrick Brophy
- I am a senior at UW Madison studying computer science
- CHTC Fellow working with Haoming Meng

London

Lucky
Problem: Diagnosing an Origin’s Health

- PelicanOrigins are the backbone to a data federation
  - Connects and Serves an object store
- Origins are critical within a federation, no origins -> no data
- If an origin goes down so does the data that it was serving
  - A staging device can’t stage data if it can’t access an origin
- Diagnosing why an origin is failing is difficult with the current tooling
Improving the Dashboard

• I conducted a user study with several Pelican system admins from CHTC and OSDF
• Feedback from system admins guided design choices
What’s Next?

- More Metrics!
  - XRootD protocol-level (HTTP) metrics
- Alerting users of issues such as outages or warnings
- Reporting performance metrics for some period of time (day, week, month)
  - Staging Device miss rate
  - # of Objects accessed
  - Total bytes transferred
Thank you!

Please come and ask me questions!
Integrating Pelican with Pytorch

CHTC Summer Fellow: Kristina Zhao
Mentor: Ian Ross, Emma Turetsky
Introduction

Pelican

PyTorch

HTCondor

Integrating Pelican with Pytorch

Pelican Origin

Dataset Class (Pytorch)

DataLoader (Pytorch)

Batches

Model
Problem

Data Accessibility
  Large Size
  Remote

Performance:
  low metadata latency
  high data throughput
**Goals:**

Streamlined Workflows

CLI -> Pelicanfs (implement fsspec)

Smoother integration

Efficient Data Handling

Make our AI Researchers happier 😊

Integrating Pelican with PyTorch:

- Pelican Origin
- Dataset Class (PyTorch)
- DataLoader (PyTorch)
- Batches
- Model

CHTC  PATH  PARTNERSHIP to ADVANCE THROUGHPUT COMPUTING
Methodology

Research
- Pelican, fsspec, Pytorch data flows and requirements
- File format, file size, resource, limitation…

Benchmark

Develop tools/libraries

Tutorial and Documentation
Methodology

Research on Pelican and PyTorch

Benchmark
- local
- Pelicanfs
- Pelicanfs+Local Cache
- Pelicanfs+zip file

Develop tools/libraries

Tutorial and Documentation

Integrating Pelican with PyTorch
Methodology

Research on Pelican and PyTorch data needs

Benchmark

Develop tools/libraries

- Pelicanfs
- Pelican connector?

Tutorial and Documentation

CHTC
Methodology

Research on Pelican and PyTorch data needs

Benchmark

Develop tools/libraries
- Pelicanfs
- Pelican connector?

Tutorial and Documentation
Methodology

Research on Pelican and PyTorch data needs

Benchmark

Develop tools/libraries

Tutorial and Documentation
Thank you!

Discussion and Problems welcome!

Kristina Zhao
hhao292@wisc.edu
Enhancing the Building of the OSG Container Images

Pratham Patel
Who Am I?

- Hometown: Beloit, WI
- Senior at UW-Madison studying CS & DS
- I’m pretty awesome…

Just deal with it!
Enhancing the OSG Container Build System
A Three-Phase Approach to Versatility and Efficiency

Abstract:

- CHTC builds images for sites to run and for use internally.
- Images are based on upstream OS container images
- We build all images at least once a week

Focus:

- Adding versatility and streamlining the build process within the OSG images repository.
- Three-phase approach:
  1. Customizable build instructions for each image.
  2. Dynamic trigger for external repositories.
  3. Compatibility and support for ARM-based systems.
Image X

- OSG 3.6 + EL7
- OSG 23 + EL9

Development
Testing
Release

= 6 individual build for image X

Multiply that by 27 images built multiple times a week and we run into issues…
Background

Current State:

- **GitHub Actions Workflow:**
  - Automates building and pushing container images.
  - Triggered by specific conditions and events.

- **Monolithic Design:**
  - Lack of flexibility in the build process.
  - All images built using a single, unified workflow.

- **Additional Features:**
  - Not supported for ARM architecture.
Project Requirements

Versatility:

- Mechanism for custom build processes through a unique configuration file.
- Default instructions in absence of unique configuration file.

Trigger Mechanism:

- Located in the images repository.
- Activates updates with Pelican and other external repositories.

ARM Compatibility:

- Add support for building native ARM-based systems.
- Aim for ARM-optimized Pelican images.
Solution

Phase 1: Customizable Build Instructions

- **Objective:** Establish an advanced image repository framework with configurable build instructions.
- **Implementation Paths:**
  - Dynamic parameters for building images.
  - Modularize workflow into reusable components.

Phase 2: Triggering the Pelican Repository

- **Objective:** Integrate with the external repositories using a trigger.
- **Implementation Paths:**
  - Create some trigger in the images repository.
  - Use GitHub Actions to monitor and trigger Pelican repository and others.
  - Develop GitHub Action in Pelican repository to handle the trigger for updates.
Conclusion

Summary:

● Implementing a three-phase approach that enhances flexibility, efficiency, and future-readiness.
● Custom build instructions, trigger for external repositories like Pelican, and ARM-based support.

Next Steps:

● Refine customizable build instructions
● Work with Pelican team to determine trigger strategies
CHTC Fellowship: Tracking Server Inventory and Elevation

Ben Staehle
Who am I?

- My name is Ben Staehle
- Madison native - currently at UW-Madison
- This summer - CHTC Fellowship
- Working with Joe Bartkowiak on “Tracking Server Inventory and Elevation”
What are we solving?

- CHTC maintains more than 1200 assets
- System administrators are responsible for maintaining inventory records
- Interested stakeholders include UW-Madison, Morgridge Institute for Research
- Previous internal asset tracking was cumbersome
Thank you!

Questions?
Performance Monitoring in the Schedd

Wil Cram
Greg Thain
The Motivation

• The Schedd process handles commands
• Sometimes it slows down without warning
• How to find the root cause?
• Admins: who is the problem?
Definition 1.1

A task is an object with an associated “time cost”

A queue is an object that accepts incoming tasks and works on them one at a time
The Problem

• Consider a queue $Q$ with categories $1, 2, \ldots, n$
• The queue records the total time spent processing each category
• How do we determine real-time data about this system?
Enter Sampling

• We can record the totals every x seconds
• The time spent over some window is
  \[\text{new\_runtime} - \text{old\_runtime}\]
• We can divide by the sum of totals to get a relative proportion
• This tells us what is taking up time in the queue
The Schedd is a Queue!
(Basically)
The Schedd:

• Operates on tasks one at a time
• Has categories (commands, handling jobs, etc.)
• Publishes totals through ClassAds
Trivial, Right?

2024-06-28 15:32:04
Sample Window: 5 s
CPU Time: 0.088 s
CPU Usage: 1.77%

Commands (0.66%)
* wil@fourier: 0.34%
  - command_query_job_ads: 0.31%
  - command_query_ads: 0.04%
* test@fourier: 0.31%
  - command_query_job_ads: 0.31%
Other (1.11%)

Waiting 2 seconds
Wrong!

• Condor publishes the average % of time that CPU is recently “busy” (20min window)
• Multiply by window size to get CPU time?
• If the percentage doesn’t move much, sure
Sample Window: 5 s
CPU Time: 1.268 s
CPU Usage: 25.35%

Commands (14.37%)
* test@fourier: 10.31%
  - handle_q: 10.29%
  - reschedule_negotiator: 0.02%
* wil@fourier: 2.53%
  - command_query_job_ads: 2.44%
  - command_query_ads: 0.09%
* condor@child: 1.53%
  - handle_q: 1.52%

TotalFsync (3.27%)

Timers (0.97%)
* StartJobHandlerRuntime: 0.47%
* timeoutRuntime: 0.27%
* SelfDrainingQueuetimerHandlerjob_is_finished_queueRuntime: 0.15%
* checkContactQueueQueueRuntime: 0.08%

Other (6.74%)
Wrong!

- Condor publishes the average % of time that CPU is recently “busy” (20min window)
- Multiply by window size to get CPU time?
- If the percentage doesn’t move much, sure
- This average falls behind during a traffic spike
Sample Window: 5 s
CPU Time: 1.834 s
CPU Usage: 36.68%

WARNING: Summed statistics exceeded CPU time! These values are renormalized! This typically happens when traffic spikes faster than CPU Usage can update.

Commands (25.52%)
* test@fourier: 13.73%
  - handle_q: 13.72%
  - reschedule_negotiator: 0.02%
* condor@child: 11.76%
  - handle_q: 11.76%
* wil@fourier: 0.02%
  - command_query_ads: 0.02%

TotalFsinc (9.79%)

Timers (1.37%)
* SelfDrainingQueueTimerHandlerJob_is_finished_queueRuntime: 0.78%
* timeoutRuntime: 0.59%

Other (0.00%)

Waiting 4 seconds
Wrong!

- Condor publishes the average % of time that CPU is recently “busy” (20min window)
- Multiply by window size to get CPU time?
- If the percentage doesn’t move much, sure
- This average falls behind during a traffic spike
- We trust our own recent totals
- However, we lose information
Future Plans

• Improving visibility at a glance
• More process coverage
• Give intelligent warnings to users

“User condor_q_spammer has taken over 25% of busy time for the past hour. Maybe don’t let them do that?”