IceCube’s SkyDriver

An Application of the Event Workflow Management System for Scalable Solutions of Distributed Workflows

Ric Evans
Research Software Engineer
UW-Madison
IceCube / WIPAC

Throughput Computing
2024 – HTC 24
The Event Workflow Management System (EWMS) Question

How can we take a workload, consisting of millions or billions of tasks, and group it into tens of thousands of jobs?
Event-Granular HTC Workflows

To be most efficient, we want to subdivide a workflow into “smallest” unit of work (“events”)

- Multi-Messenger Astrophysics events (IceCube and LIGO triggers)
- Astronomical observations (images)
- Cryogenic electron microscopy (cryo-EM) data
- Optical Character Recognition on pages in a book
- and more!
HTCondor’s Traditional Use

HTCondor is great at aggregating distributed resources and orchestrating workflows, but…

- Imposes **1:1 job-task** mapping
- Needs $O(>30 \text{ min})$ jobs to be most efficient
  - Task lifetime $>>$ Startup+Scheduling time
HTCondor’s Traditional Use

HTCondor is great at aggregating distributed resources and orchestrating workflows, but…

- Imposes **1:1 job-task** mapping
- Needs $O(>30 \text{ min})$ jobs to be most efficient
  - Task lifetime >> Startup+Scheduling time

If we want to work on events

- Much **shorter runtime per task**
- **1:N job-task** mapping
- **Dynamic allocation** of inputs and outputs
IceCube Neutrino Observatory

*Why does IceCube need many, many short-lived tasks?*
IceCube Neutrino Observatory

The IceCube Neutrino Observatory is a cubic kilometer neutrino telescope located at the geographic South Pole premier facility for detecting neutrinos > 10 GeV, particularly > 1 TeV astrophysical neutrinos.
A neutrino is detected by IceCube!

Where did it come from?
Where do we need to point other telescopes for immediate follow-up observations?
The Problem

We need to reconstruct a **Sky Map**

Most accurate and detailed directional reconstruction comes by scanning across the sky in varying granularity: $O(100k)$ pixels

“night sky”

HEALPix algorithm
The Solution

The (Original) Skymap Scanner

1. **Preempt** $N$ HTCondor nodes for **immediate availability**

2. **Generate** $O(100k)$ **events** (5-tuples)

3. **Group** $O(1k)$ events into $N$ “input” object
   - 1 job gets 1 object, $O(1k)$ events

4. **Submit** to HTCondor for $N$ jobs

5. **Wait** for every job to finish while collecting $N$ transferred output objects

6. **Assemble** resulting skymap
   - Produce the most **probable direction** and error
The Problem with the Solution

Three False Assumptions

➢ We know how to group input events because we have a homogeneous infinitely big compute pool. We have a heterogeneous and finite pool.

➢ Task processes will never fail. CPU crashes happen. What if last event fails?

➢ No one will be mad if we take away their computing resources. Yes they will, especially before a conference.
Three False Assumptions

➢ We know how to group input events because we have a homogeneous infinitely big compute pool.

  We have a heterogeneous and finite pool
Three False Assumptions

➢ We know how to group input events because we have a homogeneous infinitely big compute pool.

   We have a heterogeneous and finite pool

➢ Task processes will never fail.

   CPU crashes happen. *What if last event fails?*
Three False Assumptions

➢ We know how to group input events because we have a homogeneous infinitely big compute pool.

We have a heterogeneous and finite pool

➢ Task processes will never fail.

CPU crashes happen. *What if last event fails?*

➢ No one will be mad if we take away their computing resources.

Yes they will, especially before a conference
Our Vision

EWMS Design + SkyDriver Application

- Design a generalized design, the Event Workflow Management System
- Make an instance of EWMS at IceCube, called SkyDriver (with a few domain-specific add-ons)
- Run Skymap Scanner tasks within the SkyDriver service
SkyDriver-EWMS Architecture

User requests a new scan
SkyDriver-EWMS Architecture

Workflow Management Service
SkyDriver-EWMS Architecture

Data Distribution Service & Message Queue Broker
SkyDriver-EWMS Architecture

SkyDriver sends events to MQ
SkyDriver-EWMS Architecture

Task Management Service on HTCondor Access Point
SkyDriver-EWMS Architecture

HTCondor starts up jobs
SkyDriver-EWMS Architecture

Task Pilot (Worker) on HTCondor Execution Point
SkyDriver-EWMS Architecture

Workers retrieve input-events & send output-events via the MQ
SkyDriver-EWMS Architecture

SkyDriver receives output events from MQ
SkyDriver-EWMS Architecture

SkyDriver receives output events from MQ
SkyDriver-EWMS Architecture

SkyDriver receives output events from MQ
SkyDriver-EWMS Architecture

SkyDriver receives output events from MQ
Motivation & Goals

How can we help HTCondor support multiple events per job?

1:N job-task pattern
Complement HTCondor's Capabilities

Thrive in heterogeneous, dynamic environments
(faster CPUs do more work, etc.)
Our Goals

What does EWMS need to do?

➢ Complement HTCondor's Capabilities
  
  *Thrive in heterogeneous, dynamic environments (faster CPUs do more work, etc.)*

➢ Support Scientific Reproducibility
  
  *Build a robust, repeatable system*
Our Goals

What does EWMS need to do?

➢ Complement HTCondor's Capabilities
   *Thrive in heterogeneous, dynamic environments (faster CPUs do more work, etc.)*

➢ Support Scientific Reproducibility
   *Build a robust, repeatable system*

➢ A Service-First Design
   *Build a platform, not an application*
Our Goals

What does EWMS need to do?

➢ Complement HTCondor's Capabilities

  *Thrive in heterogeneous, dynamic environments (faster CPUs do more work, etc.)*

➢ Support Scientific Reproducibility

  *Build a robust, repeatable system*

➢ A Service-First Design

  *Build a platform, not an application*

➢ Make everyone happy :)

---

32
1. Complement HTCondor's Capabilities

*How can we work with heterogeneous clusters?*
How do we complement HTCondor?

A few of HTCondor's Exceptional Features:

➢ Guaranteed execution
➢ Extreme scalability
➢ Parallelization without reimplementation
➢ Success in heterogeneous environments
➢ Adaptable to user requirements

Paraphrased from the HTCondor Manual
Complement HTCondor

How can a job have dynamically allocated inputs, outputs, and tasks?

File-transfer system for task I/O (of events) will not suffice:

- 1:N tasks are complex
- No dynamic scaling task per job
Complement HTCondor

How can a job have dynamically allocated inputs, outputs, and tasks?

*File-transfer system for task I/O (of events) will not suffice:*

- 1:N tasks are complex
- No dynamic scaling task per job

*Message passing (MQ):*

- Separates event I/O from job mechanics
  - Additional *input(s)* are given when needed
  - *Outputs are immediately* relayed in real-time
- Doesn’t care about fluxuations in job count
  - *Can we increase/decrease number of jobs*?

- guaranteed execution
- extreme scalability
- parallelization without reimplementation
- success in heterogeneous environments
- adaptable to user requirements
Many possible protocols

➢ Low-level and foundational decision
➢ Expensive to change after implemented
Many possible protocols

- Low-level and foundational decision
- Expensive to change after implemented

Created software to be flexible with any of these:

- RabbitMQ
- Apache Pulsar
- NATS.io

(Not) Choosing an MQ Protocol

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guaranteed execution</td>
<td>☐</td>
</tr>
<tr>
<td>Extreme scalability</td>
<td>☑</td>
</tr>
<tr>
<td>Parallelization without reimplement</td>
<td>☐</td>
</tr>
<tr>
<td>Success in heterogeneous environments</td>
<td>☑</td>
</tr>
<tr>
<td>Adaptable to user requirements</td>
<td>☑</td>
</tr>
</tbody>
</table>
Resilient to CPU crashes – Built-in failover mechanism

➢ Ack-last & fail-fast paradigm
  ○ Acknowledge input event only when task is done
  ○ MQ will redeliver to another worker when no ack
  ○ “Dead Letter” queue for problem events

Backward compatible – invisible from user’s POV

➢ Existing physics algorithms use files as input

Complement HTCondor

Pilot-Based Workers

☑ guaranteed execution
☑ extreme scalability
☑ parallelization without reimplementations
☑ success in heterogeneous environments
☑ adaptable to user requirements
2. Support Scientific Reproducibility

*How can we be assured science results are not due to software bugs?*
Reproducibility

Versioning & Containerization

What software was used in this analysis?

➢ Need to document version identifier with results

What else can affect the software?

➢ Need to know what we’re testing is what we’re running in production
➢ Using containers guarantees consistent reuse

NASA wind tunnel test
Reproducibility

Put it all in a centralized database!

For every run of SkyDriver, store:

➢ Startup parameters
➢ User-defined tags
  ○ Used to find results, limited in size
➢ Metadata
  ○ Timestamps, basic runtime stats
➢ Results A.K.A. Skymaps
Reproducibility

Include Users Throughout the Process

Feedback-Driven Enhancements

➢ Don’t spend time designing a solution for no problem

Open Beta Testing (Gamma Testing)

➢ Advertised as a prototype, un-ready system, with an end date goal (Q4 2023) – team effort
➢ Created slack channel for this purpose, closed channel when beta testing was completed
3. A Service-First Design

What do users need to know to be successful?
If our system is not simple to onboard, it won’t be used!

HTTP / REST user interface

➢ Standardized JSON input – auto-documented
  ○ Validation by JSON Schema & OpenAPI

➢ Multiple image versions available, including feature-branch versions
  ○ SkyDriver uses Skymap Scanner Images
  ○ Allows users to test customizations
Looking Back and Forward

How’s EWMS going?
Looking Back and Forward

Challenges

➢ **How generalized of a system do we want?**

➢ **Many unique tools:** Kubernetes, Helm, Docker, Python
  Packaging, REST, Input Validation, …

➢ Some errors only appear at massive workflow scales

➢ Removing **tech debt** from original Skymap Scanner
  ○ Created “organically”
  ○ “How does this work?”... “I don’t remember.”

➢ Small development team size (1.1 full-time)

---

**Oversimplified Timeline**

- 2022: MQ-equipped Skymap Scanner
- 2022-23: SkyDriver
- 2024: Generalized EWMS
Looking Back and Forward

EWMS: Ongoing and Upcoming Features

- Release generalized EWMS *(currently in alpha)*
- Automatic job scaling by detecting MQ usage and availability of compute resources in HTCondor pool
- Real-time monitoring dashboard
- Support scheduling for DAG workflows
Looking Back and Forward

Summary

How can we take a workload, consisting of millions or billions of tasks, and group it into tens of thousands of jobs?

➢ Complement HTCondor's Capabilities
   Using message passing-equipped worker pilots to thrive in heterogeneous, dynamic environments

➢ Support Scientific Reproducibility
   Providing dependable software, developed with user feedback

➢ A Service-First Design
   Putting the user’s POV first, simple interfaces and removed complexities

➢ Made everyone happy :)
Acknowledgements

PIs (EWMS)
- Miron Livny
- Brian Bockelman
- Benedikt Riedel

Developers
- Ric Evans (me)
- Benedikt Riedel
- David Schultz

IceCubers
- Massimiliano Lincetto
- Tianlu Yuan
- Claudio Kopper
- Erik Blaufuss
- Christina Lagunas
- Robert Stein

National Science Foundation Funding
- OAC #2103963 + OPP #2042807
Thank You!
Two IceCube Use Cases

CASE 1: Massive Scale

Real-time Scans

Fast & Resource Intensive -> High Priority

➔ O(10k+) CPUs, spun up ASAP

CASE 2: Moderate Scale

Historical Catalog & Simulation

Steady/Predictable -> Lower Priority

➔ Varying # of CPUs, subject to availability
Reproducibility

Development Methodology

Minimum viable product
➢ Wait to implement enhancements until needed

Test every enhancement & bug fix
➢ Use non-domain specific data & workflows

Do enhancements in order of priority
➢ Track in GitHub
Reproducibility

Test, scale up, test, scale up, test, ...

1. Test at no scale – fast
   ➢ Test individual components

2. Test at mini scale – cheap
   ➢ 1 or 2 jobs in automated CI environment (Github Actions)

3. Test at large scale – conservative
   ➢ Use production cluster w/ downsized configuration

4. Test at full scale
   ➢ Use production configuration

5. Publish Release
SkyDriver – Worker / Scanner Client POV

1. TMS submits a worker to HTCondor which then schedules the compute resource. The worker includes:
   - EWMS Pilot
   - Script to execute
   - Information to connect to DDS: Auth Token, address, input & output topics.