A Pull-Based Task Distribution System for HEP Analysis: Hardening hq for Coffea-casa

Applicant: Michael Noamesi (Gettysburg College, Physics & Computer Science)
Mentors: Peter Fackeldey (Princeton University), Oksana Shadura (University of Nebraska-Lincoln)
Project Duration: 11 weeks
Start Date: 8 June 2026

Project Description

Dask is currently the de-facto execution backend for distributing tasks across HEP analysis infrastructures. However, for large-scale analyses involving tens or hundreds of thousands of tasks, Dask reaches its limits. The scaling limitation is primarily due to Dask's Python-based single-threaded central scheduler, which can only handle up to a few thousand tasks per second gracefully. The IRIS-HEP 200 Gbps Challenge [1] at coffea-casa surfaced this constraint empirically: scaling Dask to 2000+ workers with very complex graphs was identified as an area requiring further investigation, alongside memory pressure observed at scale.

One direction for addressing this is to replace the central scheduler with a message (task) queue that workers can concurrently fetch work from. This changes the system from a *push-based* model, in which a central scheduler submits work to workers, to a *pull-based* model, in which workers fetch work from a queue. Such a system involves a client to put tasks into a queue, the queue itself deployed on coffea-casa, and workers (managed by HTCondor or Kubernetes) that connect and fetch tasks from the queue. hep-queue (hq) [2], a prototype built by Peter Fackeldey, is one implementation of this idea. It uses a stateless TypeScript/Bun HTTP server, Redis (Dragonfly-compatible) as queue and state store, and Python workers that fetch tasks via HTTP. Workers run tasks as subprocesses, isolating the polling loop from user code and the GIL. Heavy payloads are deduplicated by reference counting in `client.map` [2]. Worker liveness is tracked through heartbeats. Redis snapshots provide recovery. Component benchmarks suggest a ~15× control-plane throughput ceiling over async-Python schedulers; the stateless server scales horizontally.

hq works as a prototype but is not yet ready for deployment at a multi-tenant analysis facility. Reaching that point requires adding TLS, lightweight worker telemetry, per-task software environments, dynamic worker scaling, robust task failure handling, and validation at scale on coffea-casa.

Project Goals

1. Characterize failure modes through fault injection; fix the recovery gaps.
2. Add TLS encryption to the HTTP boundary.
3. Add lightweight worker telemetry piggybacked on heartbeats as plain JSON (tasks/sec, success/failure counts, runtimes), surfaced through the hq-server.
4. Add support for custom per-task software environments, so individual tasks can run in their own pinned environments.
5. Package and stand up a test deployment on coffea-casa, with Kubernetes workers and the HTCondor pool as available.
6. Benchmark hq at scale on the AGC $\bar{t}t$ workload [3], using the Dask and TaskVine [4] results from the 200 Gbps Challenge as reference points.

Project Timeline

Weeks 1–2: Read the hq codebase. Deploy the full stack locally. Stand up the AGC $\bar{t}t$ workload on hq to establish a working baseline. Familiarize with the coffea-casa stack (HTCondor, Kubernetes, Dask Gateway, XCache).

Weeks 3–4: Add TLS to the HTTP boundary. Inject faults: kill workers mid-task, restart hq-server, restart Redis, partition the network. Document recovery behavior. Fix gaps.

Week 5: Extend heartbeats with lightweight JSON telemetry from workers (tasks/sec, success/failure counts, runtimes). Surface through hq-server endpoints.

Weeks 6–7: Implement custom per-task software environments (possibly using pixi-pack), so individual tasks can declare and run in their own pinned environments without rebuilding worker images.

Week 8: Helm chart and deployment manifests. Stand up a test deployment on coffea-casa with Kubernetes workers and the HTCondor pool as available.

Weeks 9–10: Run the AGC $\bar{t}t$ workload at increasing worker counts; measure throughput, latency, and recovery. Compare against published Dask and TaskVine numbers as reference points. If time permits during this stretch, prototype queue multiplexing for per-user fairness and observe its effect on contention.

Week 11: Deployment guide, final report, IRIS-HEP presentation, upstream pull requests.

References

[1] S. Albin *et al.*, “Tuning the CMS Coffea-casa facility for 200 Gbps Challenge,” arXiv:2503.12991 (2025).

[2] P. Fackeldey, “hq (hep-queue),” https://github.com/pfackeldey/hq (2026).

[3] A. Held, O. Shadura, “The IRIS-HEP Analysis Grand Challenge,” PoS ICHEP2022, 235 (2022).

[4] B. Sly-Delgado et al., "TaskVine: Managing In-Cluster Storage for High-Throughput Data Intensive Workflows," in Proc. SC '23 Workshops, ACM (2023).DOI: 10.1145/3624062.3624277.