

Agents and Birds

Leveraging LLMs for Throughput Computing

Brian Bockelman, HTC'26

So I was talking to Miron one day...

- ▶ We are always concerned about dividing CHTC into “HTCondor” and “Pelican” camps!
 - ▶ **Miron:** We have an enormous investment in software! You are throwing it all away and reinventing the wheel.
 - ▶ **Me:** I’m not enthusiastic about attaching dusty old C++ libraries (HTCondor) to my shiny new Go code (Pelican).
- ▶ Ah-ha moment!

We can leverage the knowledge in the software without linking against the library.

Matchmaking: Distributed Resource Management for High Throughput Computing

Rajesh Raman
Miron Livny
Marvin Solomon
University of Wisconsin
1210 West Dayton Street
Madison, WI 53703
{raman,miron,solomon}@cs.wisc.edu

Abstract

Conventional resource management systems use a system model to describe resources and a centralized scheduler to control their allocation. We argue that this paradigm does not adapt well to distributed systems, particularly those built to support high-throughput computing. Obstacles include heterogeneity of resources, which make uniform allocation algorithms difficult to formulate, and distributed ownership, leading to widely varying allocation policies.

Faced with these problems, we developed and implemented the classified advertisement (classad) matchmaking framework, a flexible and general approach to resource management in distributed environment with decentralized ownership of resources. Novel aspects of the framework include a semi-structured data model that combines schema, data, and query in a simple but powerful specification language, and a clean separation of the matching and claiming phases of resource allocation. The representation and protocols result in a robust, scalable and flexible framework that can evolve with changing resources.

The framework was designed to solve real problems encountered in the deployment of Condor, a high throughput computing system developed at the University of Wisconsin—Madison. Condor is heavily used by scientists at numerous sites around the world. It derives much of its robustness and efficiency from the matchmaking architecture.

1. Introduction

A principal consideration of resource management systems is the efficient assignment of resources to customers. The problem of making such efficient assignments is re-

ferred to as the *resource allocation or scheduling* problem, and it is commonly formulated in the context of a *scheduling model* that includes a *system model*, which is an abstraction of the underlying resources. The system model provides information to the allocator regarding the availability and properties of resources at any point in time. The allocator uses this information to allocate resources to tasks so as to optimize a stated performance metric. This paradigm is useful for *high performance applications*, which have tight constraints. Efficient scheduling of resources is critical in meeting these constraints. However, an increasing number of organizations now have environments that are not amenable to this resource management paradigm; their customers are interested in *throughput* and their computing resources are *distributively owned*.

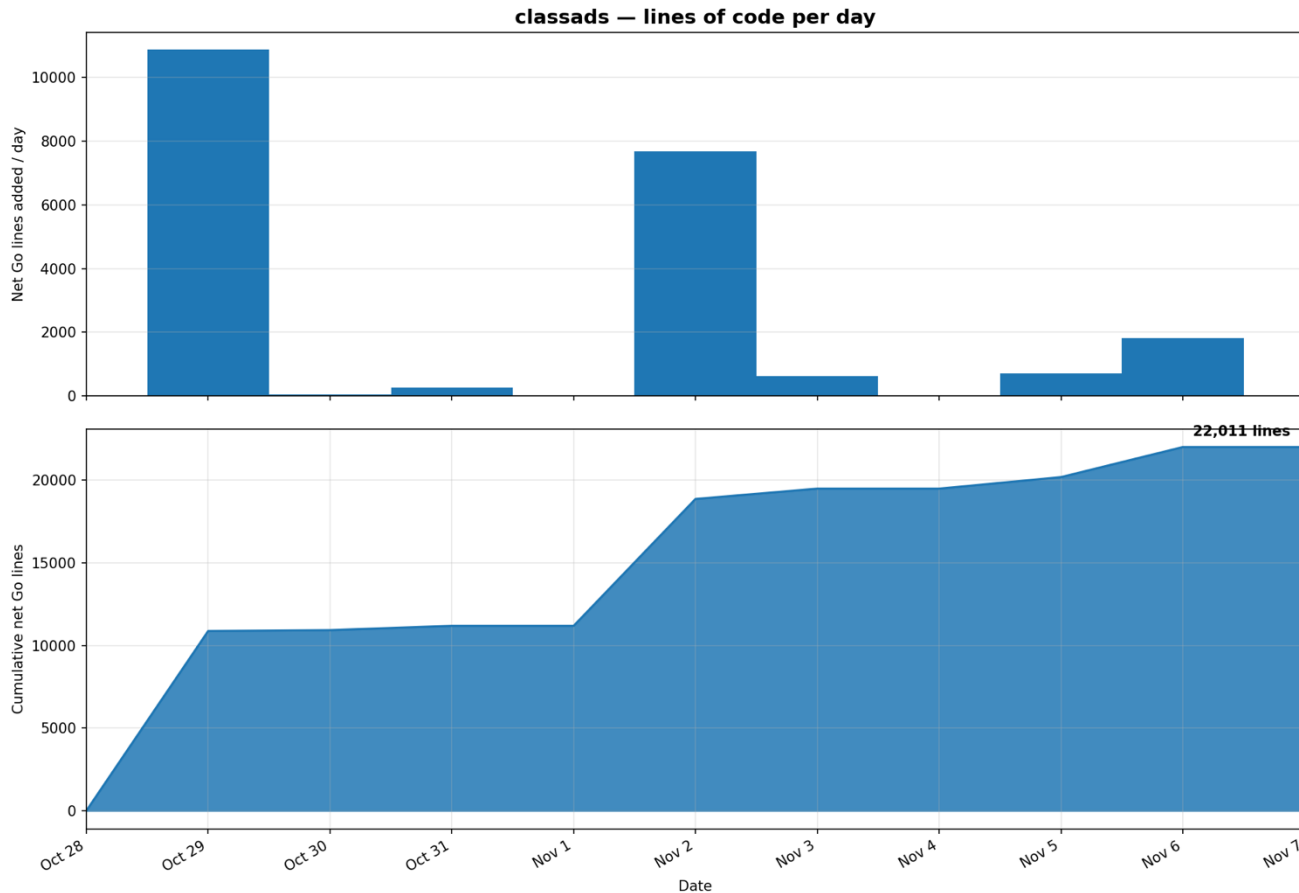
In a distributively owned environment, the owner of a resource has the right to define its usage policy, which may be very sophisticated. For example, the policy may state that a job can run on a workstation only if it belongs to a particular research group, or if it is run between 6 p.m. and 6 a.m., or if the keyboard hasn't been touched for over fifteen minutes and the load average is less than 0.1. Distributed ownership makes it impossible to formulate a monolithic system model. There is therefore a need for a resource management paradigm that does not require such a model and that can operate in an environment where resource owners and customers dynamically define their own models. The Matchmaking resource management paradigm presented in this paper was designed to address this need.

Matchmaking uses a semi-structured data model—the *classified advertisements* data model—to represent the principals of the system and folds the query language into the data model, allowing entities to publish queries (i.e., requirements) as attributes. The paradigm also distinguishes between matching and claiming as two distinct operations

Software is Knowledge; Agents can use Knowledge

- ▶ Take **ClassAds**: the Pelican plugin reads a list of input files in a ClassAd, performs downloads, writes a summary in a ClassAd. In October 2025, this was a hellscape of regular expressions as a “parser”.
- ▶ However, we have:
 - ▶ The original [ClassAd paper](#) PDF describing the concept.
 - ▶ The YACC file describing the ClassAd grammar.
 - ▶ A well-architected C++ library implementing ClassAds.
- ▶ **Idea:**
 - ▶ **“Dear ChatGPT”, can you take these inputs and turn them into a Go ClassAd library?**
- ▶ ...And thus <https://github.com/PelicanPlatform/classad> was born!

A Week of Rapid Progress

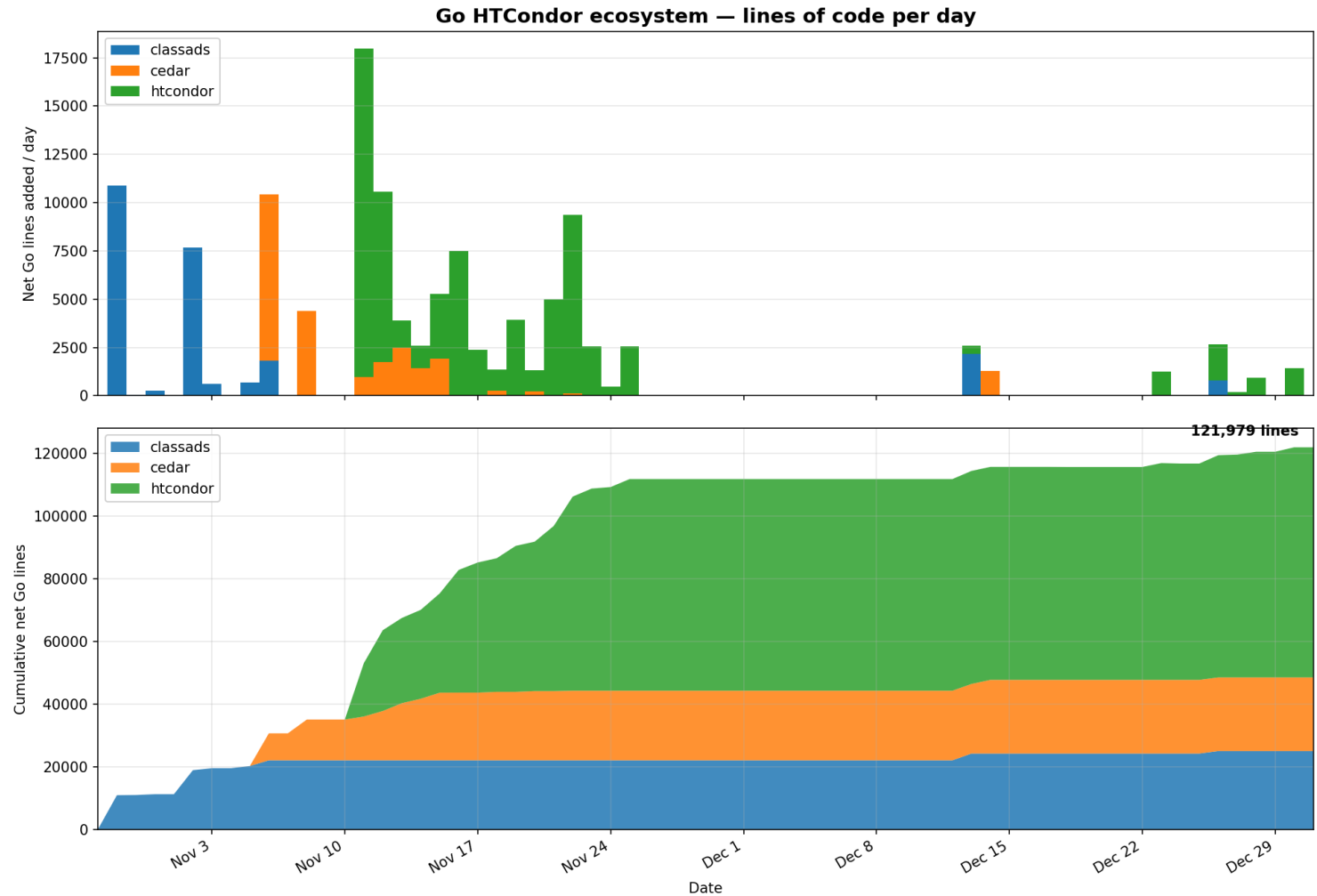


Another key ingredient: a HTCondor expert with an architectural vision and oversee the implementation (keep the agent off false paths).

- ▶ The agent was able to put together the core parser and evaluation engine in a single day, including unit tests.
- ▶ **Conundrum:** We implemented a ClassAd library but is it compatible with the ClassAd language?
- ▶ **Two additional sources:**
 - ▶ Online documentation of examples and behavior.
 - ▶ The Python bindings provide an “oracle”.
- ▶ With the bindings, the agent could determine if its own code was right:
 - ▶ Develop a change, test it against the reference implementation, and **repeat until correct.**
- ▶ The agent could run its own autonomous loop – an accelerant.

Why Stop There?

- ▶ ClassAds is not unique here: we can set up an autonomous loop with a live HTCondor system.
- ▶ While ClassAds is well-documented, HTCondor's communication protocol (CEDAR) is custom, binary, and entirely undocumented. The APIs between clients is undocumented.
 - ▶ There's a single implementation; a second had never been attempted.
- ▶ ... so we attempted it!

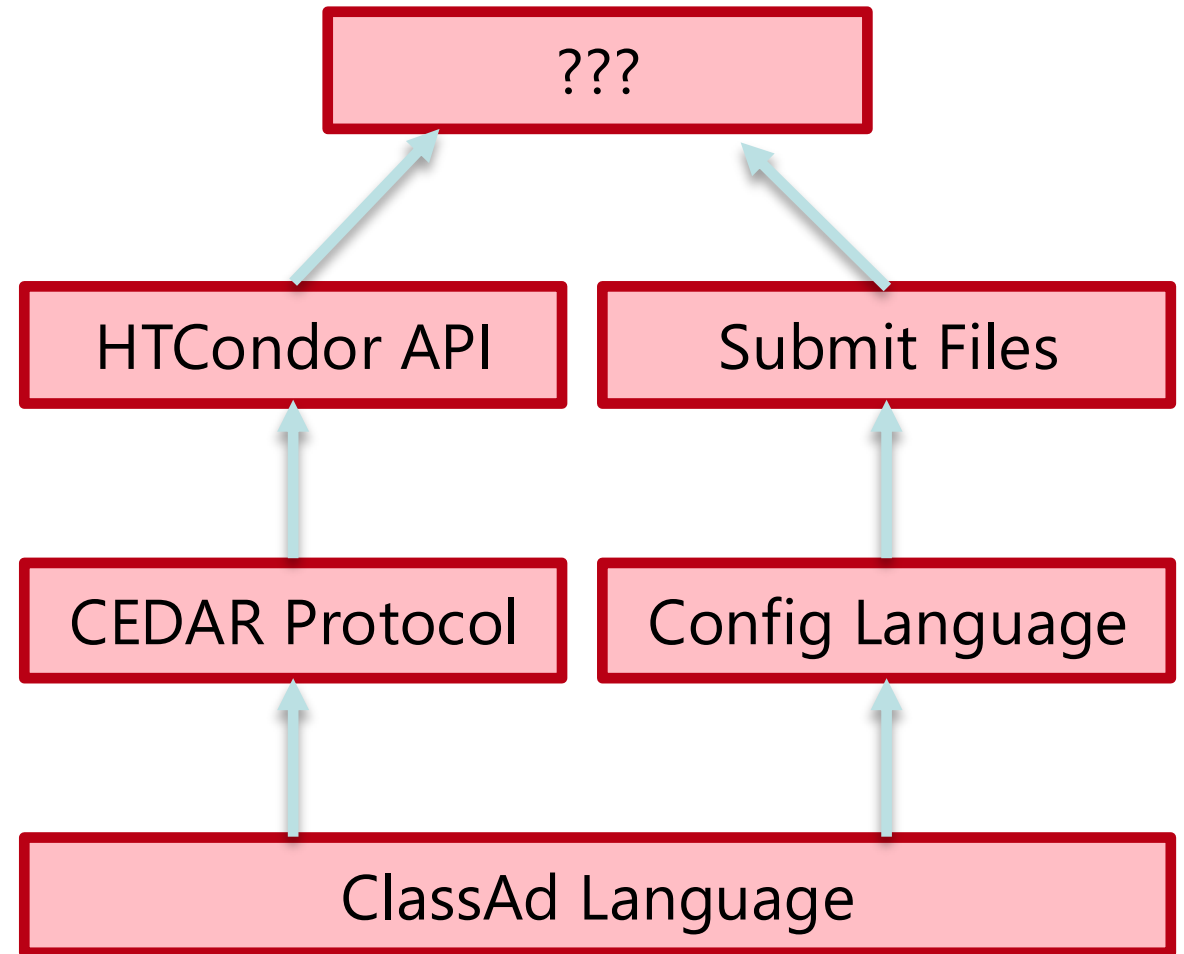


For Comparison – Python binding rewrite

- ▶ We knew HTCondor needed to drop its “boost.python” dependency.
 - ▶ 20 February 2023: Decision was made 20 February 2023 to rewrite the Python bindings.
 - ▶ 11 July 2024: Presented preview version to community at HTC’24
 - ▶ 31 October 2024: htcondor2 available in the stable series (24.0.0).
 - ▶ 25 July 2025: Last bits of rewritten functionality (HTCondor Ping) merged.
- ▶ ~500 developer hours logged.
- ▶ Python bindings interface with existing C++ library; Go libraries handle from the TCP socket on up.
- ▶ Completely fair comparison? Likely not...
 - ▶ Indicative of what can be accomplished? Sure!

Aside – the HTCondor “Tech Stack”

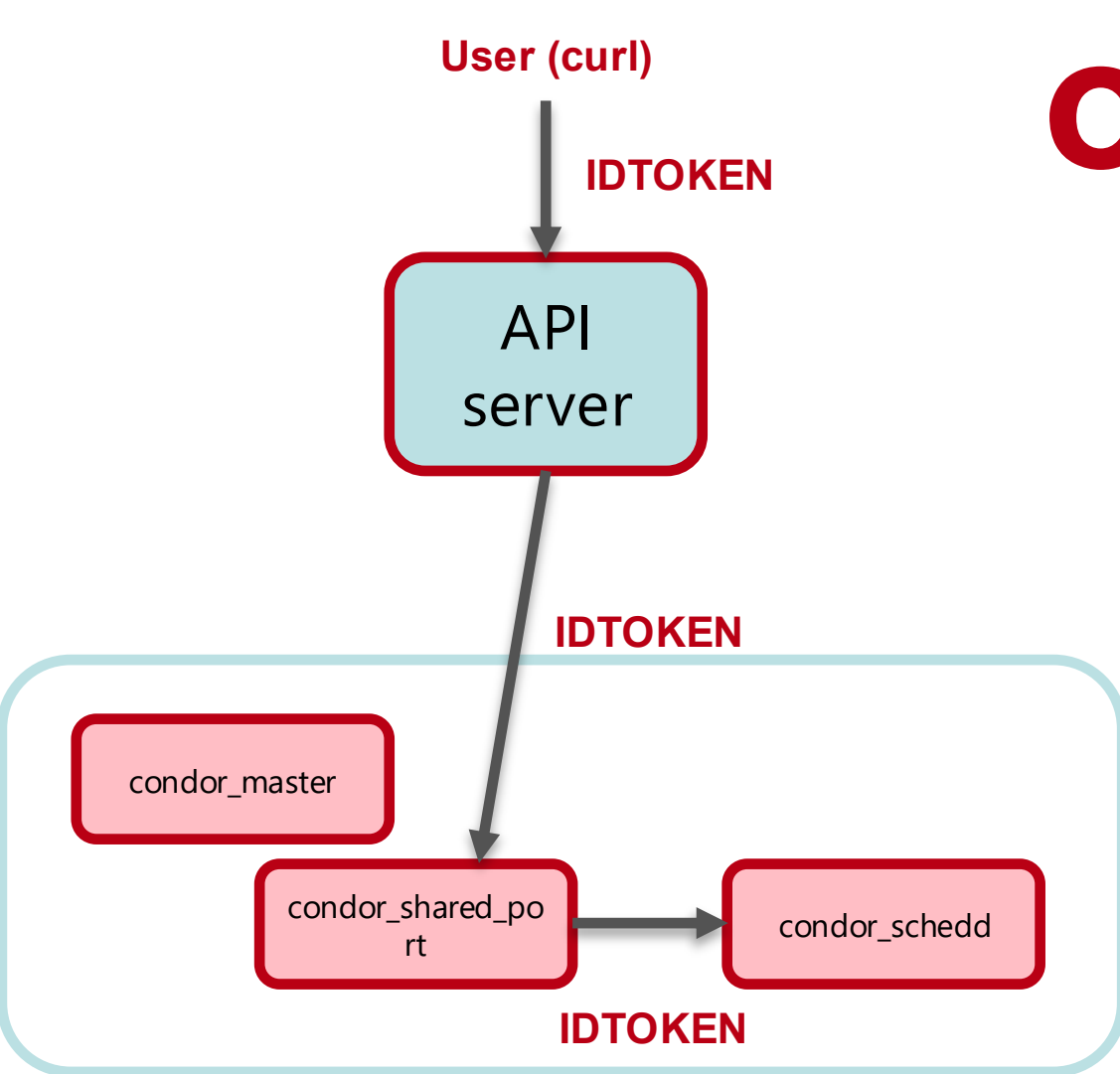
- ▶ **ClassAd language:** Core serialization format and matchmaking evaluation engine.
- ▶ **CEDAR protocol:** Proprietary binary protocol used by HTCondor, including authentication mechanisms.
- ▶ **Config language:** Serialization format used by HTCondor configuration files, resulting in a key-value (untyped) hash table.
- ▶ **Submit files:** Same as configuration language; includes translation of hash table to ClassAd-based job description.
- ▶ **HTCondor API:** Remote procedure calls (RPC) exposed by various daemons (e.g., job submit, status, remove, edit; daemon status queries).



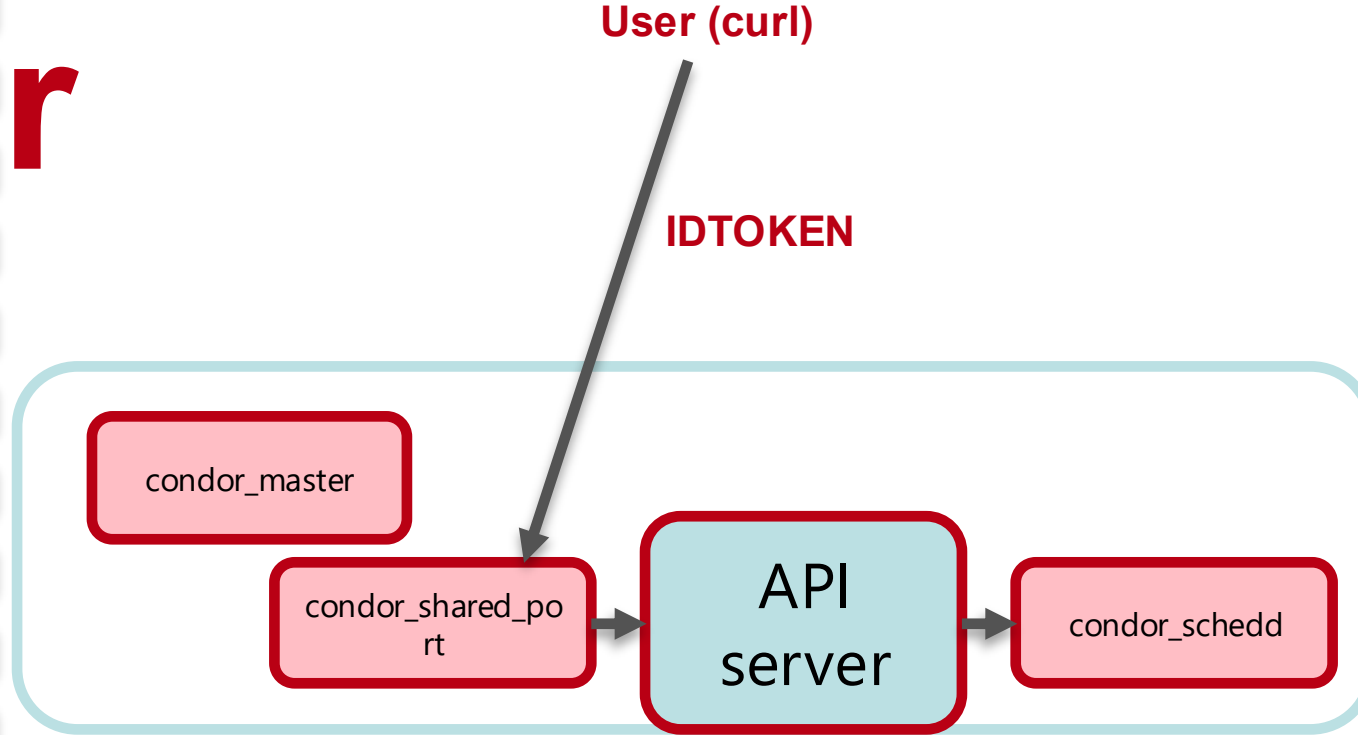
Toward what end?

- ▶ I assume a library with no users is a broken library. What should we use this for?
- ▶ **Idea:** Let's build out a REST API server using the Go libraries!
 - ▶ Go is great at web services; Go is great for high-concurrency applications.
 - ▶ A thorough REST interface would open HTCondor up to more programming languages.
 - ▶ And advance usage of programmatic interfaces!
- ▶ What are other interfaces that HTCondor is lacking?

Example – REST API



or



The API server can run on a standalone host or as part of the HTCondor installation using the same port as the rest of the system (9618).

By default It passes through the user's token, letting the condor daemon handle the authentication/authorization

REST API

The screenshot shows a web browser window displaying the HTCondor API Documentation. The browser's address bar shows the URL `ap40-api.osgdev.chtc.io/docs`. The page lists various REST endpoints with their corresponding HTTP methods and descriptions:

- POST** `/jobs` Submit a job
- GET** `/jobs/archive` Query job archive
- GET** `/jobs/epochs` Query job epoch history
- POST** `/jobs/hold` Hold jobs by constraint
- POST** `/jobs/release` Release jobs by constraint
- GET** `/jobs/transfers` Query transfer history
- DELETE** `/jobs/{jobId}` Remove a job
- GET** `/jobs/{jobId}` Get job details
- PATCH** `/jobs/{jobId}` Edit a job
- GET** `/jobs/{jobId}/files/{filename}` Download a specific file from job output sandbox
- POST** `/jobs/{jobId}/hold` Hold a job
- PUT** `/jobs/{jobId}/input` Upload job input files
- POST** `/jobs/{jobId}/input/multipart` Upload job input files via multipart form-data
- GET** `/jobs/{jobId}/output` Download job output files

- ▶ So, what's supported?
- ▶ The typical CRUD: create, read/list, edit, and delete jobs.
- ▶ More HTCondor-specific operations: hold/release jobs, query the job archive for job/transfer/epoch history.
- ▶ Data movement:
 - ▶ POST job input as a tarball or a multi-file upload.
 - ▶ GET job results as a tarball. GET stdout/err.
 - ▶ Tail stdout/err of a running job.
- ▶ Connect to a job via SSH (requires client to perform a WebSocket upgrade).

REST API - Example

Curl

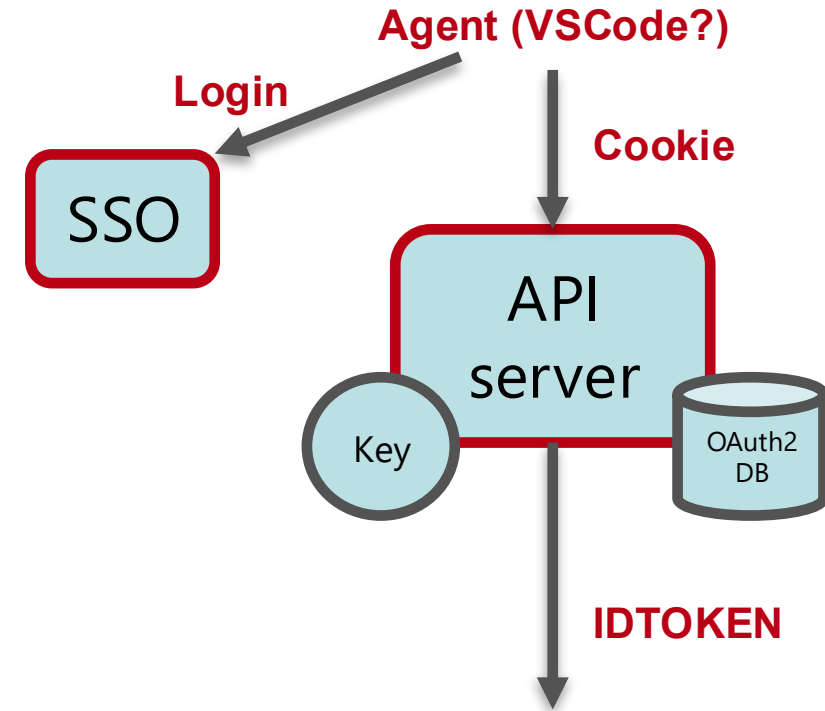
```
curl -X 'POST' \  
  'https://localhost:8080/api/v1/jobs' \  
  -H 'accept: application/json' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "submit_file": "executable = /bin/sleep\narguments = 30\noutput = sleep.out  
  }'
```



```
{  
  "cluster_id": 0,  
  "job_ids": [  
    "string"  
  ]  
}
```

From REST to AI Agents

- ▶ REST is wonderful – but it’s not the “native language” of AI agents.
- ▶ Nearly all agents speak “Model Context Protocol” (MCP).
 - ▶ MCP allows for structured or unstructured data.
 - ▶ More like streaming of data than an RPC.
 - ▶ MCP can be “spoken” via stdin/out (launch a process) or over HTTP
 - ▶ If HTTP is used, then endpoints are public or OAuth2-authenticated
- ▶ MCP standardizes introspection – what “tools” exist? What resources are available?
- ▶ Can add instructions or context to a response – hint the agent on what to do next or how to use the tools.



For the MCP, we must maintain a full OAuth2 identity provider; VSCode (or claude or codex...) is an OAuth2 client and will need to authenticate on behalf of the user & perform refresh flows.

Model Context Protocol - MCP

- ▶ The MCP is implemented in the same API server; simply provide the endpoint URL and finish the auth.
- ▶ Subsequently, you can utilize it from your AI agent web interface, coding agent, etc.

History and archives

- Query archived records of completed jobs.
- Query job epoch history (job restarts and execution attempts).
- Query file transfer history records.

Diagnostics and matchmaking

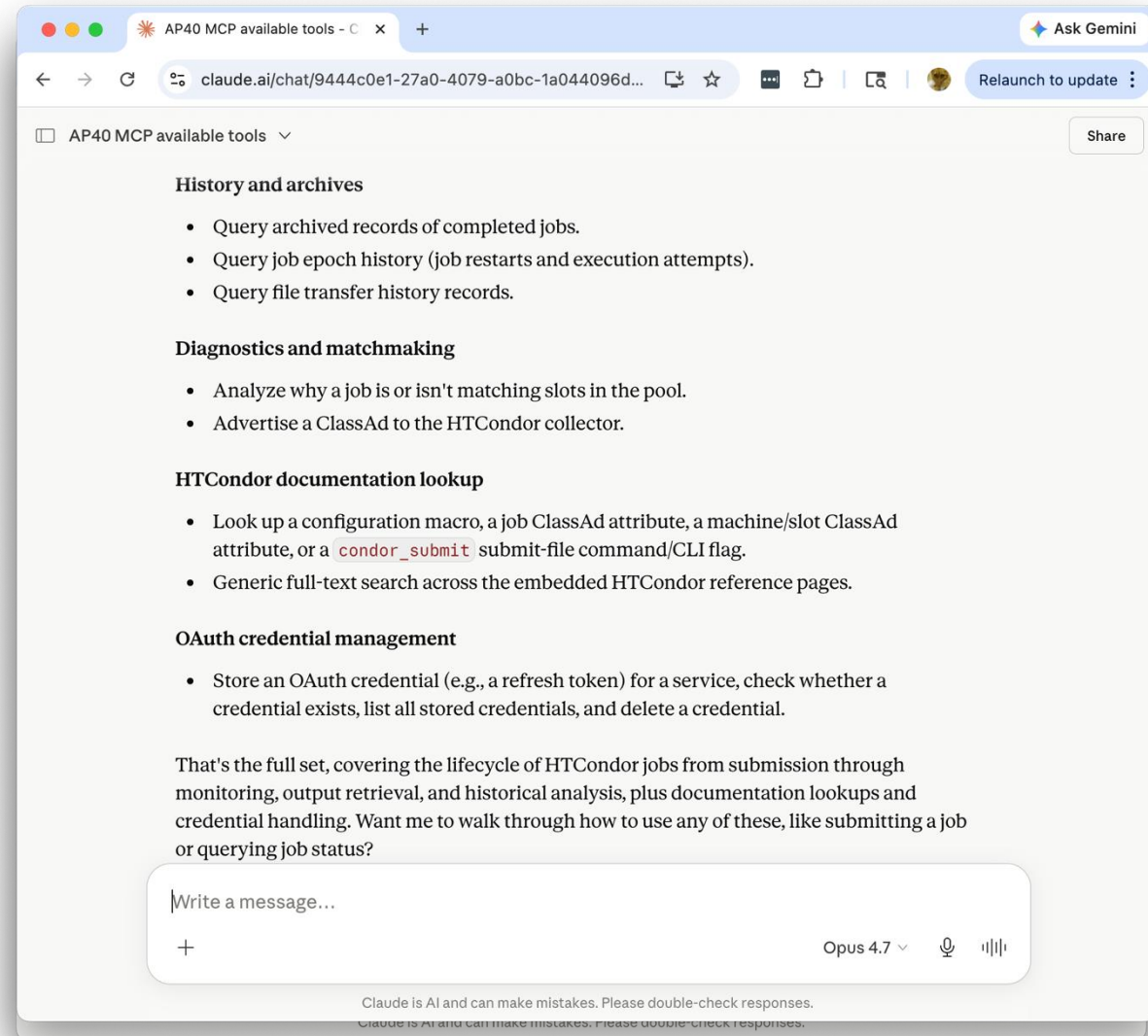
- Analyze why a job is or isn't matching slots in the pool.
- Advertise a ClassAd to the HTCondor collector.

HTCondor documentation lookup

- Look up a configuration macro, a job ClassAd attribute, a machine/slot ClassAd attribute, or a `condor_submit` submit-file command/CLI flag.
- Generic full-text search across the embedded HTCondor reference pages.

OAuth credential management

- Store an OAuth credential (e.g., a refresh token) for a service, check whether a credential exists, list all stored credentials, and delete a credential.



AP40 MCP available tools - C x + Ask Gemini

claude.ai/chat/9444c0e1-27a0-4079-a0bc-1a044096d... Relaunch to update

AP40 MCP available tools

Share

History and archives

- Query archived records of completed jobs.
- Query job epoch history (job restarts and execution attempts).
- Query file transfer history records.

Diagnostics and matchmaking

- Analyze why a job is or isn't matching slots in the pool.
- Advertise a ClassAd to the HTCondor collector.

HTCondor documentation lookup

- Look up a configuration macro, a job ClassAd attribute, a machine/slot ClassAd attribute, or a `condor_submit` submit-file command/CLI flag.
- Generic full-text search across the embedded HTCondor reference pages.

OAuth credential management

- Store an OAuth credential (e.g., a refresh token) for a service, check whether a credential exists, list all stored credentials, and delete a credential.

That's the full set, covering the lifecycle of HTCondor jobs from submission through monitoring, output retrieval, and historical analysis, plus documentation lookups and credential handling. Want me to walk through how to use any of these, like submitting a job or querying job status?

Write a message...

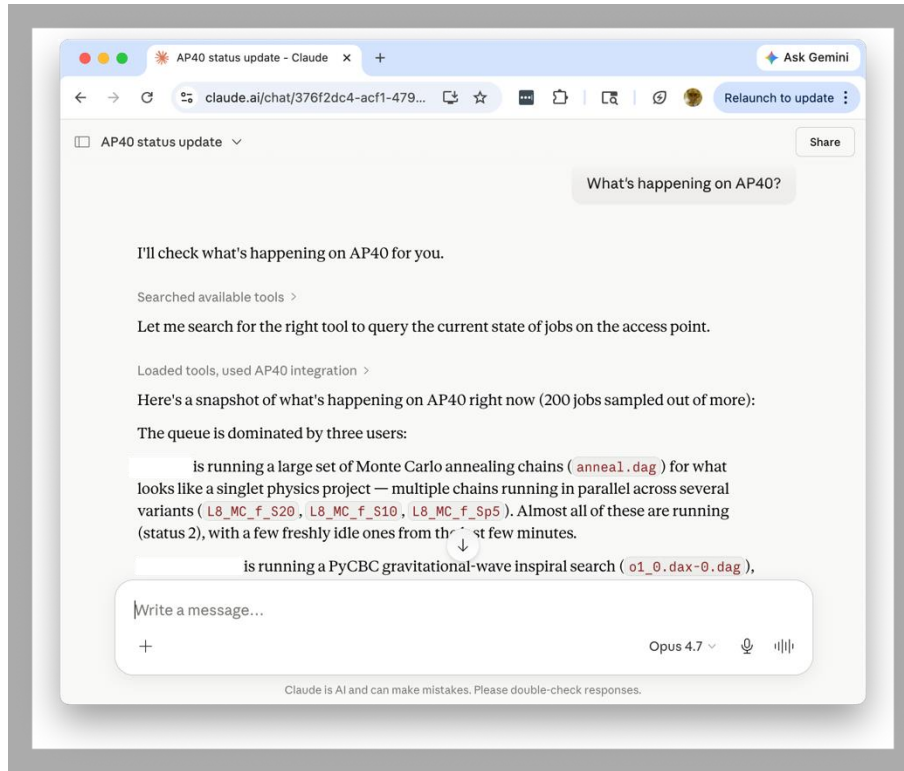
+ Opus 4.7

Claude is AI and can make mistakes. Please double-check responses.

HTCondor MCP -

- ▶ As we've seen, LLMs combined with tools can do remarkable things starting with just natural language prompts.

Claude Browser Interface



VSCode Interface

> Used 6 references

> Ran `submit_job` - api40-api-v1 (MCP Server) ✓

Auto approved for this session ([Edit](#))

Success! Job cluster 12714297 was submitted. The issue was the `log` attribute in the submit file. The schedd was rejecting jobs that specified a log file path, likely due to security policies that prevent writing to arbitrary paths on the submit host.

👍 👎 Claude Sonnet 4.5 • 1x

Check the status of job 12714297.0

> Used 6 references

> Ran `get_job` - api40-api-v1 (MCP Server) ✓

Auto approved for this session ([Edit](#))

Job 12714297.0 is currently **on hold** (JobStatus=5) with the reason "Spooling input data files" (HoldReasonCode=16).

Key details:

- **Owner:** brian.bockelman.1
- **AccountingGroup:** group_opportunistic.OSG-Staff.brian.bockelman.1
- **Status:** Held (5) - temporary hold for spooling
- **Executable:** `/bin/echo` with argument "Hello from HTCondor test"
- **Resources:** 1 CPU, 128MB memory, 1MB disk

The job submission was successful!



From REST API -> Web Interface

- ▶ Given a REST API, one can start building out a browser-based interface, making the system more accessible to less-experienced users.
 - ▶ For HTCondor, we focus on mechanisms for placing & managing workloads, not individual jobs.
- ▶ Web UIs can also provide remarkable density of information compared to the CLI

1. Template
Define the submit-file body and which columns the table will fill in.

[Use template from library](#) [Write new template](#)

Template

Hello World BUILT-IN

Hello World BUILT-IN

One echo per row, with \$(name) substituted in.
Columns: name

[Show submit-file body](#)

2. Table
Column headers come from the template. 1 row = 1 job.

[Manual table](#) [Run N copies](#) [From directory / tarball](#)

Or import: [Paste CSV](#) [Upload CSV](#)

#	NAME
1	

[+ Add row](#) [x Duplicate all](#)

3. Common Inputs
Files attached to every job in the batch. (Per-job files come from step 2's upload mode if you used it.)

Drag & drop input files here, or click to browse.
Files will be spooled to the schedd and made available to your job.

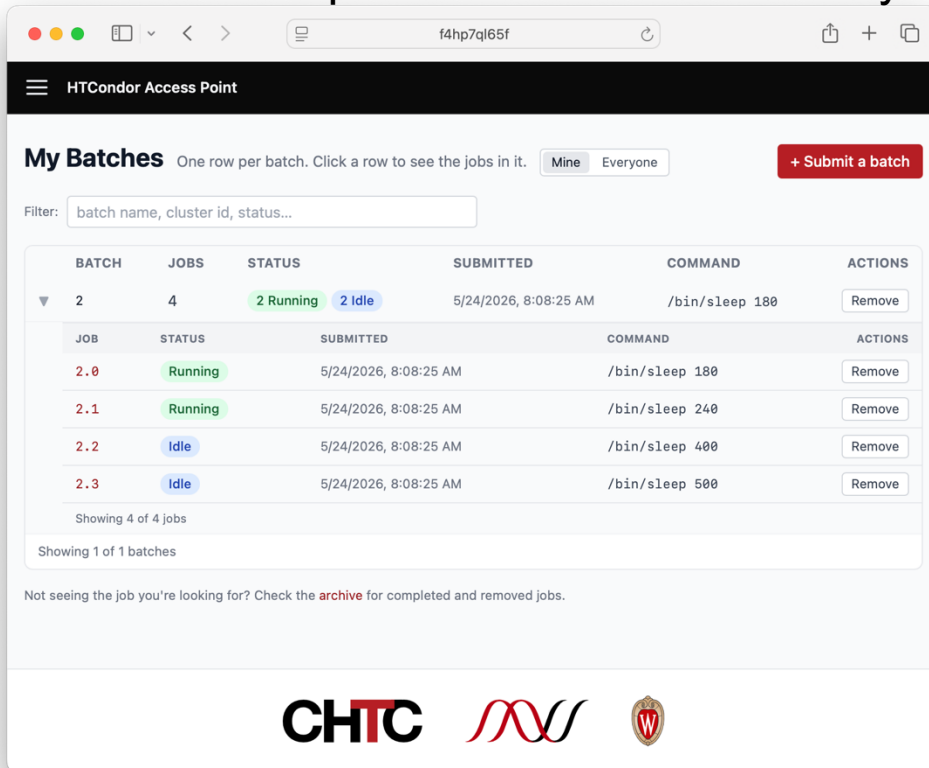
HTCondor Access Point

← Interactive Terminal 1.0 Ready [End session](#)

```
Welcome to slot1_1@F4HP7QL65F!  
Your condor job is running with pid(s) 77401.  
F4HP7QL65F:scratch bbockel$ ls  
interactive-watchdog.sh interactive.err interactive.out  
F4HP7QL65F:scratch bbockel$ tail interactive.out  
F4HP7QL65F:scratch bbockel$ tail interactive.err  
[interactive-watchdog] starting pid=77401 scratch=/var/folders/3r/x119t4ks0wsd5httljvnm86r000gp/T/htc  
ondor-demo-3579907041/execute/dir_77347/scratch poll=30s freshness=120s  
F4HP7QL65F:scratch bbockel$
```

From REST API -> Web Interface

- ▶ Given a REST API, one can start building out a browser-based interface, making the system more accessible to less-experienced users.
 - ▶ For HTCondor, we focus on mechanisms for placing & managing workloads, not individual jobs.
- ▶ Web UIs can also provide remarkable density of information compared to the CLI



HTCondor Access Point

My Batches One row per batch. Click a row to see the jobs in it. Mine Everyone + Submit a batch



Filter:

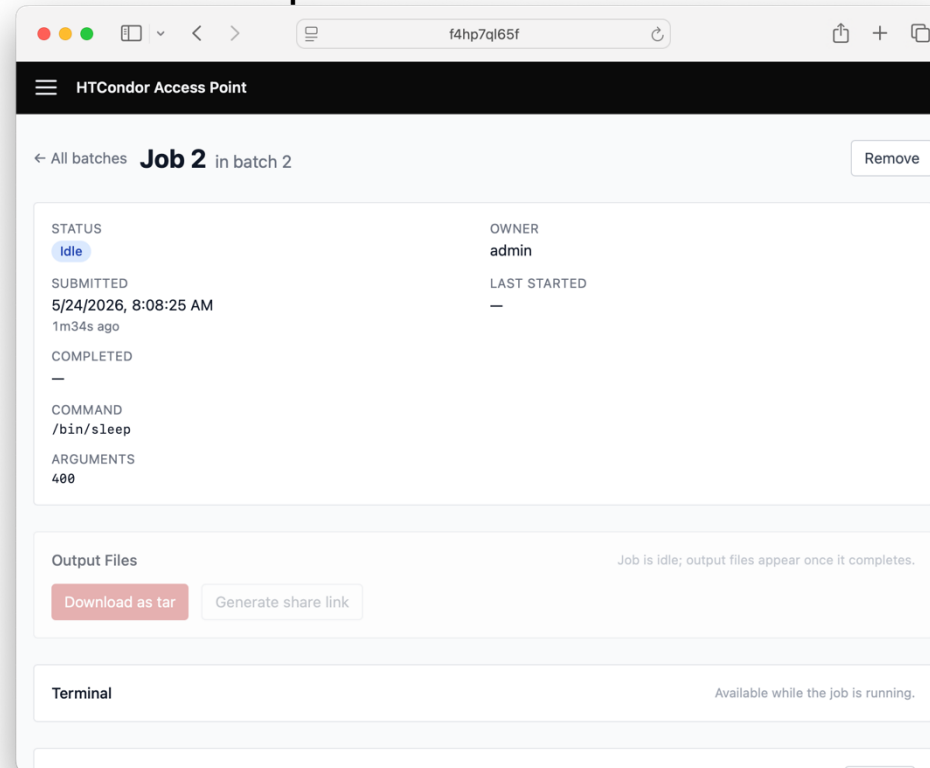
BATCH	JOBS	STATUS	SUBMITTED	COMMAND	ACTIONS
▼ 2	4	2 Running 2 Idle	5/24/2026, 8:08:25 AM	/bin/sleep 180	Remove
JOB	STATUS	SUBMITTED	COMMAND	ACTIONS	
2.0	Running	5/24/2026, 8:08:25 AM	/bin/sleep 180	Remove	
2.1	Running	5/24/2026, 8:08:25 AM	/bin/sleep 240	Remove	
2.2	Idle	5/24/2026, 8:08:25 AM	/bin/sleep 400	Remove	
2.3	Idle	5/24/2026, 8:08:25 AM	/bin/sleep 500	Remove	

Showing 4 of 4 jobs

Showing 1 of 1 batches

Not seeing the job you're looking for? Check the [archive](#) for completed and removed jobs.

CHTC  



HTCondor Access Point

← All batches **Job 2** in batch 2 Remove

STATUS Idle OWNER admin

SUBMITTED 5/24/2026, 8:08:25 AM LAST STARTED 1m34s ago

COMPLETED —

COMMAND /bin/sleep

ARGUMENTS 400

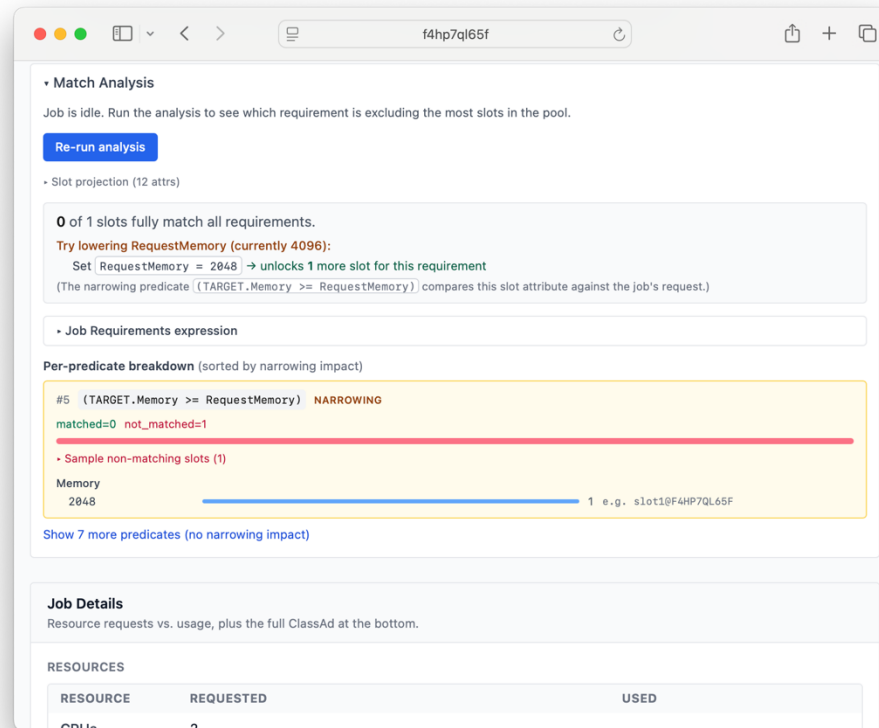
Output Files Job is idle; output files appear once it completes.

Download as tar Generate share link

Terminal Available while the job is running.

From REST API -> Web Interface

- ▶ Given a REST API, one can start building out a browser-based interface, making the system more accessible to less-experienced users.
 - ▶ For HTCondor, we focus on mechanisms for placing & managing workloads, not individual jobs.
- ▶ Web UIs can also provide remarkable density of information compared to the CLI



From Web Interface to Agents

- ▶ A tradeoff in web UIs is simplicity versus complexity: add enough functionality to make it useful for experts and you overwhelm the beginners.
 - ▶ What if we allow agents to help out in the web interface?

Submit a Batch

Pick or write a template, fill the table with one row per job, attach any input files, then submit.

Submit assistant

Create two sleep jobs, for 5 and 10 minutes, respectively.

I'll create two sleep jobs for you. Let me first check if there's a suitable template available, then set up your jobs.

`list_submit_templates` ✓

Perfect! There's already a "Sleep" template designed exactly for this. Let me activate it and configure your two jobs.

`select_template` ✓ `set_table_rows` ✓

Ask the assistant... Send

The assistant only sees your own jobs.

1. Template

Define the submit-file body and which columns the table will fill in.

Use template from library Write new template

Template

Sleep

← All batches **Job 0** in batch 2

Job assistant

I'll help you investigate why job 2.0 is idle. Let me check the job's event log and what's preventing it from being scheduled.

`get_job_log` ✓ `get_match_analysis` ✓

Your job is idle because it's requesting too much memory.

The match analysis shows that **all 3 slots in the pool fail** the memory requirement:

- **Your job requests:** 10,240 MB (10 GB) of RAM
- **Available memory in the pool:**
 - 1 slot with 1,792 MB (~1.75 GB)
 - 2 slots with 128 MB each

No slot has enough memory to run your job.

To fix this:

Ask the assistant... Send

The assistant only sees your own jobs.

HTCondor Access Point

My Batches

One row per batch. Click a row to see the jobs in it. + Submit a batch

Filter: batch name, cluster id, status...

BATCH	JOBS	STATUS	SUBMITTED	COMMAND	ACTIONS
1	2	2 Running	5/9/2026, 10:41:31 AM	/bin/sleep 300	Remove

JOB	STATUS	SUBMITTED	COMMAND	ACTIONS
1.0	Running	5/9/2026, 10:41:31 AM	/bin/sleep 300	Remove
1.1	Running	5/9/2026, 10:41:31 AM	/bin/sleep 600	Remove

Job assistant

Auto-approve: hold release remove

What's job 1.1 doing?

`query_jobs` ✓

Job 1.1 is running /bin/sleep 600 — sleeping for 600 seconds (10 minutes). It started just after it was queued.

`highlight_job` ✓

Ask the assistant... Send

The assistant only sees your own jobs.

CHTC

So what's the bad news?



Tim Theisen 6:48 AM

🔥 ap2001 schedd down since 11:30 last night. Signal 6 when trying to start local job, Trying to run job, but it's already

```
05/29/26 06:26:37.532 (pid:3287238) (D_COMMAND) Return from Handler <DaemonCore::HandleReqPayloadReady>
05/29/26 06:26:37.532 (pid:3287238) (D_ALWAYS:2) Job prep for 6986725.0 will not block, calling aboutToS
05/29/26 06:26:37.532 (pid:3287238) (D_ALWAYS:2) aboutToSpawnJobHandler() completed for job 6986725.0, a
05/29/26 06:26:37.532 (pid:3287238) (D_ALWAYS:2) Starting local universe job 6986725.0
05/29/26 06:26:37.532 (pid:3287238) (D_ALWAYS:2) About to spawn /usr/sbin/condor_starter condor_starter
0 -header (6986725.0) ' ' -job-input-ad - --schedd-addr <128.105.68.112:9662?alias=ap2001.chtc.wisc.edu>
05/29/26 06:26:37.532 (pid:3287238) (D_ERROR) ERROR "Trying to run job 6986725.0, but already marked RUN
/var/lib/condor/execute/slot2/dir_1509572/scratch/userdir/build-fw8jqA/BUILD/condor-25.11.0/src/condor_s
Caught signal 6: si_code=4294967290, si_pid=3287238, si_uid=0, si_addr=0x3228C6
Stack dump for process 3287238 at timestamp 1780053997 (18 frames)
/lib64/libcondor_utils_25_11_0.so(_Z18dprintf_dump_stackv+0x28)[0x7f591ac68fd8]
/lib64/libcondor_utils_25_11_0.so(_Z17unix_sig_coredumpiP9siginfo_tPv+0x6f)[0x7f591ae23a3f]
/lib64/libc.so.6(+0x3fc60)[0x7f591a23fc60]
/lib64/libc.so.6(+0x8d21c)[0x7f591a28d21c]
/lib64/libc.so.6(raise+0x16)[0x7f591a23fbb6]
/lib64/libc.so.6(abort+0xd3)[0x7f591a229873]
/lib64/libcondor_utils_25_11_0.so(+0x123063)[0x7f591ab23063]
condor_schedd(_Z16mark_job_runningRK7PROC_ID+0x3f8)[0x555ac9661868]
condor_schedd(_ZN9Scheduler17spawnLocalStarterEP10shadow_rec+0x2a7)[0x555ac96655b7]
condor_schedd(_ZN9Scheduler15spawnJobHandlerEiiP10shadow_rec+0x1d8)[0x555ac9646fb8]
condor_schedd(_ZZ26aboutToSpawnJobHandlerDoneiiPvi+0x73)[0x555ac96470f3]
condor_schedd(_ZN9Scheduler15StartJobHandlerEi+0xef)[0x555ac9657adf]
/lib64/libcondor_utils_25_11_0.so(_ZN12TimerManager7TimeoutEPiPd+0xef)[0x7f591ae3c9af]
/lib64/libcondor_utils_25_11_0.so(_ZN10DaemonCore6DriverEv+0x1b7)[0x7f591ae0e9b7]
/lib64/libcondor_utils_25_11_0.so(_Z7dc_mainiPPc+0x13d6)[0x7f591ae30fe6]
/lib64/libc.so.6(+0x2a610)[0x7f591a22a610]
```

Another example of overconfidence: In Pelican, we used the agent to overhaul / cleanup the internals of our configuration subsystem in December 2025. Caused numerous regressions: turns out our configuration system was under-specified and important behavior didn't have test coverage.

- ▶ We believe in “no free lunch”! What’s the bad news?
- ▶ LLM agents are an accelerant: they allow users to try things that are “expert level”.
 - ▶ ... and maybe things no human would try!
 - ▶ Example (left): Claude suggested to a user `condor_qedit` a job from “vanilla” to “local” universe” to assist in debugging (?).
 - ▶ Triggered a HTCondor bug that causes `schedd` to crash if you edit the job’s universe.
- ▶ LLM agents can provide a false sense of security: “The robot is smart and certainly wouldn’t do something bad”
 - ▶ End of day: you can’t delegate responsibility!

From “Party Trick” To “Science”

▶ **Coding Impact:**

- ▶ Clearly provide value: not “magic” but useful tools that, when applied well, allows CHTC to use resources more effectively.

▶ **User Impact:**

- ▶ We are seeing users with fewer technical skills more effectively using HTCondor. Broadens our science impact!
- ▶ Mixed bag: part of our goal was to teach skills! Temptation to “outsource your brain”

▶ **“AI Agent to do my Science”**

- ▶ Ok, an AI agent can submit and manage a job. That’s a “party trick” – amazing and surprising (at least from 2 years ago!).
 - ▶ Submitting a job at a time with a natural language prompt is no way to scale to a 10,000 job workflow.
- ▶ The agent needs to go beyond “hello world” and “sleep jobs”; it can create “science jobs” but do they do the right thing – or just look good?
 - ▶ Similarly, one needs to manage data inputs/outputs at scale; manage environments (containers).
- ▶ How far away are we from “doing real science”?

Only a year into the “agentic era” – things are getting interesting!

Questions?

This project is supported by the National Science Foundation under Cooperative Agreements PHY-2323298. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.