RESTful interfaces to HTCondor

Greg Thain

Agenda

- Previous mistakes
- The Religion of REST
- HTCondor design as applied to REST
- Comments and discussion.

Agenda

- Previous mistakes
- The Religion of REST
- HTCondor design as applied to REST
- Comments and discussion.

WARNING: Work in Progress!!!

In the beginning….

…There was the command line

```
$ condor_submit job.sub
$ condor_q
$ condor_rm
```

And it was good…

And it was good…

At least for users who were human

for some definition of “human”

And then came the machine interfaces

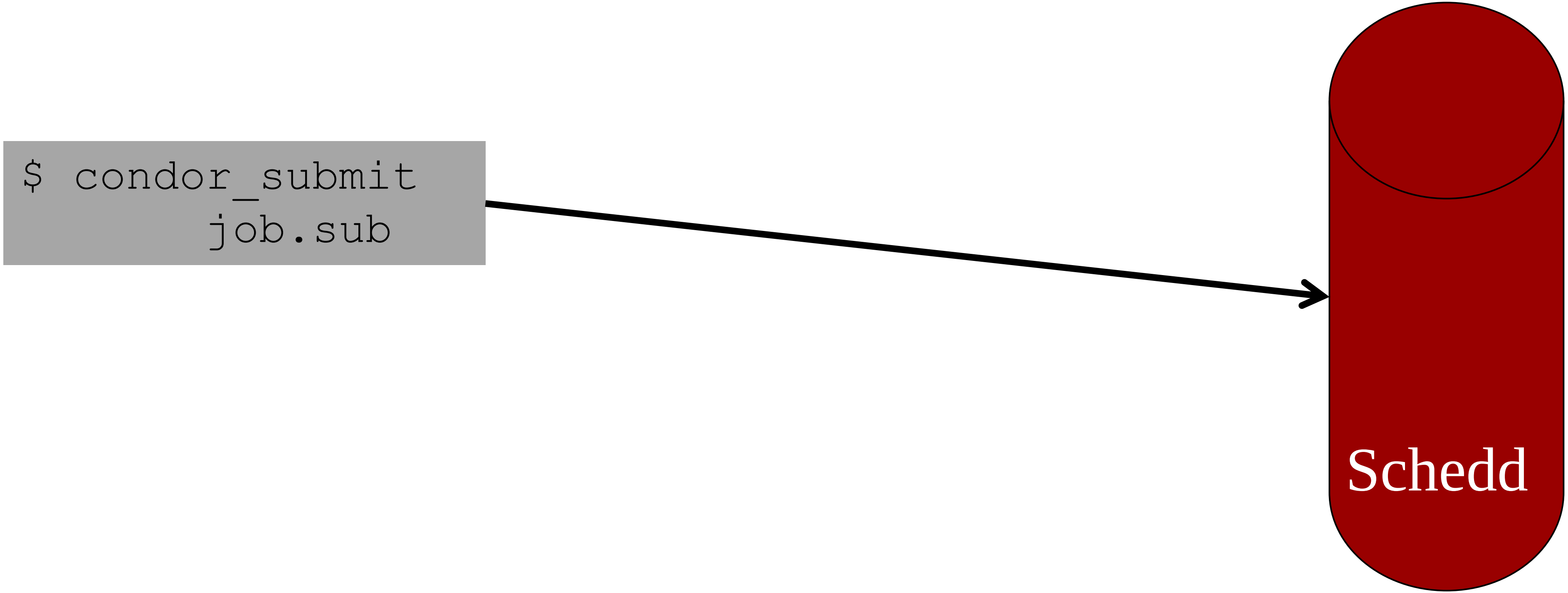
Machine interfaces

- Web server
- Science gateways
- Custom GUI interfaces
- CLIs are horrible for this
    - string parsing input and output clumsy
    - Error handling difficult
    - Fragile

Interfaces to do what?

What	CLI equivalent
Submit jobs	condor_submit
Remove Jobs	condor_rm
Query jobs	condor_q
Query Machines	condor_status
Query everything else	condor_status –something

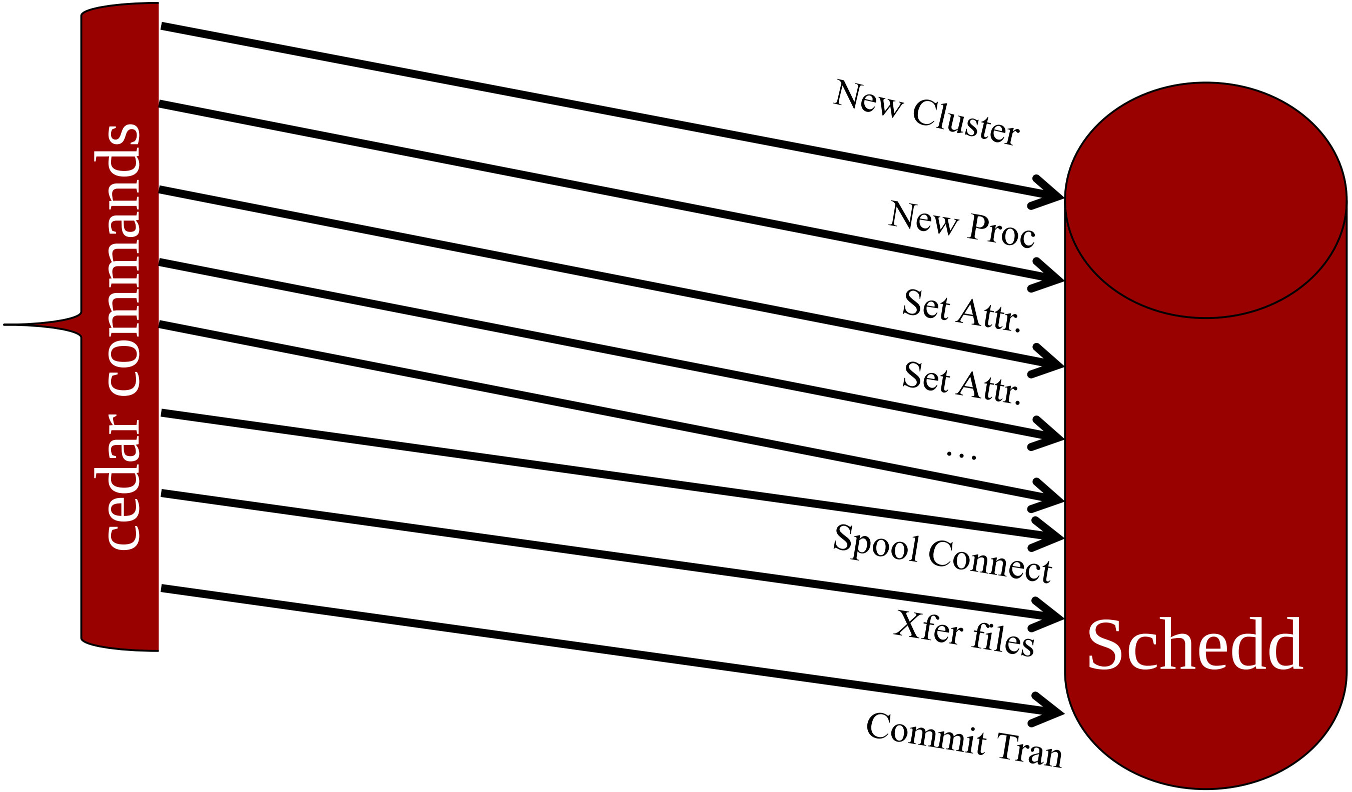
condor_submit user's view



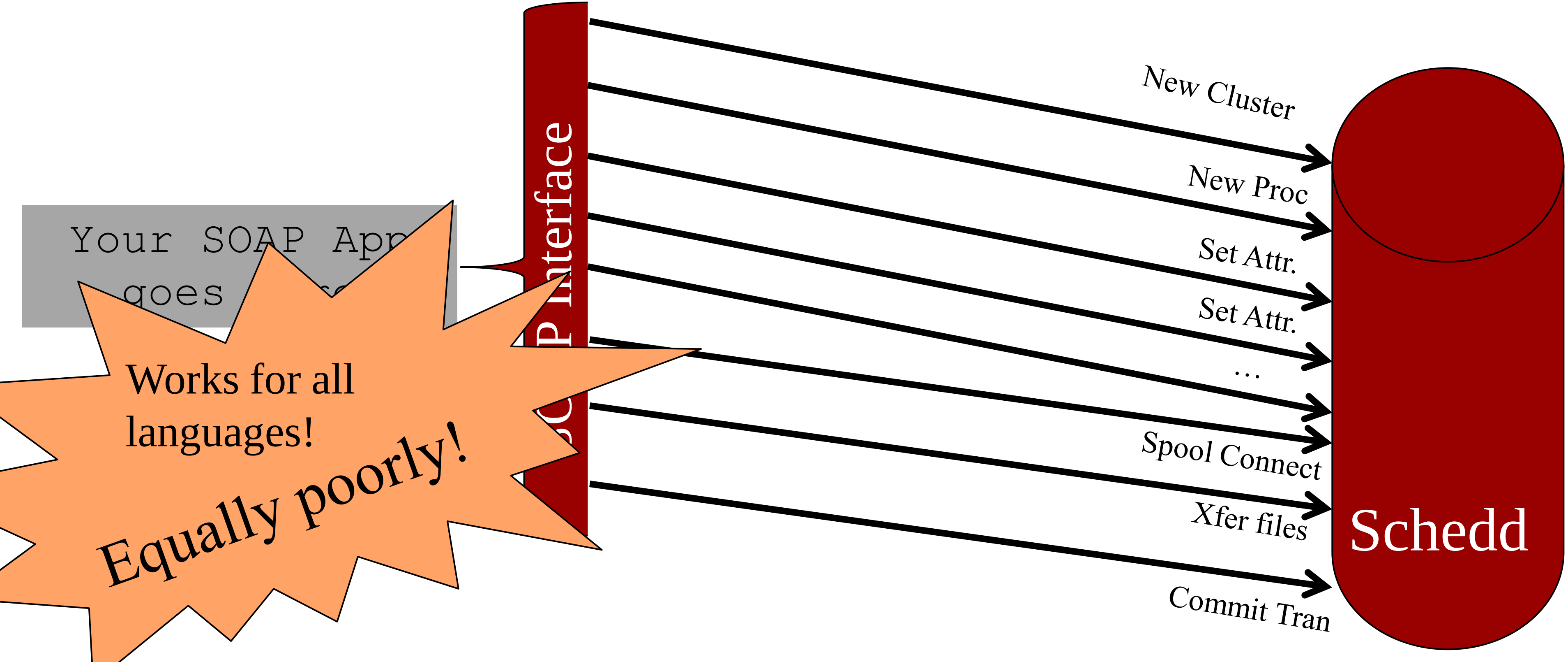
```
$ condor_submit
         job.sub
```

Behind the hood of submit

```
$ condor_submit
        job.sub
```



Our SOAPy mistake…



Works for all languages!

Equally poorly!

What are the right interfaces?

- What are the right “objects”?
- What are the right entities?
- What are the right methods on those entities?

- These are the questions we want to ask!
  - Not “how to expose HTCondor wire protocols”
  - Or what’s the quickest to implement

REST has some opinions!

- Small, fixed number of generic methods:
  - Like ER modeling, CRUD:
    - GET, PUT, POST, PATCH, HEAD, DELETE,…
- Rich set of *endpoints* with well defined IDs
- Description language open, usually json
- Implicit transactions
- Also, we’d like YOUR opinions

Example HTCondor endpoints

- /jobs
- /history
- /status
- /config

- These are nouns, not daemons or services

Just in case we get it wrong…

- /v1/jobs
- /v1/history
- /v1/status
- /v1/config

- Room for future changes!

Not every verb with all nouns…

- /v1/jobs
- /v1/history
- /v1/status
- /v1/config

GET
HEAD
POST
PATCH

Example endpoint queries

- GET /v1/jobs{/clusterid}{/procid}{/attribute}{?projection,constraint}
  - Constraint like condor_q –constraint
  - Projection like –af, only get these attrs
  - Returns json document

Query example

- GET
- /v1/jobs{/clusterid}{/procid}{/attribute}{?projection,constraint}
  - Constraint like condor_q –constraint
  - Projection like –af, only get these attrs
  - Returns json document

Query example

```
$ curl localhost:8888
/v1/jobs/clusterid/procid?projection =
“Owner,Requirements,Foo”
[
“jobid”: “123.34”
“Owner”: “gthain”
“foo”:  17
]
```

History endpoint

- Just like jobs, but hits condor_history jobs

```
$ curl localhost:8888
/v1/history/clusterid/procid?projection =
“Owner,Requirements,Foo”
[
“jobid”: “123.34”
“Owner”: “gthain”
“foo”:  17
]
```

“status” endpoint

- Really slots
- GET /v1/status{/name}{?projection,constraint}

status query

```
$ curl localhost:8888
/v1/status/name?projection =
“Memory,Start,Foo”
[
“Memory”: 8192
“Start”: “\/Expr ”
“foo”:  17
]
```

“config” endpoint

- Read only access to config
- GET /v1/status{/name}{?projection,constraint,query}

Try out prototype!

- https://github.com/htcondor/htcondor-restd

- REST is language agnostic, but
  - We have open-api spec to generate python,go etc

Future work

- Lot’s of it
  - Mutation, requiring authentication
  - Support for all REST verbs:
    - PUT, PATCH, DELETE

Even more future work…

- Design questions
  - One rest server per machine?
  - Jobs vs history?
  - Shared port
- Minimum set of verbs we need
  - What about condor_hold/condor_release

Conclusion