

CHEP 2013

Minimizing draining waste through extending the lifetime of pilot jobs in Grid environments

by

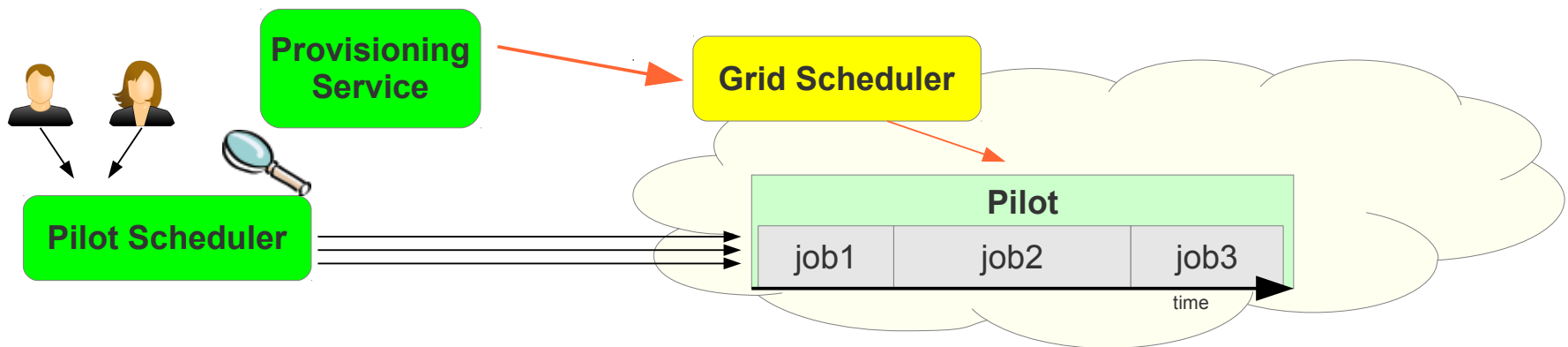
I. Sfiligoi¹, T. Martin¹, B. P. Bockelman², D. C. Bradley³,
F. Würthwein¹

¹University of California San Diego ²University of Nebraska-Lincoln

³University of Wisconsin-Madison

Where are we coming from?

- Grid users have embraced the Pilot model
 - Separates resource provisioning (via pilots) from user job scheduling
 - Pilot resources are temporary, but can execute several user jobs



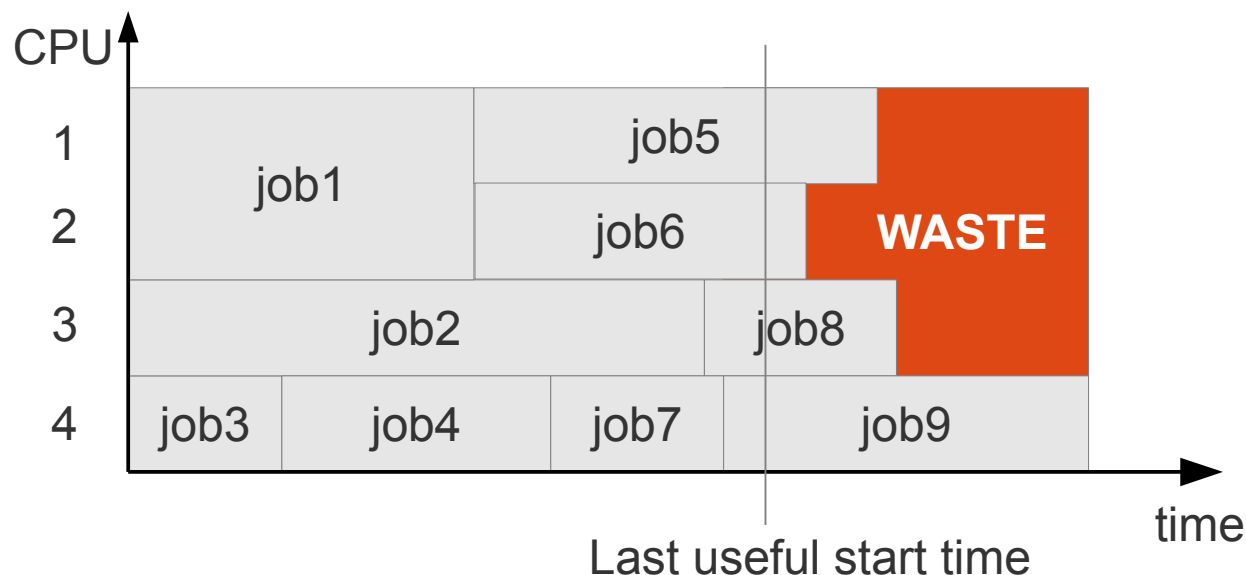
- Pilot overheads have by-and-large been small
 - At most minutes wasted for job fetching and cleanup

What is changing?

- A pilot has **traditionally managed a single CPU**
 - Which was assigned to a single user job at a time
- Several scientific communities now want more flexibility
 - A single job may need more than one CPU
 - But single-CPU jobs should not be forbidden
- As a consequence, pilots will be expected to **grab multiple-CPU's at once, and then partition** them among user jobs

Why is this a problem?

- Pilot **overhead will drastically increase**
 - Due to idling of part of the managed resources once it is not starting new jobs anymore
- The pilot has to wait for the longest user job
 - And user job lifetimes can vary a lot



We need long-lived pilots

- **Minimizing the absolute waste a hard problem**
 - Would need to know *at least* a very good estimate of the expected runtime of each user job
 - Which many studies have shown to be a hard problem
- **Keeping relative waste small requires just long-lived pilot jobs**
 - Conceptually, a very easy solution
- **In practice, most sites limit pilots to 24h-48h**
 - Which is still pretty short!

$$\text{rel_waste} = \text{abs_waste} / \text{pilot_runtime}$$

The problem of long runtimes ^{1/2}

- While long runtimes are great for pilots, **currently they are a real problem for the resource providers**
- In HTC we try to maximize resource utilization
(High Throughput Computing)
 - **If a user does not have enough jobs in the queue to fill his share of resources, jobs from other users are scheduled on them**
- But when that user submits a job he **expects it to start quickly**
 - But there are no idle resources at that point!
 - **The user's job has to wait for another job to finish**

The problem of long runtimes ^{2/2}

- With **short runtimes**, job turnover is high
 - Average turnover $\sim \text{num_cpus}/\text{avg_job_runtime}$

- However, **longer runtimes**

→ **lower job turnover**

→ **longer wait times for an available resource**

- Resource providers thus would have to choose between

- **Lowering quality of service**, or
- **Forceful termination of running jobs**

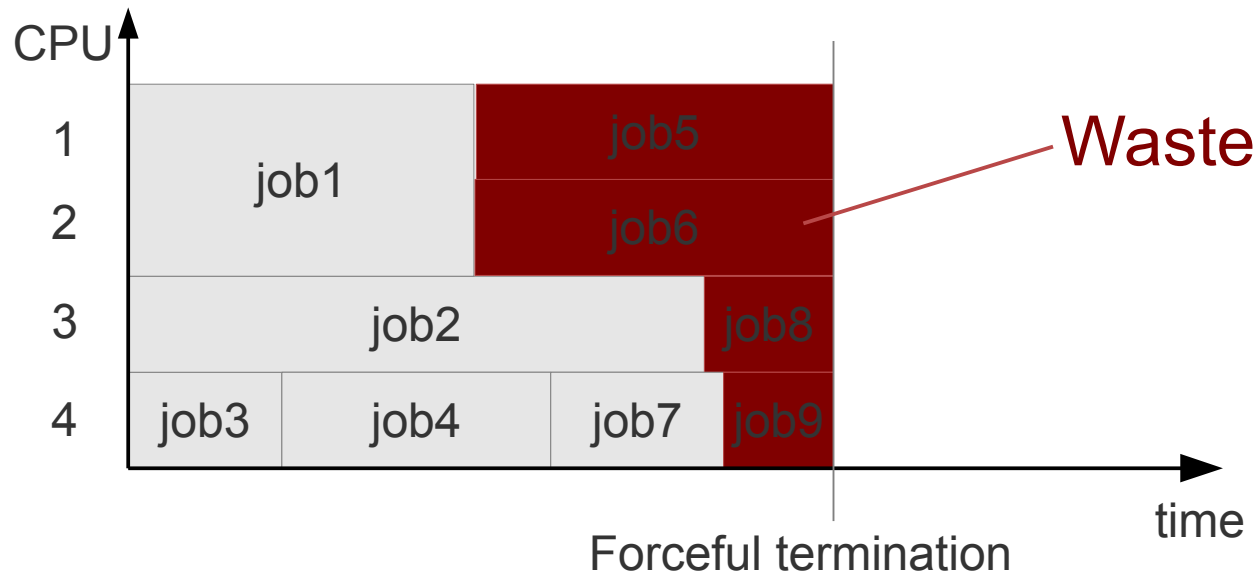
```
num_cpus = 2400
avg_job_runtime = 4h
Avg_turnover = 10/minute
```

An example

SLA likely to force terminations

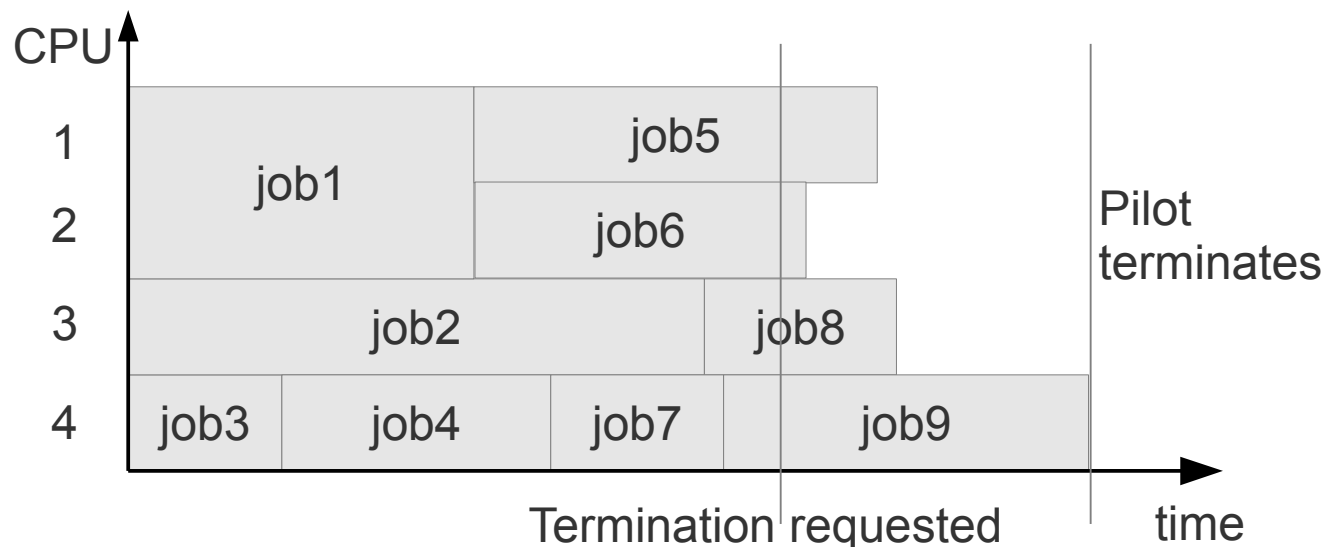
Need for termination smartness

- Random forceful job termination is undesirable
 - For most Grid jobs it means lost work
 - Pilot jobs not much better
 - Work of currently running user job(s) likely to be lost



Pilot jobs are special

- In one aspect, pilot jobs are special
 - Compared to typical Grid jobs
 - Pilot jobs **could go cleanly away on a relatively short notice**
 - Terminate on served user job termination
- Short compared to total runtime



Adding smartness to termination of pilot jobs

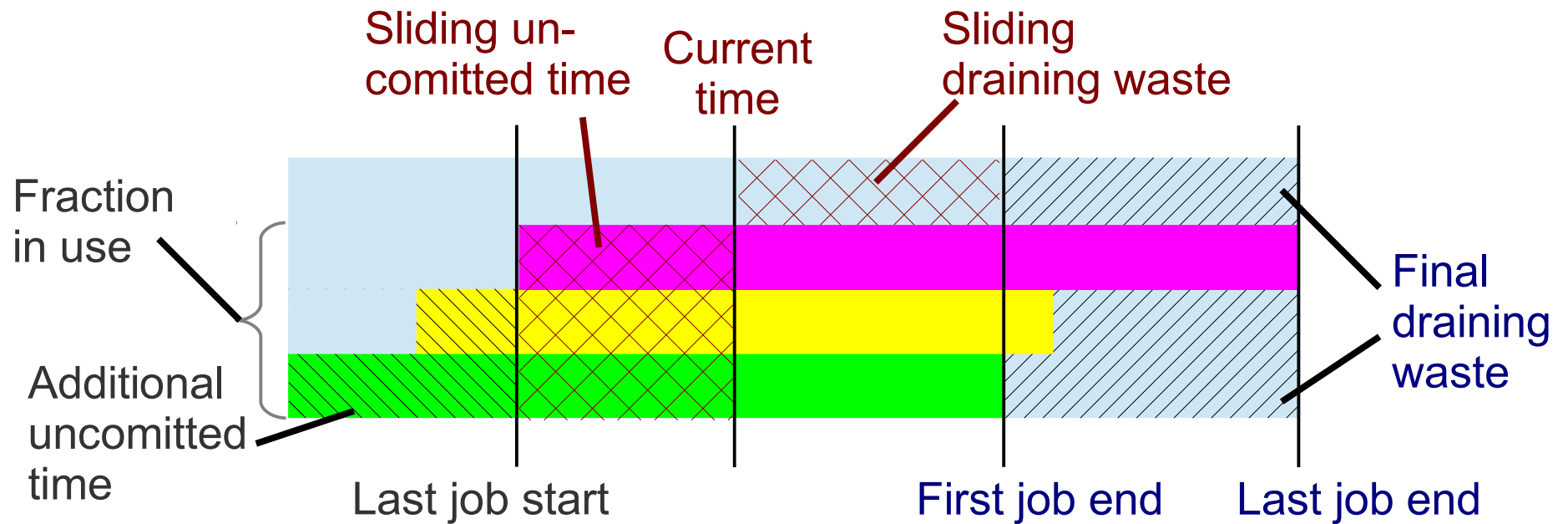
- **Resource providers are not evil** (the vast majority, at least)
 - But **currently lack technical means** for implementing a smart termination policy
- What is needed is
 - Enough information for **picking the best victims**
 - A mechanism to **request termination of the chosen pilot**

Selecting the cooperative victim

- Most HTC systems have some sort of notion of priority between users
 - So the **victim** pilot job will come from **the least privileged user**
 - **No new info needed here**
 - **Within the jobs of that user**, the ideal victim will likely be the one that results in either
 - Minimal waste, or
 - Minimal time-to-completion
 - The pilot owner may have preferences, too
- Need a way for the pilot to communicate relevant info to the resource provider**

Information available to the pilot

In a picture



Legend:
 Information known at discrete intervals
 Estimated information
 Information that can be derived

Communication channel

- The pilots must provide the needed information **in a standardized way**
 - It is not reasonable to expect the resource owner to know the internal workings of all pilot SW!
- Our proposal is to use **a file in the pilot's startup dir as the communication channel**
 - Simple text file, with each line containing `<key> = <value>`

File name:
.pilot.ad

Proposed attributes

- UNIX time (integer)
 - **LAST_JOB_START**
 - **FIRST_EXP_JOB_END**
 - **LAST_EXP_JOB_END**
 - **LAST_MAX_JOB_END**

} Good faith estimate

} Guaranteed
- Fractions, integer in the [0-1024] range (with 1024 meaning 100%)
 - **USED_FRACTION1k**
 - **ADD_UNCOM_TIME1k**
 - **ADD_FINAL_EXP_WASTE1k**

} Integral of fraction over time
- Arbitrary integer
 - **PRIORITY_FACTOR**

} Higher is better

File name:
.pilot.ad

Numbers only need
to be updated
on job start/end

Additional goodies - Heartbeat

- Resource providers have been **burned by stuck jobs** in the past
 - Short runtime limits make sure they do not waste much
 - Long running jobs can mitigate by providing a **heartbeat**
- We propose to use the **file's last modification time as heartbeat**
 - A stuck pilot is *unlikely* to do it

File name:
.pilot.ad

Additional goodies – Current state

- A pilot job **may already be in draining state**
 - i.e. will go away as soon as the current user job(s) terminate
 - Either due to a **previous order** from the resource provider, **or internal logic**
- More often than not, you do **not** want to **look for additional victims if there are sufficient pilots already draining**
- **We thus propose to add an additional attribute:**
 - **CAN_POSTPONE_LAST_JOB** – Boolean

File name:
.pilot.ad

Terminating the victim pilot

- Once a victim pilot is chosen
 - The resource owner must **order the termination of the chosen pilot**
- Two options:
 - **Hard kill** ← **Currently only option**
 - **Ask the pilot to gracefully terminate on its own**
 - Possibly within a set grace period

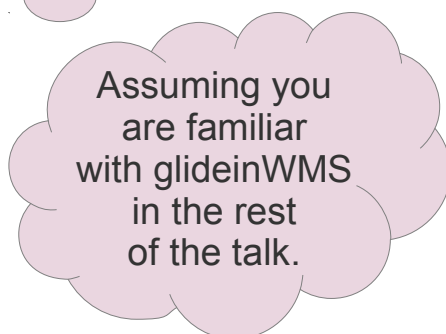
Requesting pilot termination

- Once again, we need **a standardized channel**
 - It is not reasonable to expect the resource owner to know the internal workings of all pilot SW!
- Our proposal, again, is to use a file in the pilot's startup dir as the communication channel
(but this time the resource provider writes, and pilot reads)
 - Again, simple text list of
<key> = <value>
- Two attributes defined
 - **VACATE_DESIRED** - Boolean
 - **PAYLOAD_DEADLINE** - Unix timestamp, hard deadline (optional)

File name:
.site.ad

Building a prototype

- We have a **prototype version of glideinWMS** that supports the above spec
 - Has been deployed on a subset of the production instances
 - Has been submitting instrumented glidein pilots for about a month
- We have **configured 3 USCMS sites to recognize instrumented pilots**
 - And raised the pilot job limits to 1 week

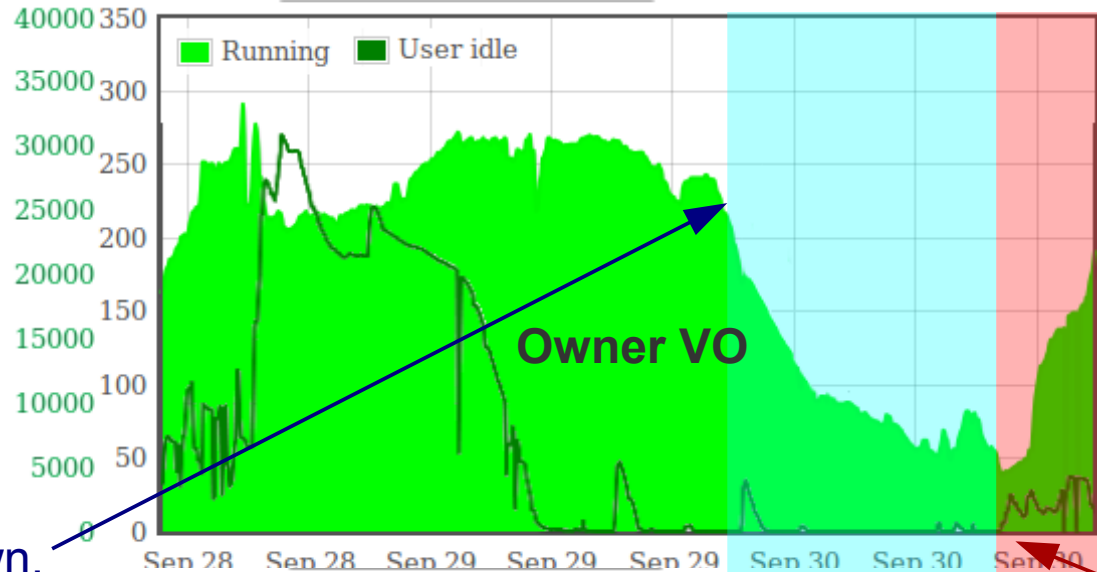


Assuming you are familiar with glideinWMS in the rest of the talk.

Running out of talk time

- Not enough time left to go over the details of the prototype
- Details available
 - Later on these slides, which are posted on the conference site
 - In the paper

Just a final picture



Owner VO has little work, pilots scaling down.

Opportunistic VO takes the unused slots.



Owner VO needs more resources.

Site admin requests opportunistic VO to go away.

See the inversion in running trend after a short delay.

Conclusions

- Scientific communities are moving to multi-CPU pilot jobs
- In order to keep relative overheads low, longer lived pilot jobs are needed
- Long pilot jobs will force resource providers to forcefully terminate a significant fraction
 - Requiring smart job termination strategies
- We address this by **proposing a standardized mechanism** to exchange relevant information between pilot jobs and resource providers

Acknowledgements

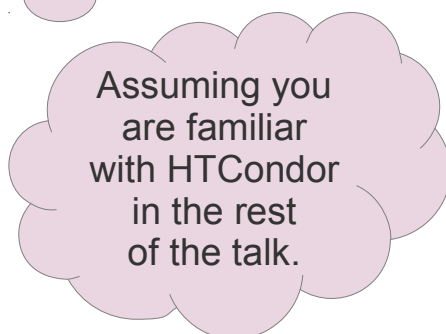
- This work was partially sponsored by the US National Science Foundation under Grants No. PHY-1148698 and PHY-1120138.

Additional material

For offline consultation

Building a prototype

- We have a **prototype version of glideinWMS** that supports the above spec
 - Has been deployed on a subset of the production instances
 - Has been submitting instrumented glidein pilots for about a month
- We have **configured 3 USCMS sites to recognize instrumented pilots**
 - And raised the pilot job limits to 1 week



Assuming you are familiar with HTCondor in the rest of the talk.

glideinWMS changes

Information Advertising

- The glidein wrapper sets **USED_FRACTION1k = 0** before HTCondor daemons start
- Using the HTCondor CRON mechanism, periodically:
 - Identify if any jobs are running by parsing the Startd logs
 - Write the status in `.pilot.ad`
 - If no running, set **USED_FRACTION1k = 0**
 - Else, define all the attributes, and set **USED_FRACTION1k = 1024**

File name:
.pilot.ad

Yes,
works only for
single CPU Pilots
for now.

Extending it
should be
easy enough.

glideinWMS changes

Receiving orders

- Again, HTCondor CRON used to parse .site.ad
 - Then publish as glidein attribute **GLIDEIN_RetireNow** – Boolean
- This attribute is then used by HTCondor
- If set to **true**:
 - No new jobs will start
 - The glidein will terminate as soon as the user job is done

File name:
.site.ad

Grid site configuration


- The authors have administrative privileges on 3 USCMS Grid sites
 - So we tampered with those
 - They all use HTCondor as batch system
- Two different approaches used
 - UCSD and UWM went for a semi-automated solution
 - UNL went for a fully automated solution
 - Using a pre-release version of HTCondor

Propagating Pilot Information

- Using HTCondor CRON-like capabilities
- Read the .pilot.ad files and propagate them into HTCondor ClassAds
 - Same syntax, just prepend **PILOT_** to all attribute names
- UCSD&UWM → Slot ClassAd
UNL → Job ClassAd

The fully automated solution

- Negotiator keeps track which jobs need to be preempted
- Instead of immediately sending a hard kill, set
VACATE_DESIRED = true
PAYLOAD_DEADLINE = <now + 2h>
 - If the Pilot is not gone in time, it will be hard killed



Currently requires
a custom HTCondor

The semi-automatic solution

- The system admin monitors the resource usage:
 - e.g. by using `condor_status` and `condor_userprio`
- When share imbalance is detected
 - Pick the victims
 - For each victim Pilot
 - Reach the startup dir with `condor_ssh_to_job`
 - Create a `.site.ad` containing **VACATE_DESIRED = true**

Currently a manual step, but should be easy to script

Fully automated from cmdline

No automated hard killing.

Side benefits

- Having a **standardized communication channel** between the resource provider and the pilots allows for **all kind of information to be exchanged**
- Obvious candidates include (but are not limited to):
 - Resource Provider: Batch slot description
 - e.g. # CPUs and a memory limit
 - Pilot: User job identities

Side benefits

- Having a **standardized communication channel** between the pilots and the pilots allows **all kind of information**

To be useful, requires standardization of attributes

- Obvious candidates include (but are not limited to):

- Resource allocation

Beyond the scope of this document