# Dynamical Provisioning of Cloud Computing Resources for Batch Processing

Marty Kandes

Distributed High-Throughput Computing Group
San Diego Supercomputer Center
University of California, San Diego

HEPiX Fall 2016
Lawrence Berkeley National Laboratory
October 19th, 2016

# About Me

- Open Science Grid (OSG) (Started February 2015)

  GlideinWMS Factory Operations (60%)

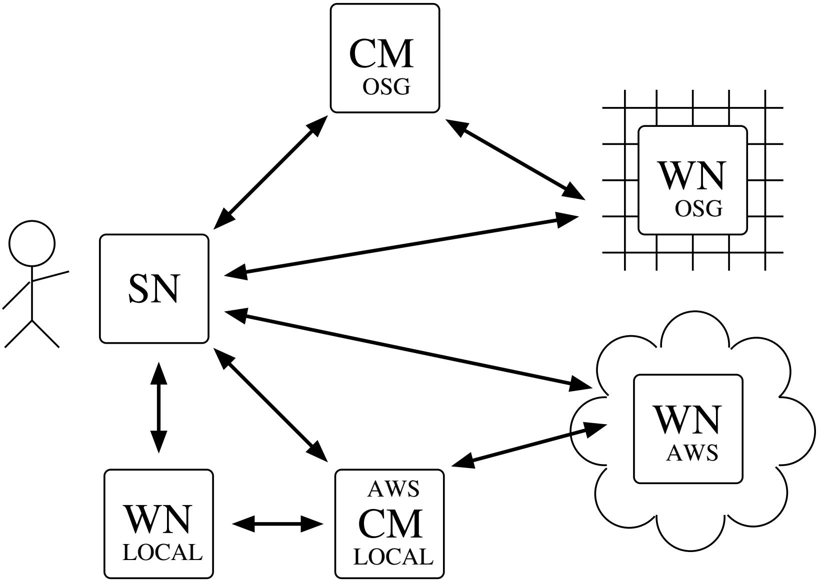
  Software Development & Testing (40%)

- Computational Physics, Applied Math, Traditional HPC

# Objective

Build a service for provisioning cloud-based computing resources (HTCondor execute nodes) that can be used to augment users' existing, fixed resources and meet their batch job demands.

# Vision (Extend HTCondor Pools to The Cloud)



# condor_annex = HTCondor + Amazon Web Services

Perl-based script that utilizes the AWS command-line interface and other AWS services to orchestrate the delivery of HTCondor execute nodes to your HTCondor pool.

Some key features:

- Supports bidding for spot instances.
- Instances sitting idle, not running user jobs will terminate after a fixed idle time (20 min).
- Each annex has a finite (max wallclock) lifetime.

Currently being developed into an HTCondor daemon that will provide the same and/or similar functionality as the prototype.

# How does condor_annex work?

1. Reads in and parses annex configuration options.
2. AWS CloudFormation manages complete life cycle of annex resources and services from annex creation to termination.
3. AWS Simple Storage Service (S3) stores shared configuration information (HTCondor configuration files, pool password).
4. AWS Autoscaling Group (ASG) manages annex size.
5. AWS CloudWatch monitors annex resources and services. Most important metric is custom annex lease/lifetime.
6. AWS Identity and Accesss Management (IAM) Roles, AWS Lambda Functions, and AWS Simple Notification Service (SNS) help monitor and enforce annex lease.
7. AWS Elastic Compute Cloud (EC2) provides annex instances.

# How to install and configure condor_annex?

1. Sign-up for AWS account.
2. Generate credentials to acccess AWS CLI.
3. Generate keypair to allow SSH access to annex instances.
4. Configure HTCondor pool to use password authentication.
5. Build condor_annex-compatible Amazon Machine Image.
6. Install and configure AWS CLI on HTCondor submit node.
   ```
   yum install python-pip
   pip install awscli
   aws configure
   ```
7. Install and configure condor_annex on HTCondor submit node.
   ```
   yum install git
   yum install perl-JSON
   git clone
   https://github.com/htcondor/htcondor.git -b
   V8_5-condor_annex-branch
   ```
8. Make custom changes (e.g., firewalls, CCB) as necessary.

# How to run condor_annex?

```
/opt/htcondor/src/condor_annex/condor_annex
--project-id "$PROJECT_ID"
--region "$AWS_DEFAULT_REGION"
--vpc "$AWS_VPC_ID"
--subnet "$AWS_SUBNET_ID"
--keypair "$AWS_KEY_PAIR_NAME"
--instances $NUMBER_OF_INSTANCES_TO_ORDER
--expiry "$AWS_LEASE_EXPIRATION"
--password-file "$CONDOR_PASSWORD_FILE"
--image-ids "$AWS_AMI_ID"
--instance-types "$AWS_INSTANCE_TYPE"
--spot-prices $AWS_SPOT_BID >>  /condor_annex.log
```

# Disclaimers (condor_annex is still a prototype)

- AWS CloudTrail can (and should) be used to augment condor_annex provided logging information.
- Watch out for AWS account limits.

```
A client error (LimitExceededException) occurred when
calling the CreateStack operation:  Limit for stack has been
exceeded
```

- Known race-like conditions may occur.

```
10:07:51 UTC-0700 DELETE_FAILED AWS::SNS::Topic Topic User:
arn:aws:sts::720506099995:assumed-role/htcondor-annex-172-31-43-238
R94GI3W/awslambda_729_20160709170725120 is not authorized to
perform:  SNS:GetTopicAttributes on resource:
arn:aws:sns:us-west-2:720506099995:htcondor-annex-172-31-43-238-wor
```

- Hard-coded configuration (e.g., S3 URL for pool password).

# condor_annex User Beta Test I

Physics Computing Facility (PCF) UCLHC "Brick" @ UCSD

- Integrated computing platform for Physics faculty, students, and staff, but open to all UCSD researchers.
- User jobs may target local in-"Brick" resources (48 cores), CMS Tier 2 (7.8k cores); Comet @ SDSC (48k cores), as well as OSG and UCLHC resources.
- condor_annex will allow researchers to purchase AWS resources on-demand to run time-sensitive jobs as well.
- Ready for users to begin testing November 2016.
- `https://github.com/mkandes/condor_annex`

# condor_annex User Beta Test II

Open Science Grid (OSG) via Extreme Science and Engineering Discovery Environment (XSEDE)

- Login/submit node for XSEDE users to access OSG resources.
- condor_annex will allow XSEDE users to purchase AWS resources on-demand to run time-sensitive jobs as well.
- Ready for users to begin testing December 2016.
- `https://github.com/mkandes/condor_annex`

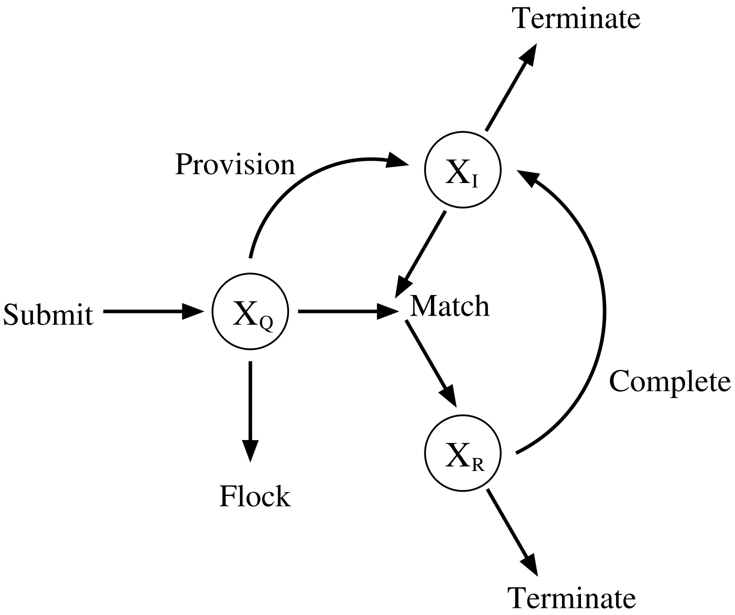
# Provisioning Problem

How many instances do we order with condor_annex to meet current user job demand?

# Optimization Problem vs. Control Problem

- Forget optimally scheduling jobs and resources; too hard.
- Seek to provision resources in a controlled way.
- Build a system that aims to provision resources safely and use them as efficiently as possible.

# Provisioning Model I/II: State Diagram



# Provisioning Model I: System of Equations (ODEs)

$$\frac{dX_Q}{dt} = \Sigma_Q - \sigma_{QIR} X_I X_Q$$
$$\frac{dX_I}{dt} = \sigma_{QI} X_Q - \sigma_{QIR} X_Q X_I + \sigma_{RI} X_R - \sigma_{IT} X_I$$
$$\frac{dX_R}{dt} = \sigma_{QIR} X_Q X_I - \sigma_{RI} X_R - \sigma_{RT} X_R$$

- $X_Q$ is the number of jobs in the queue waiting to run
- $X_I$ is the number of idle machines (not running jobs)
- $X_R$ is the number of busy machines (busy running jobs)
- $\Sigma_Q$ is the rate of job submission
- $\sigma_{QIR}$ is the matchmaking rate
- $\sigma_{QI}$ is the provisioning rate
- $\sigma_{RI}$ is the completion rate
- $\sigma_{IT}$ is the idle-termination rate (1/20 min).
- $\sigma_{RT}$ is the running-termination rate (1/annex lifetime)

# Provisioning Model I: Equilibria

Solve.

$$\frac{dX_Q}{dt} = f_Q\left(X_Q, X_I, X_R\right) = 0$$
$$\frac{dX_I}{dt} = f_I\left(X_Q, X_I, X_R\right) = 0$$
$$\frac{dX_R}{dt} = f_R\left(X_Q, X_I, X_R\right) = 0$$

Find two equilibrium points.

$$X_Q^* = \frac{\Sigma_Q \sigma_{RT}}{2\sigma_{QI}\left(\sigma_{RI} + \sigma_{RT}\right)}\left[1 \pm \sqrt{1 + \frac{4\sigma_{IT}\sigma_{QI}\left(\sigma_{RI} + \sigma_{RT}\right)^2}{\Sigma_Q \sigma_{QIR}\sigma_{RT}^2}}\right]$$

$$X_I^* = \frac{\Sigma_Q \sigma_{RT}}{2\sigma_{IT}\left(\sigma_{RI} + \sigma_{RT}\right)}\left[-1 \pm \sqrt{1 + \frac{4\sigma_{IT}\sigma_{QI}\left(\sigma_{RI} + \sigma_{RT}\right)^2}{\Sigma_Q \sigma_{QIR}\sigma_{RT}^2}}\right]$$

$$X_R^* = \frac{\Sigma_Q}{\sigma_{RI} + \sigma_{RT}}$$

# Provisioning Model I: Stability of Equilibria

Find Jacobian.

$$J = \frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{df_Q}{dx_Q} & \frac{df_Q}{dx_I} & \frac{df_Q}{dx_R} \\ \frac{df_I}{dx_Q} & \frac{df_I}{dx_I} & \frac{df_I}{dx_R} \\ \frac{df_R}{dx_Q} & \frac{df_R}{dx_I} & \frac{df_R}{dx_R} \end{bmatrix}$$

Compute eigenvalues of Jacobian about equilibria.

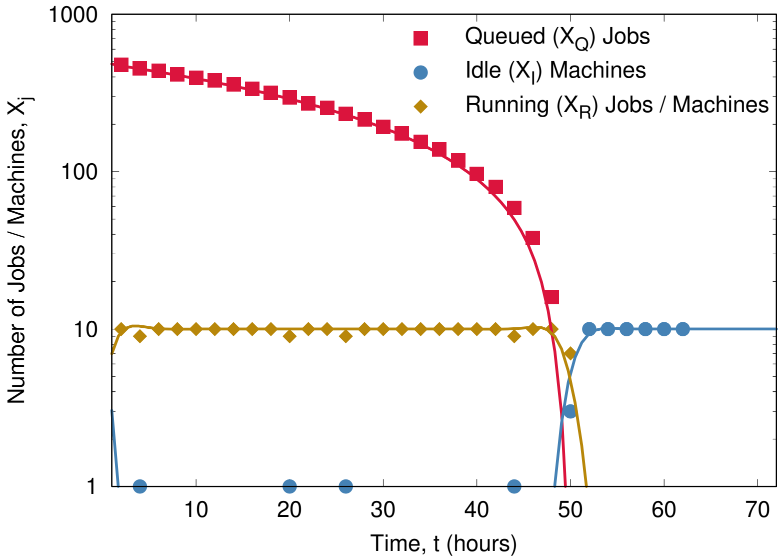
$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^*) + J(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) + \cdots$$

If the eigenvalues all have real parts that are negative, then the system is **stable** near the stationary point, if any eigenvalue has a real part that is positive, then the point is **unstable**.

# Simple Run Test: Protocol

1. Launch $M$ on-demand (t1.micro) instances.
2. Wait for on-demand instances to start-up and join the pool.
3. Submit $N$ jobs with average lifetime of $\tau_{RI}$.
4. Monitor $X_Q(t)$, $X_I(t)$, and $X_R(t)$ over time.
5. Test ends when $X_Q(t) = 0$ and $X_I(t) = M$.
6. Repeat for different $M$, $N$, $\tau_{RI}$.

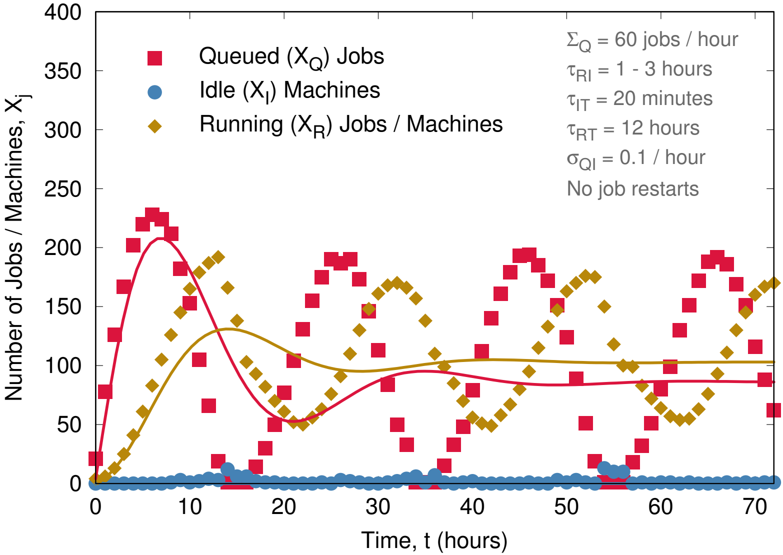
# Simple Run Test: Experimental Results v. PM1 (ODEs)



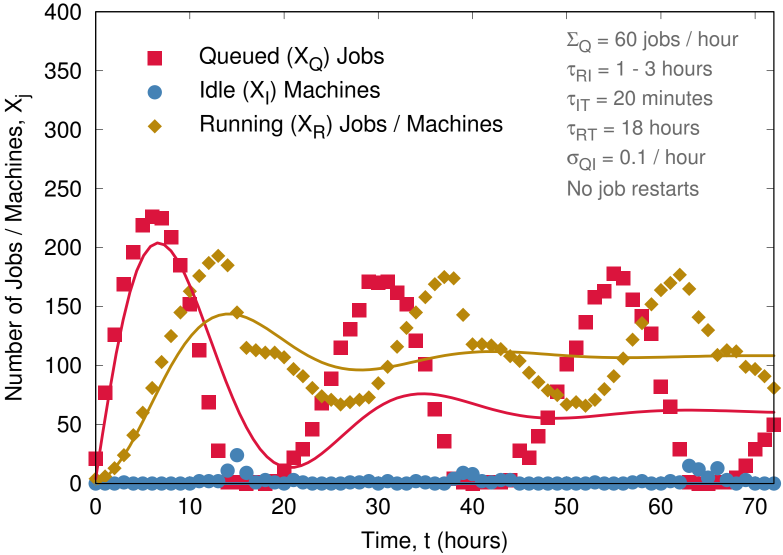
# Submission Response Test: Protocol

1. Fix annex lifetime, $\tau_{RT}$.
2. Submit jobs at rate of $\Sigma_Q$ with an average job lifetime of $\tau_{RI}$.
3. Order $\sigma_{QI} X_Q(t)$ instances with condor_annex every hour.
4. Monitor $X_Q(t)$, $X_I(t)$, and $X_R(t)$ over time.
5. Test ends after 72 hours (3 days).
6. Repeat for different $\tau_{RT}$, $\Sigma_Q$, $\tau_{RI}$, $\sigma_{QI}$.

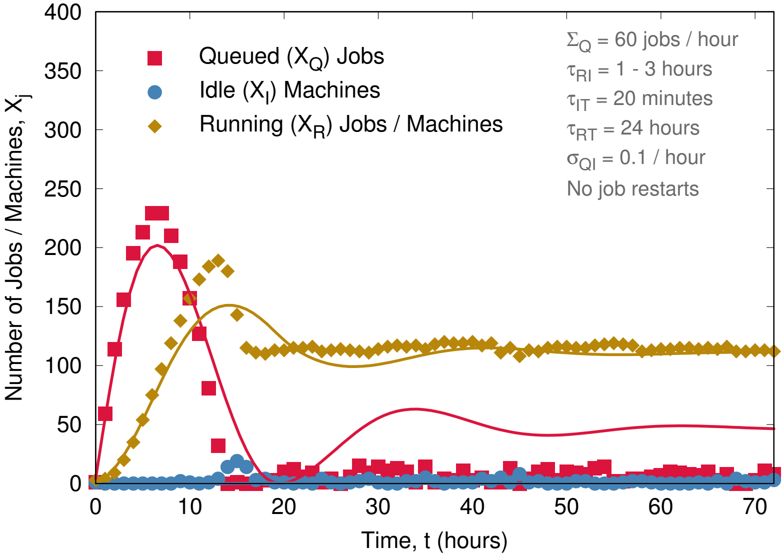
# Submission Response Test: Exp. Results v. PM1 (ODEs)



# Submission Response Test: Exp. Results v. PM1 (ODEs)



# Submission Response Test: Exp. Results v. PM1 (ODEs)



# Possible Source of Oscillations

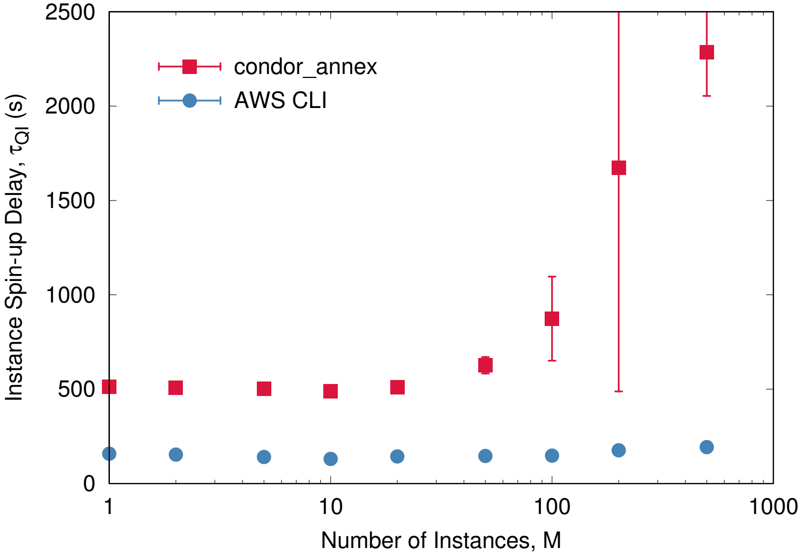
In the mathematical theory of bifurcations, a **Hopf bifrucation** is a critical point where a system's stability switches and a periodic solution arises.

In mathematics, **delay differential equations** (DDEs) are a type of differential equation in which the derivative of the unknown function at a certain time is given in terms of the values of the function at previous times.

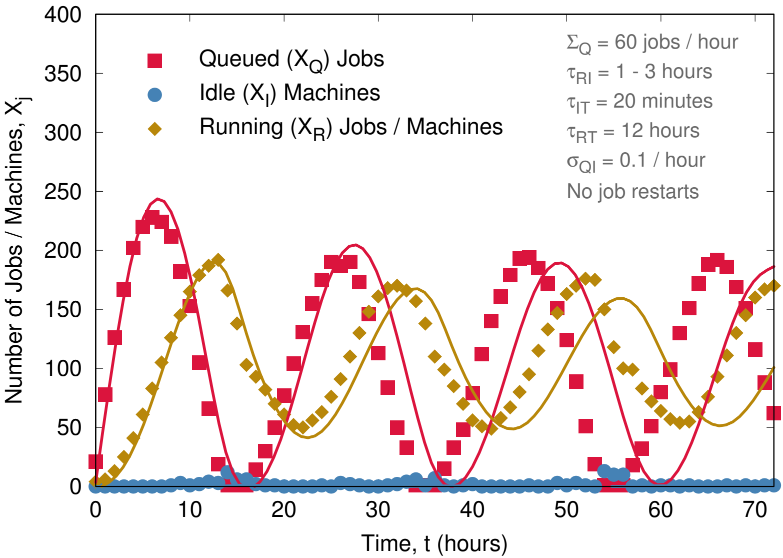
# Instance Spin-up Delay Test: Protocol

1. Submit $N$ jobs with job lifetime of $\tau_{RI}$.
2. Order $M < N$ instances with annex/instance lifetime of $\tau_{RT}$.
3. Monitor $X_Q(t)$, $X_I(t)$, and $X_R(t)$ over time.
4. Test ends when $X_R(t = \tau_{QI}) = M$.
5. Record instance spin-up delay, $\tau_{QI}$.
6. Repeat for different $M$.
7. Compare condor_annex vs. AWS CLI.

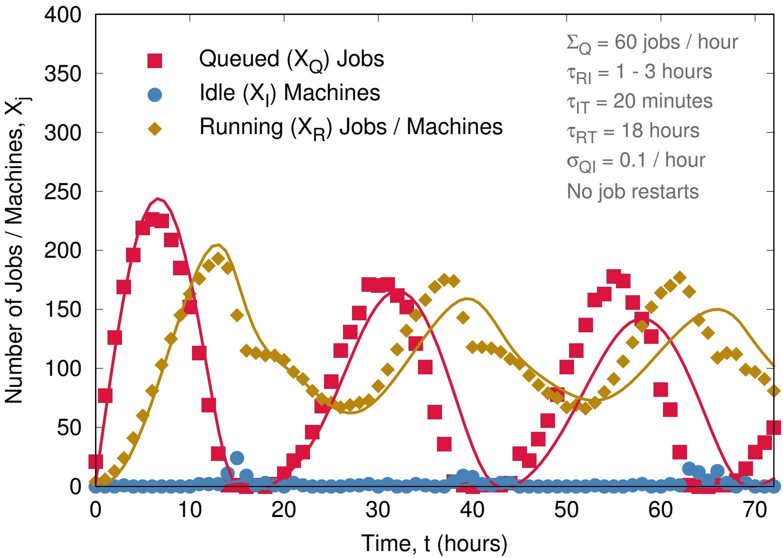
# Instance Spin-up Delay Test: Experimental Results



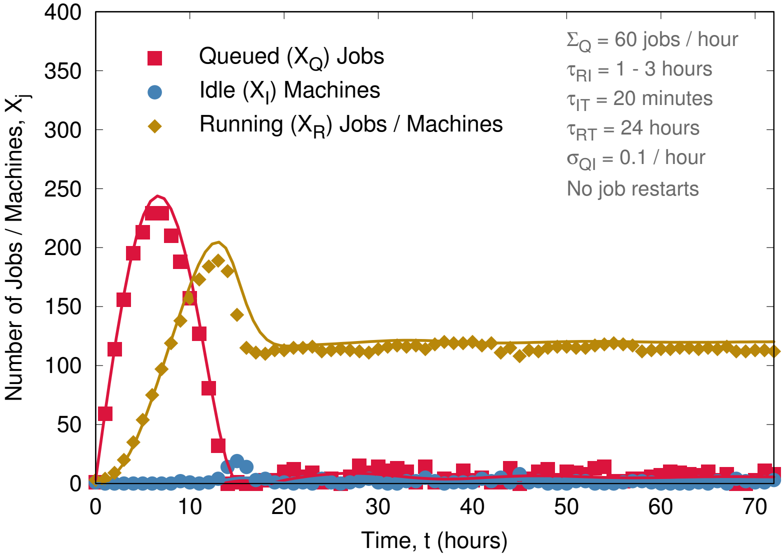
# Submission Response Test: Exp. Results v. PM2 (DDEs)



# Submission Response Test: Exp. Results v. PM2 (DDEs)



# Submission Response Test: Exp. Results v. PM2 (DDEs)



# Provisioning Service Development & Testing Plan

- Python (HTCondor bindings) + SQLite application database
- Database schema developed (June/July 2016)
- Database schema implemented (August 2016)
- System/administrative functions (December 2016)
- Large Workflows / Provisioning Model I (January 2017)
- Test with condor_annex prototype (Feburary 2017)
- condor_annex as an HTCondor daemon (?)
- Generalized Workflows / Provisioning Model II/III/? (?)
- `https://github.com/mkandes/zephyr`

# Acknowledgments

**Todd Miller** @ UW - Madison
Center for High Throughput Computing, HTCondor

**Frank Würthwein** @ UCSD
Open Science Grid, Executive Director

**Jeff Dost** @ UCSD
Open Science Grid, Glidein Factory Operations / Software

**Edgar Fajardo** @ UCSD
Open Science Grid, Software

**Heather Matson** @ Amazon (`matsonh@amazon.com`)
AWS Higher Education, Enterprise Account Manager

**Sanjay Padhi** @ Amazon (`sanpadhi@amazon.com`)
AWS Scientific Computing, Principal Technical Business Manager

# AWS Cloud Credits for Research Program

Supports researchers who seek to:

- Build cloud-hosted publicly available science-as-a-service applications, software, or tools to facilitate their future research and the research of their community.
- Perform proof of concept or benchmark tests evaluating the efficacy of moving workloads or open data sets to the cloud.
- Train a broader community on the usage of cloud for research workloads via workshops or tutorials.
- `https://aws.amazon.com/research-credits/`