A Brief Overview of CIPRES Job Management

Mark A. Miller
San Diego Supercomputer Center
CIPRES is a classic browser-based gateway designed to submit jobs to multiple hosts. (REST services are now available also)
CIPRES Use Case:

• 25,000 submissions per month (34 per hour)
• Some require long run times (up to 168 h max)
• Many codes have no restart ability (some do)
• Relatively few cores (2 - 64, generally)
• Small input data sets (1 - 50 MB, generally)
• Many community codes (more than we can implement)
• Relatively unsophisticated (from HPC point of view) users
• Less than 2 fte of support staff
CIPRES Use Case:

• 25,000 submissions per month (34 per hour)
• Some require long run times (up to 168 h max)
• Many codes have no restart ability (some do)
• Relatively few cores (2 - 64, generally)
• Small input data sets (1 - 50 MB, generally)
• Many community codes (more than we can implement)
• Relatively unsophisticated (from HPC point of view) users
• Less than 2 fte of support staff

Your use case will be different…each use case has its own issues
The Workbench Framework (Java) deploys generic “tasks”....
Specific information is coded in a Central Registry
Tasks are sent to remote execution hosts.
Web Application

- Task Update servlet
- Create Task Forms
- View Task Form

User

Create/validate new task

Submit

Display tasks in GUI

Update status

CIPRES DB

Running

Tasks

Execution Hosts

Submit
curl, task is done

SDK

- Validate, create configuration files
- Store in "tasks" table, if valid, enter in running tasks table as "new"

CIPRES DB
Users interact with Web forms to configure jobs.
Jobs configured based on user input

Web Application
- Task Update servlet
- Create Task Forms
- View Task Form
- Update status
- Submit
- Display tasks in GUI

Execution Hosts
- curl, task is done
- Submit

CIPRES DB
- Running
- Tasks

SDK
- Validate, Create configuration files

User
- Create/validate new task

CIPRES DB
- Store in "tasks" table, if valid, enter in running tasks table as "new"
Submissions are validated, Entered in the database for tracking
Web Application

- Task Update servlet
- Create Task Forms
- View Task Form
- Update status
- Submit
- Display tasks In GUI

Execution Hosts

- Jobs are submitted to a specific remote host

User

- Create/validate new task

SDK

- Validate, Create configuration files
- Store in "tasks" table, if valid, enter in running tasks table as "new"

CIPRES DB

- curl, task is done

- Submit

CIPRES DB

- Submit
User is notified of status changes via the interface
Questions we asked early on:

• Why do job submissions fail?
• When do we care a lot, a little, or not at all?
• What parts does CIPRES have control over
• What parts are not under CIPRES control?
Job Attrition on the CIPRES Science Gateway*

*March – August 2010
Error Impact analysis

<table>
<thead>
<tr>
<th>Error Type</th>
<th>CPU time</th>
<th>User</th>
<th>Staff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input error (user)</td>
<td>0</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>System error</td>
<td>0</td>
<td>med</td>
<td>low</td>
</tr>
<tr>
<td>Communication error</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>Unknown error</td>
<td>?</td>
<td>med</td>
<td>low</td>
</tr>
</tbody>
</table>
Job Submissions/Results Retrieval is managed by daemons

**submitJobsD**
1. Find all "new" tasks
2. Submit to correct execution host
3. Set status to "submitted"

**loadResultsD**
1. Find all "done" tasks
2. Transfer results to CIPRES DB
3. Remove job from "WorkQ"
**submitJobsD**

1. Find all “new” tasks
2. Submit to correct execution host
3. Set status to “submitted”

**Submission daemon:**

1. Authenticates
2. Creates working directory
3. Stages input files
4. Causes a script on the host to create a submit script and submit the job
submitJobsD

1. Find all “new” tasks
2. Submit to correct execution host
3. Set status to “submitted”

Submission:

1. Authenticate
2. Create working directory
3. Stage input files
4. Script on submission host creates submit script

A custom submit script is created to interact with the submit daemon for each execution host
Retrieve results daemon:

1. Communicates with a custom script on the execution host about completed tasks (e.g. qstat –u cipres)
2. Transfers any completed results to CIPRES DB
3. Removes the job from the WorkQ

CIPRES DB

Execution Hosts

<table>
<thead>
<tr>
<th>loadResultsD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Find all “done” tasks</td>
</tr>
<tr>
<td>2. Transfer results to CIPRES DB</td>
</tr>
<tr>
<td>3. Remove job from “WorkQ”</td>
</tr>
</tbody>
</table>
User errors:

About 20% of jobs have immediate “failure” due to user error.

% success per code depends on whether input files are created by the user or a software package.

Can help users detect errors before submission, or even suggest corrections.

Cost of the error to CIPRES: zero, unless reported. If reported: about 1.5 fte months per year about $15-20,000 per year.
System errors:

3-4% jobs experience some form of system failure.

98% of system failure occurs in crisis islands;

Crisis island: the system is down. All submissions fail and are retried and fail again.

2% is sporadic scheduler transient unavailability (could be solved by re-trying). 100/250,000 jobs a year could be helped.
Kinds of system errors:

- Expired or unavailable allocation
- Bug in the CIPRES interface causes misconfiguration (these are rare)
- Emergency maintenance
- Unavailability of the scheduler/system for unknown reasons
- Something unexpected in the interaction between CIPRES and the infrastructure.
Kinds of system errors:

- Expired or unavailable allocation
- Bug in the CIPRES interface causes misconfiguration (these are rare)
- Emergency maintenance
- Unavailability of the scheduler/system for unknown reasons
- Something unexpected in the interaction between CIPRES and the infrastructure.
Features we have added:

- ability to halt submissions from a given user account
- ability to monitor usage by each account automatically
- ability for users to track their SU consumption
- ability to forecast SU cost of a job for users
- ability for user to kill a job
- ability to charge to a user’s personal XSEDE allocation
- notification of job failures mailed to staff
Questions we ask now:

• How can we ensure job runs are configured correctly?
• How can we detect wasteful misconfigured jobs quickly?
• What does it mean when there are no submissions?
Features we would like to add:

• Streamlined ability to deploy on new hosts
• User can determine what host a job is submitted to
• Route jobs to hosts based on job characteristics and host availability
• Automatic configuration of jobs, so users can’t make mistakes
• Better tools for job restarting
• User-friendly error messaging
• Automated log data gathering for internal failure analysis
• Job submission retries on failure