Building a complex user application for CMS with DAGMan

Brian Bockelman
Computing is Hard

• I will discuss personal experiences and lessons learned when building a new analysis system for CMS.

  • I selected four favorite problems I thought the community would find interesting.

  • I tell the problems, not the success stories.

• The project has a large team (including present and past members).

  • Each one would give a different point-of-view.

• So consider my words an opinion, not the word of God.
What’s in a CMS Analysis?

• For the purpose of presentation, a CMS analysis is:
  
  • **Input**: Some code tarball and configuration.
  
  • **Input**: Name of dataset.
  
  • **Output**: Dataset resulting from running the CMS application and user code on the input dataset.
  
  • The rest is just details!
Ok, maybe more details

• A **dataset** is composed of one or more files.

• We break the analysis **task** into a set of **jobs**:
  
  • Each job can be run independently.

  • Each job reads in one or more files from the dataset.

    • The sites where the job can run are determined from the location of the datasets.

  • Each job has a single result file that must return to a specified location.
Let’s Get CRABby

• CMS obviously has a system to do this - we found the Higgs, didn’t we?

• CRAB = CMS Remote Analysis Builder. Currently, CRAB2.

• The CRAB2 client runs on the user’s development node.
  • Splitting of task to jobs is done on client.
  • Job submits are done from client.
  • Job tracking (done? failed? needs resubmit?) done from client.
  • Results are copied from worker node to a user-specified remote storage system.

• CRAB2 allows multiple job tracking backends. >95% jobs use HTCondor, but about 10 different ones have been in existence.
CRAB2

• User writes a **config file and code.**
CRAB2

• User writes a config file and code.

• “crab create”
CRAB2

- User writes a config file and code.
- “crab create”
- “crab submit”
CRAB2

• User writes a config file and code.
• “crab create”
• “crab submit”
• “crab status” (times many)
CRAB2

- User writes a config file and code.
- “crab create”
- “crab submit”
- “crab status” (times many)
- “crab getoutput”
  - Retrieves stdout - job outputs go to storage service.
CRAB2 by the numbers

- Across the system, there are:
  - 200,000 jobs / day.
  - 25,000 cores used continuously.
  - 300 users.
  - 50 execution sites
  - One user support person.
What’s Wrong With This?

• CRAB2 has some seriously annoying flaws:
  
  • The job “babysitter” is the user, not a running process. [If a job fails 1 minute after you go to bed, it won’t be run until you wake up and press “resubmit”.
  
  • All logic is client-side; users must install a new client to get bugfixes. Users hate installing updates.
  
  • High rate of failures, especially stageout. (More on this later)
  
  • Each physics group has a hapless grad student whose role in life is to run that group’s workflows.
Enter CRAB3

• The CRAB task is basically a DAG; CRAB2 relies on “DAGGradStudent”.

• For CRAB3, we decided to have a backend which does task tracking instead of job tracking.

• We experimented with several task layers, then settled on DAGMan about 6 months ago.
CRAB3 Architecture

Submit host

Task Parameters and Code

HTTP POST

CMS Web host

CRAB3 Frontend Server

CRAB3 Backend Server
(Job creation, DAG submission)

HTTP GET
CRAB3 Architecture

Submit host

HTTP POST

CMS Web host

CRAB3 Frontend Server

HTTP GET

CMS CRAB3 Backend

CRAB3 Backend Server
(Job creation, DAG submission)

Submit host

Task Parameters and Code

CMS Web host

CRAB3 Frontend Server

CMS CRAB3 Backend

CRAB3 Backend Server
(Job creation, DAG submission)

Condor Schedd Host

Per-Task DAG

Job 1 -> Post Job

Job N -> Post Job

HTCondor submit
CRAB3 Architecture

HTCondor Schedd Host

Per-Task DAG

Job 1 → ASO Job

Job N → ASO Job

glideinWMS pool

Job 1 instance
HTCondor startd

Job N instance
HTCondor startd

(and 20,000 more)
What could possibly go wrong?
Interesting Problem 1: Remote Submit

- Strong push from CMS to centralize operations behind our central web portal.
- Submit hosts are operated by a separate team.
- Remote submit must somehow include user’s X509 proxy certificate.
Solution: Python and MyProxy

• Python bindings provide the web portal with a mechanism to perform status queries. The backend server uses these for job submits.

• We have translated all the CRAB “verbs” to equivalent action on the DAGMan task in the schedd.

  • For example, if the DAG fails, the job goes onto HOLD; the equivalent of `condor_release` performs the resubmit.

• User uploads their X509 proxy to a separate MyProxy server and gives permission to the CRAB3 backend for retrieval; the backend then retrieves the proxy and pushes it to the schedd with the job submit.

  • We had to extend the python bindings to allow the backend to periodically push updated proxies to the schedd.
Interesting Problem 2: Task Interaction

- When running across 50 sites, failures are a fact of life. We use DAGMan post-scripts extensively to determine whether a failure should be retried.

- Inevitably, some jobs still fail (some errors require a human to examine or fix) permanently.

  - Consequently, the grad student then runs down the hall to yell at the sysadmin to fix things.

  - Once fixed, the grad student wants to be able to resubmit just that failed job!

  - Alternately, maybe the grad student would like to kill one DAG node while they investigate a job failure.

- Whether killing or resubmitting, we don’t want to upset the rest of the jobs! Hence, we don’t want to use rescue DAGs.
Solution

- **A wide series of hacks!**

- To kill individual running jobs, we simply remove them from the schedd queue.

- To kill the task, we hold the corresponding DAGMan job. This sends a SIGUSR1 signal to DAGMan, which removes all jobs from queue.

- To resubmit jobs, we change the hold signal to SIGKILL and hold/release the DAGMan job.
  - On release, the wrapper will rewrite the task’s user log to change the exit code.
  - The condor_dagman process will restart and connect to running jobs using the log file.
Interesting Problem 3: Task Monitoring

• Each job in the user task is a node in a DAG. What’s the state of the node?
  
  • If there is no corresponding job in the schedd, is it because the node failed? The node is successful? DAGMan has hit an idle job limit? Running a post-job?
  
  • Worse - condor_q is too heavyweight on the schedd to allow users to query directly for jobs in the schedd.
  
• How do we concisely present the relevant task information to users?

• How do we highlight potential problems?
Solution

• The DAGMan node_status file provides information about each node (one line per node in the DAG).

  • The file is placed in a web-accessible directory of the schedd; the CRAB frontend parses it and sends it to the user.

  • We worked closely with the HTCondor team to shake the bugs out of the handling of this file and deliver a new file format.

• We delivered a rewrite of the condor_q protocol to reduce the resources needed for queries.

• We wrote a custom monitoring application, glidemon, to summarize the task status.
```bash
-bash-4.1$ crab status crab_xrootd_test_14 -u
Task name: 140427_011655_vocms20:bbockelm_crab_xrootd_test_14
Task status: SUBMITTED
Details:
   failed  37.8% (1430/3783)
   finished 54.4% (2057/3783)
   running  7.8% (296/3783)

Site Summary Table (including retries)

<table>
<thead>
<tr>
<th>Site</th>
<th>Runtime</th>
<th>Waste</th>
<th>Running</th>
<th>Successful</th>
<th>Stageout</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1_DE_KIT</td>
<td>517:48:17</td>
<td>3210:55:46</td>
<td>13</td>
<td>97</td>
<td>0</td>
<td>253</td>
</tr>
<tr>
<td>T1_IT_CNAF</td>
<td>47:30:16</td>
<td>370:41:27</td>
<td>3</td>
<td>12</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>T1_TW_ASGC</td>
<td>28:18:49</td>
<td>104:23:49</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>T1_UK_RAL</td>
<td>62:13:12</td>
<td>782:06:28</td>
<td>1</td>
<td>15</td>
<td>0</td>
<td>111</td>
</tr>
<tr>
<td>T1_US_FNAL</td>
<td>0:00:00</td>
<td>275:14:05</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>393</td>
</tr>
<tr>
<td>T2_AT_Vienna</td>
<td>79:41:24</td>
<td>145:07:19</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>T2_BE_IHBE</td>
<td>600:19:45</td>
<td>426:31:39</td>
<td>5</td>
<td>63</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>T2_BE_UCL</td>
<td>1002:10:53</td>
<td>1450:34:27</td>
<td>69</td>
<td>63</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>T2_BR_SPRACE</td>
<td>632:12:39</td>
<td>385:16:44</td>
<td>13</td>
<td>114</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>T2_CH_CERN</td>
<td>135:03:54</td>
<td>571:53:50</td>
<td>3</td>
<td>23</td>
<td>0</td>
<td>70</td>
</tr>
<tr>
<td>T2_CH_CSCS</td>
<td>977:35:00</td>
<td>1683:48:52</td>
<td>40</td>
<td>172</td>
<td>0</td>
<td>152</td>
</tr>
<tr>
<td>T2_DE_DESY</td>
<td>324:51:02</td>
<td>839:32:14</td>
<td>6</td>
<td>49</td>
<td>0</td>
<td>93</td>
</tr>
</tbody>
</table>
```
Task
140425_194504_vocms20:bloom_crab_data_test_140425_private

(Only current jobs are shown. Show all jobs including resubmitted)

Summary of job failures
Top 5 Error Types

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>Error Code (long)</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>65 (8001)</td>
<td>Other CMS Exception</td>
</tr>
<tr>
<td>2</td>
<td>134 (134)</td>
<td>Abort (ANSI) or IOT trap (4.2 BSD)</td>
</tr>
<tr>
<td>1</td>
<td>85 (8021)</td>
<td>FileRead Error (May be a site error)</td>
</tr>
</tbody>
</table>

Top 5 Sites

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>Site Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>T2_US_Purdue</td>
</tr>
<tr>
<td>1</td>
<td>T2_US_Nebraska</td>
</tr>
<tr>
<td>1</td>
<td>T2_UK_SGrid_RALPP</td>
</tr>
</tbody>
</table>

Individual job information (Log files)

Show 25 entries

<table>
<thead>
<tr>
<th>Job Id</th>
<th>Id in Task</th>
<th>Version</th>
<th>Submit Server</th>
<th>Submit</th>
<th>Start</th>
<th>End</th>
<th>Site</th>
<th>Wall Time</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1516147.0</td>
<td>1</td>
<td>0</td>
<td>vocms20.cern.ch</td>
<td>25 Apr 21:46</td>
<td>25 Apr 22:01</td>
<td>25 Apr 22:02</td>
<td>T2_US_Purdue</td>
<td>00:01:41</td>
<td>Completed</td>
</tr>
<tr>
<td>1516148.0</td>
<td>0</td>
<td>0</td>
<td>vocms20.cern.ch</td>
<td>25 Apr 21:46</td>
<td>25 Apr 22:01</td>
<td>25 Apr 22:02</td>
<td>T2_US_Purdue</td>
<td>00:01:41</td>
<td>Completed</td>
</tr>
</tbody>
</table>
Interesting Problem 4: Output file management

• Until now, I’ve only described half the battle!

• Depending on the user, outputs are somewhere between 1 and 3,000MB.

• (Average job output size) X (# of jobs) = we can’t use HTCondor file transfer.

• Instead, the job output is copied to a storage system the user specifies.

• The #1 cause of failure in CRAB2 is **failed stageout**.
Introducing ASO

• To reduce error rates and improve CPU efficiency, the CRAB3 job wrapper copies output files to the “nearest” storage service, register the output file in a database, then exits.

• The Asynchronous StageOut (ASO) server will see the new output file and copy it to the user-specified storage service.

• Once the file is at the final location, the job can be marked as done.
ASO - Copy

CERN

CouchDB  ASO Server, CouchDB

Worker Node  Storage

Source Site

Storage

Destination Site
ASO - Register

CERN

CouchDB  ASO Server, CouchDB

Worker Node  Storage

Source Site

Register

Storage

Destination Site
ASO - Copy to Remote

Worker Node → Storage
Source Site → COPY

Storage → Storage
Source Site → Destination Site

CERN

CouchDB
ASO Server, CouchDB

Register
Setup xfer
Setup xfer
A Beautiful Can Of Worms

• **However** - DAGMan cannot mark a job as completed until ASO says the file has safely arrived.

  • Thus, the post-job must not exit until ASO says the transfer is done.

  • Each post-job takes up X MB of RAM, so we must limit the total number of post-jobs to Y per task.

  • Once Y post-jobs are running, *no other job can complete*.

  • Hence, the combination of DAGMan + ASO has the potential for causing head-of-line blocking behavior.

• We must have timeouts to make sure job completion progresses.
But wait, there’s more!

• Essentially, we have two databases we must synchronize - ASO and DAGMan. All actions need to be idempotent or have a concurrency protocol.

• How do we prevent terabytes of data from accumulating at remote sites (flow control)?
  
  • CRAB2 had a crude and wasteful (but effective) way of throttling the production of output data - idle the CPU!

• The user and post-job need an indication of whether transfers are progressing.
  
  • Similar issue exists with running jobs.
Solutions

• We don’t have any here!

• We are looking forward to trying an upcoming experimental feature in 8.1.6 which allows the startd to start a new job while the current job is staging out.

  • Allows us to better throttle stageout without wasting CPU.

• Reduces the number of services involved in the job lifetime - hope to reduce the surface area for bugs.
Conclusions

• CMS users don’t use DAGMan, they use CRAB3.
  • However, CRAB3 happens to rely on DAGMan for task management.
  • DAGMan automates the mundane tasks for us.
  • Ultimately, the biggest waste of resources was making physicists babysit jobs; wasted CPU was secondary.

• We’ve worked closely with the HTCondor team to iron out the biggest kinks.
  • While I focussed on the interesting problems, I left out quite a few mundane bug fixes the team delivered for CMS by the HTCondor team.
  • DAGMan and the python bindings have been hopefully made better for the entire community.
  • Quite a bit of work left to do!
Future Work

• There’s an immense amount of hackery around our kill/resubmit procedure; we would prefer to communicate directly with the DAGMan process.
  
  • We almost certainly don’t handle all the edge cases with killing the post-job.

• Our most pressing problem is the handling of the post-jobs.
  
  • The current best idea is to have a post-job return code that indicates DAGMan should re-run the post-job at a later time.

• Users hate “stuck” jobs; we want to do a better job of providing feedback by allowing condor_tail or updating the job ad with the # of events processed.