Opportunistic Computing at Notre Dame

Expanding to OSG-connect

Kevin Lannon, Matthias Wolf, and Anna Woodard

June 02, 2014

Motivation

- Our analysis workflow on the ND T3 takes 3 weeks
  - using all of our T3, around 900 cpus

  ⇒ would like to have a shorter analysis turnaround time

- Idle cpus at ND available through a condor queue

  ⇒ why not use these several 1000 unused cpus?

Challenges of opportunistic computing

On the ND campus-wide condor queue:

- heterogeneous environment
  - can only expect a RedHat release (or derivative thereof)
  - no OSG environment
  - no CMS software
- uncertain computing time available
  - cluster owners' jobs evict opportunistic jobs
  - users compete for open slots

How do we run our jobs?

CC tools to the rescue

The Cooperative Computing lab at Notre Dame provides:

chirp	userland file server - no root access needed - globus authentication - accesses hadoop directly or via FUSE
parrot	allows to transparently access CVMFS - no root access needed
work queue	framework for workload distribution - no cluster customization required - runs on single machines, condor, slurm, PBS, SGE, …

Opportunistic computing at ND

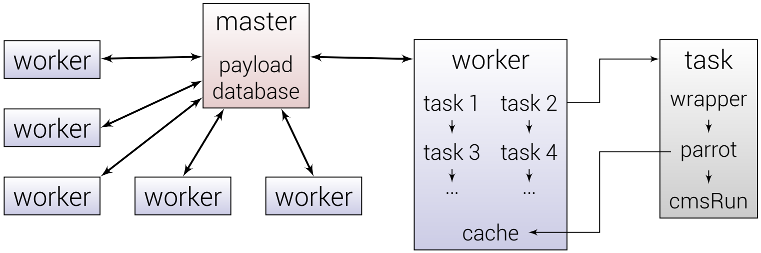
Summer 2013:

- ad-hoc modification to CRAB2 to use parrot_run
- required a lot of steps from the user
- successfully ran several thousand jobs on non-CMS, non-OSG nodes
    - solved a lot of parrot issues doing this

Fall 2013:

- got acquainted with work_queue
    - separation of resource and task management
        - uses workers to run several jobs consecutively
        - allows to preserve CVMFS cache over several jobs
        - can submit workers at different clusters
- started work on *lobster*

Work queue / *lobster* at ND



Master

- started by user
- tracks workers
- tracks tasks/payload

Worker

- submitted by user
- connects to master
- runs tasks
  - in sequence
  - in parallel
- provides cache
  - shared by tasks
  - reused by tasks

Task

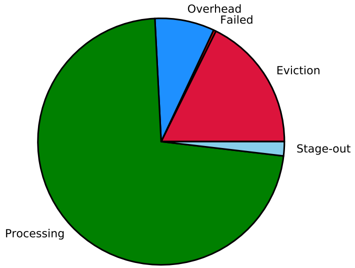
- runs a wrapper
- finds source for CMSSW
- sets up working environment
- executes cmsRun

Current status at ND

- Running on several thousand cores with no CMS/OSG customization
  - using the squid of our T3
- Caching as much as possible on the worker / node
  - everything from CVMFS
  - sandbox and supporting executables
- Extended our cluster by an order of magnitude in size
  - penalty for running opportunistically small

Total job time breakdown:

Overhead	time up to 1st event
Processing	from 1st event to stage-out
Eviction	lost time from killed workers



Running on OSG-connect

Started playing around with OSG-connect:

- master running at ND
- workers submitted via OSG login node
  - requires a copy of cctools on the login node
    - otherwise mis-match of worker and master protocol versions
  - use as many local resources as possible
    - checking for CMS CVMFS
    - guessing proxy server for parrot, if needed
- overall: little customization needed

Current status on OSG-connect

- Use parrot if site-config in CVMFS not present
  - lots of failed jobs relying on “obscure” site-config
  - provide “bare-bones” site-config for xrootd
  - subject to improvement
- Local proxy usage unclear if CERN listed as fallback
- Jobs completed on:

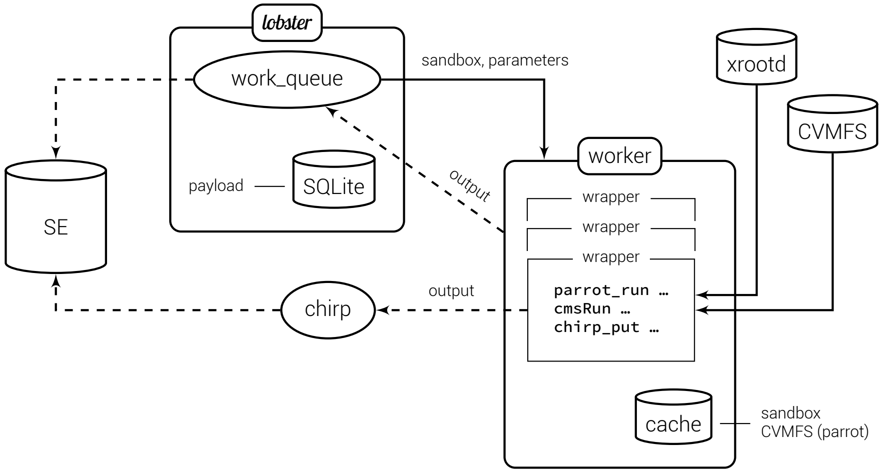
Site	CMS CVMFS	used parrot	used proxy	jobs
unl.edu	yes	no	n/a	116
smu.edu	no	yes	yes	16
swt2.uta.edu	yes	yes	unclear	42
Caltech	yes	no	n/a	13
smu.edu	no	yes	yes	19
uconn.edu	yes	yes	yes	32
mwt2.org	no	yes	unclear	21
aglt2.org	unclear	yes	yes	1

Open issues

- Slow connectivity
  - spooling sandbox takes a lot of time
  - need caching of sandbox
- CVMFS site-config confusion
  - need to reliably test for correctness of site-config
  - need a good default
- Reliable proxy configuration
  - similar problems as with site-config
- Basic library requirements
  - need to ship older libssl, …

Backup

Lobster workflow


lobster
work_queue
payload
SQLite
sandbox, parameters
output
SE
xrootd
CVMFS
worker
wrapper
wrapper
wrapper
parrot_run …
cmsRun …
chirp_put …
output
chirp
cache
sandbox
CVMFS (parrot)