Using non-CMS resources (assuming local batch interface)

Dirk Hufnagel, FNAL

Bologna Meeting, 18.09.2014

Use case

- We have a resource we are allowed to use, but that resource

  a) only has a locally usable batch system to run jobs
  b) does not provide a local CMS environment
  for our jobs to function (software / services)

- Caveat: If the resource has a grid or cloud interface, point a is not relevant, but point b usually still would be

Problems

- This immediately leads to two separate problems
  - Submission challenge:

    How do we get our jobs onto this resource ?
    (in a way that integrates well into our WM systems)

    Caveat: Grid or cloud interfaces work as-is

  - Runtime challenge:

    How do we get our jobs to run ?
    (given no local CMS software and services and
    no local support to provide them on a system
    wide basis)

Submission challenge

- No problem if grid interface or cloud interface exists. For the latter, it depends on the implementation details, but OpenStack and EC2 are both supported directly by glideinWMS (future relevant ones would also be supported this way presumably)

- If the resource only has local batch submissions (the HPC resources we used do or are only starting to play with grid interfaces and then not expose all their functionality), we have an issue.

- For transparent use in CMS, access needs to be integrated into glideinWMS.

This is where BOSCO came in

BOSCO

- BOSCO is a tool to allow researchers to access remote accounts at different computing centers directly from their desktops via ssh tunnels. They would submit locally, the jobs would get routed to the remote resource, jobs could be tracked from the local desktop and output could be retrieved.

  http://bosco.opensciencegrid.org/

- Sounds close to what we need

BOSCO

- Under the hood, BOSCO is essentially a complete HTCondor installation on the local desktop that connects via ssh to the remote resource, with some small code setup needed at the remote end to be able to interface with the batch system

- Now, glideinWMS is also mostly HTCondor under the hood, so this is interesting

- So what we did is to take the ssh tunneling used by BOSCO (which was an undocumented HTCondor feature at the time) and made it usable in a glideinWMS setup to submit pilots from the factory.

BOSCO

- The only piece we directly use from BOSCO right now is the deployment instructions for setting up the code at the remote site to interface with the local batch system
- Once this is setup, access is directly through the glideinWMS factory, with an endpoint configured in the factory that can submit pilots through an ssh tunnel.

  (like for other grid or cloud entries)

BOSCO : Security

- For accessing the remote resource through an ssh tunnel, one has to have an ssh private key that logs into a production account at the remote end

- At the moment the key is stored in the factory, which is not optimal, as the factory is supposed to be open and usable by many users, even many VOs

- We work around it right by having dedicated glideinWMS setups just for BOSCO style submissions, eventually ssh key management will move into the frontend.

BOSCO : deployment

- Eventually we want to automate the code setup at the remote resource for interfacing to the batch system

- Convenience feature really, as it is a one-time task for each remote resource.

Runtime challenge

- The CMS software needs both code, some general software environment describing the local resources and how to interact with them and services to function.
- Code: usually provided via cvmfs
- SITECONF: config file read from standardized place in the local filesystem describing how jobs can read data, how they can stageout data, what the local proxy servers to access information over http (for cvmfs and for conditions data) are etc
- At a non-CMS site where we are one use among many, there is little chance to make modifications system wide to provide these.

Runtime challenge

- I mentioned services before and by that I mean the proxy servers for accessing information over http

- We **can** live without them locally, but that means using some proxy servers that are in a non-local network (at a close CMS site), decreasing both the local job efficiency due to added latency and increasing the offsite network bandwidth use.

- This is an efficiency optimization to first order, but can become critical depending on the scale of CMS work at the non-CMS resource (overloading networks etc).

Parrot wrapper script

- GlideinWMS pilots support validation scripts
- Usually these are to make sure that the local environment **can** in principle run payloads, no point in even trying if it'll eventually fail
- But it can support more and that's where the parrot wrapper script comes in
- Parrot is a virtual file system that allows to mount (remote) file systems in user space

http://ccl.cse.nd.edu/software/parrot/

Parrot wrapper script

- The glideinWMS parrot wrapper script we use runs inside the pilot and (among a few other things) wraps the payload into a parrot shell

- It provides the SITECONF (configuration is stored in the factory) and also mounts cvmfs (only if needed), so the payload can access the CMS code

- For non-CMS cpu-only resources with no local proxy servers a SITECONF would configure things to read data exclusively remotely via xrootd and stageout to a close CMS site (FNAL or CERN for instance) and also configure for FNAL or CERN proxy servers

Parrot wrapper script

- There is support for auto-detecting local proxy servers and injecting them into the SITECONF, but this is still somewhat experimental

- There are also concerns about the dimensioning of the local proxy servers. We don't want to kill them.

- This feature needs some work to be generally useful. It's active at the moment for using US grid sites opportunistically, but there we limit ourself to 1000 jobs per resource to avoid damage to the site infrastructure.